Grażyna Koba / Katarzyna Koba-Gołaszewska

informatyka 1-3

Podręcznik dla szkół ponadpodstawowych ▪ **Zakres podstawowy**

#2

Algorytmika i programowanie

Oznaczenie nieobowiązkowego materiału w związku z usunięciem odpowiednich treści z podstawy programowej zgodnie z rozporządzeniem MEN z 28.06.2024 r.

MiGra

Autorzy: Grażyna Koba, Katarzyna Koba-Gołaszewska

Konsultacja merytoryczno-dydaktyczna:
Marta Skała-Kowalczyk
Michał Łętowski
Marcin Mytych
Piotr Babij
Paweł Rogoziński

Bolesław Gremblewski

Redakcja i korekta: dr Halina Kubicka
Opracowanie graficzne: Roman Jankowski, Grażyna Koba, Renata Płaczkowska
Projekt okładki, layout: Studio Grafiki INCOLOR Roman Jankowski
Skład: Studio Grafiki INCOLOR Roman Jankowski, Redo Renata Płaczkowska

Podręcznik dopuszczony do użytku szkolnego przez ministra właściwego
do spraw oświaty i wychowania i wpisany do wykazu podręczników przeznaczonych
do kształcenia ogólnego do nauczania informatyki, na podstawie opinii rzeczoznawców:
mgr inż. Anny Koludo, mgr. Andrzeja Dajnowicza, dr hab. Magdaleny Trysińskiej.

Zakres kształcenia: podstawowy.
Etap edukacyjny: III.
Typ szkoły: liceum ogólnokształcące i technikum.
Rok dopuszczenia: 2022.

Numer ewidencyjny w wykazie: 1142/2/2022

Zastrzeżonych nazw firm i produktów użyto w tej książce wyłącznie w celu identyfikacji.

ISBN 978-83-964794-1-9

Copyright © by MIGRA Sp. z o.o., Wrocław 2022

Wydanie I

Fotografie: SSPL/Getty Images – str. 16 (sklep komputerowy), str. 18 (budyń); Grażyna Koba – str. 55 (warzywa); SSPL/Getty Images – str. 71 (choinka), str. 73 (bank), str. 79 (ciężarówka), str. 96 (monitory); Grażyna Koba – str. 130 (warzywa); SSPL/Getty Images – str. 146 (choinka), str. 148 (bank), str. 153 (ciężarówka), str. 170 (monitory), str. 197 (rys. 1.), str. 200 (sklep), str. 215 (stos książek), str. 241 (kłódka), str. 263 (magazyn), str. 265 (monety), str. 273 (króliki); Grażyna Koba – str. 296 (ogród); Marcin Mytych – str. 296 (rys. 2.), str. 297 (tabela 1.), str. 298 (rys. 4. i 5.), str. 299 (rys. 6.), str. 305 (rys. 2. i 3.); SSPL/Getty Images – str. 305 (okno); Marcin Mytych – str. 306 (rys. 1., 2. i 3.); SSPL/Getty Images – str. 307 (głośnik); Marcin Mytych – str. 309 (rys. 1.); SSPL/Getty Images – tylna okładka.

Wydawnictwo: MIGRA Sp. z o.o.
ul. Świeradowska 51/57
50-559 Wrocław
tel.: 71 75 06 230
faks: 71 75 06 235
e-mail: biuro@migra.pl

Spis treści

Rozdział X Systemy liczbowe 5

Temat 50. System dwójkowy 6
Temat 51. System szesnastkowy 11

Rozdział XI Rozwiązywanie problemów i programowanie 15

Temat 52. Sytuacje problemowe i określanie specyfikacji zadania 16
Temat 53. Techniki algorytmiczne i programowanie 23

Rozdział XII Programowanie w języku C++ 33

Temat 54. Środowisko programistyczne języka C++, stosowanie zmiennych i wyprowadzanie wyników 34
Temat 55. Algorytmy z warunkami w języku C++ 47
Temat 56. Sprawdzanie poprawności danych i zapisywanie rozwiązania problemu w języku C++ 53
Temat 57. Instrukcje warunkowe zagnieżdżone i instrukcja wyboru 58
Temat 58. Algorytmy iteracyjne w języku C++ 63
Temat 59. Instrukcje iteracyjne zagnieżdżone w języku C++ 69
Temat 60. Stosowanie instrukcji iteracyjnych `for`, `while` i `do ... while` w języku C++ 73
Temat 61. Funkcje zwracające wartość w języku C++ 79
Temat 62. Funkcje niezwracające wartości w języku C++ 87
Temat 63. Deklarowanie i stosowanie tablicy w języku C++ 91
Temat 64. Wykonywanie operacji na elementach tablicy w języku C++ 96
Temat 65. Dane tekstowe w języku C++ 101

Rozdział XIII Programowanie w języku Python 107

Temat 66. Środowisko programistyczne języka Python, stosowanie zmiennych i wyprowadzanie wyników 108
Temat 67. Algorytmy z warunkami w języku Python 122
Temat 68. Sprawdzanie poprawności danych i zapisywanie rozwiązania problemu w języku Python 128
Temat 69. Instrukcje warunkowe zagnieżdżone i z klauzulą `elif` w języku Python 133
Temat 70. Algorytmy iteracyjne w języku Python 138
Temat 71. Instrukcje iteracyjne zagnieżdżone w języku Python 144
Temat 72. Stosowanie instrukcji iteracyjnych `for` i `while` w języku Python 148
Temat 73. Funkcje zwracające wartość w języku Python 153
Temat 74. Funkcje niezwracające wartości w języku Python 161
Temat 75. Definiowanie i stosowanie listy w języku Python 165
Temat 76. Wykonywanie operacji na elementach listy w języku Python 170
Temat 77. Dane tekstowe w języku Python 175

Rozdział XIV Wybrane algorytmy i techniki algorytmiczne 181

Temat 78. Wybrane algorytmy na tekstach 182
Temat 79. Wyszukiwanie wzorca w tekście metodą naiwną 187
Temat 80. Szyfrowanie tekstu metodą podstawieniową 191
Temat 81. Szyfrowanie tekstu metodą przestawieniową 195
Temat 82. Algorytmy porządkowania liczb 200
Temat 83. Metoda połowienia 205
Temat 84. Metoda zachłanna 209
Temat 85. Rekurencja a iteracja 213
Temat 86. Wybrane algorytmy na liczbach 220
Temat 87. Działania na ułamkach zwykłych z wykorzystaniem NWD i NWW 226

Rozdział XV Programowanie wybranych algorytmów 229

Temat 88. Programowanie wybranych algorytmów na tekstach 230
Temat 89. Programowanie algorytmu wyszukiwania wzorca w tekście metodą naiwną 235
Temat 90 Programowanie algorytmu szyfrowania tekstu metodą podstawieniową 241
Temat 91. Programowanie algorytmu szyfrowania tekstu metodą przestawieniową 246
Temat 92. Programowanie algorytmów porządkowania 252
Temat 93. Programowanie algorytmu wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia 258
Temat 94. Programowanie algorytmów zachłannych 263
Temat 95. Programowanie algorytmów rekurencyjnych i iteracyjnych 269
Temat 96. Programowanie zamiany reprezentacji liczb między pozycyjnymi systemami liczbowymi 276
Temat 97. Programowanie wybranych algorytmów na liczbach 282
Temat 98. Programowanie działań na ułamkach zwykłych z wykorzystaniem NWD i NWW 288

Rozdział XVI Projekty – elementy robotyki z wykorzystaniem Arduino 293

Temat 99. Sterowanie nawadnianiem ogrodu – projekt 294
Temat 100. Projekt „Zamknij okno” 304
Temat 101. Projekt „Wycisz radio” 306
Temat 102. Budowanie robota – projekt 308

Uwaga:

W przypadku pytań, poleceń, ćwiczeń i zadań zawartych w podręczniku, wymagających udzielenia przez ucznia pisemnej odpowiedzi, nie należy umieszczać jej w podręczniku.



Rozdział X

Systemy liczbowe

Temat 50. System dwójkowy

Temat 51. System szesnastkowy

Temat 50. System dwójkowy

1. **Wprowadzenie do systemów liczbowych**
2. **Wartość liczby zapisanej w systemie dwójkowym**
3. **Wyznaczanie rozwinięcia dwójkowego liczby dziesiętnej**

Warto powtórzyć

1. Na czym polega główna idea maszyny von Neumanna?
2. W jakiej postaci są zapisane rozkazy procesora?
3. W jaki sposób w matematyce zapisuje się rozwinięcie dziesiętne liczby? Wyjaśnij na przykładzie liczby 9871.

1. Wprowadzenie do systemów liczbowych

Kombinacje zer i jedynek wystarczają matematycznie do zapisania dowolnej liczby. Dlatego w komputerach wykorzystuje się system binarny, czyli dwójkowy, który ma do dyspozycji dwie cyfry: 0 i 1.

Dwa podstawowe rodzaje **systemów liczbowych** to:

- **system addytywny** – zapisaną w nim **liczbę** odczytuje się jako sumę wartości jej poszczególnych **cyfr**; przykładem jest rzymski system liczbowy;
- **system pozycyjny** – o wartości cyfry decyduje jej miejsce w zapisie liczby; obecnie najczęściej stosowanymi pozycyjnymi systemami liczbowymi są: system dziesiętny, używany w życiu codziennym, oraz system dwójkowy, stosowany w systemach komputerowych.

System liczbowy S

Sposób zapisywania i odczytywania liczb.

Podstawą systemu dziesiętnego jest liczba 10, będąca jednostką drugiego rzędu. Jednostką trzeciego rzędu jest liczba 100, czwartego rzędu – liczba 1000 itd. Jednostką pierwszego rzędu jest liczba 1. Ta sama cyfra ma w systemie dziesiętnym różne znaczenie, w zależności od miejsca, na którym stoi w zapisie liczby.

Przykład 1. Pozycyjne systemy liczbowe

W liczbie **1476** zapisanej w systemie dziesiętnym cyfra 6 ma wartość 6, ponieważ znajduje się w kolumnie jedności; cyfra 7 oznacza 70, ponieważ znajduje się w kolumnie dziesiątek; cyfra 4 oznacza 400, ponieważ znajduje się w kolumnie setek, a cyfra 1 oznacza 1000, ponieważ znajduje się w kolumnie tysięcy.

Czyli: $1476 = 1 \cdot 1000 + 4 \cdot 100 + 7 \cdot 10 + 6 \cdot 1$.

Jeżeli ponumerujemy cyfry, zaczynając numerowanie od zera w kolejności od prawej strony, to zauważymy, że tak nadane numery cyfr odpowiadają kolejnym wykładnikom potęgi liczby 10 przy danej cyfrze:

3 2 1 0

$\mathbf{1476} = \mathbf{1} \cdot 10^3 + \mathbf{4} \cdot 10^2 + \mathbf{7} \cdot 10^1 + \mathbf{6} \cdot 10^0$

W systemie dziesiętnym używamy potęg liczby 10, podczas gdy np. w systemie trójkowym używamy potęg liczby 3, a w siedemnastkowym – potęg liczby 17. Potęgowaną liczbę nazywamy **podstawą systemu liczbowego**.

W systemie dziesiętnym do zapisywania liczb używamy dziesięciu cyfr. W systemie trójkowym są to trzy cyfry (0, 1, 2), a w systemie siedemnastkowym – siedemnaście (problem oznaczenia cyfr o wartościach większych niż 9 pokażemy dalej na przykładzie systemu szesnastkowego).

Do zapisania liczby w systemie pozycyjnym o podstawie p konieczny jest zbiór cyfr $\{0, 1, ..., p - 1\}$.

W systemie pozycyjnym o podstawie $\boldsymbol{p}$ liczba $\boldsymbol{n}$-cyfrowa: $(\boldsymbol{a}_{n-1}\boldsymbol{a}_{n-2}... \boldsymbol{a}_1\boldsymbol{a}_0)_p$ ma rozwinięcie:

$$(\boldsymbol{a}_{n-1}\boldsymbol{a}_{n-2}... \boldsymbol{a}_1\boldsymbol{a}_0)_p = \boldsymbol{a}_{n-1} \cdot \boldsymbol{p}^{n-1} + \boldsymbol{a}_{n-2} \cdot \boldsymbol{p}^{n-2} + ... + \boldsymbol{a}_1 \cdot \boldsymbol{p}^1 + \boldsymbol{a}_0 \cdot \boldsymbol{p}^0, \quad [1.]$$

gdzie: $\boldsymbol{a}_{n-1}, \boldsymbol{a}_{n-2}, ..., \boldsymbol{a}_1, \boldsymbol{a}_0$ to kolejne cyfry danej liczby,
$\boldsymbol{p}$ – podstawa systemu liczbowego.

2. Wartość liczby zapisanej w systemie dwójkowym

Sposób reprezentacji danych stosowany w komputerach pokażemy na przykładzie liczb naturalnych.

Jeśli przykładowa liczba ma postać: 11110, to taki zapis można zinterpretować jako jedenaście tysięcy sto dziesięć w systemie dziesiętnym lub jako liczbę zapisaną w systemie dwójkowym. Z tego powodu musimy uściślić zapis, przez określenie podstawy systemu liczbowego. Podstawę zapisujemy w indeksie dolnym na końcu liczby.

Na przykład:
11110_2 – liczba w systemie dwójkowym,
20122_3 – liczba w systemie trójkowym,
11110, 20122 – liczby w systemie dziesiętnym.

Ponieważ na co dzień posługujemy się systemem dziesiętnym, przyjęło się, że liczby bez podstawy zapisanej na końcu to właśnie liczby dziesiętne. Czyli np. zamiast pisać 1476_{10}, piszemy po prostu 1476.

Aby obliczyć wartość dziesiętną liczby zapisanej w systemie dwójkowym, należy dokonać rozwinięcia liczby według wzoru [1.], przyjmując jako podstawę systemu liczbę 2.

Ćwiczenie 1. Obliczamy wartość liczby dwójkowej

Posługując się wzorem [1.], sprawdź, czy wartość liczby 11110_2 wynosi 30.

Wskazówka: Ponumeruj cyfry, zaczynając numerowanie od zera w kolejności od prawej strony podobnie jak w przykładzie 1.:

4 3 2 1 0
1 1 1 1 0$_2$

Przykład 2. Obliczanie wartości dziesiętnej liczby zapisanej w systemie dwójkowym

Ponieważ $0 \cdot 2^n = 0$ i $1 \cdot 2^n = 2^n$, w praktyce wystarczy zsumować tylko te potęgi dwójki, które znajdują się przy cyfrach 1, np.:

11 10 9 8 7 6 5 4 3 2 1 0
101101110110$_2$ = **1 0 1 1 0 1 1 1 0 1 1 0**$_2$ =
$= 2^{11} + 2^9 + 2^8 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1 =$
$= 2048 + 512 + 256 + 64 + 32 + 16 + 4 + 2 = 2934.$

W systemach komputerowych dane zapisuje się przy użyciu liczb bitów będących potęgami liczby 2. Do zapisu liczb używa się: 8 bitów – tzw. krótkie słowo (1 bajt), 16 bitów – tzw. słowo, 32 bitów – tzw. długie słowo (podwójne słowo), 64 bitów – tzw. poczwórne słowo. Liczba 2934 mogłaby więc być zapisana jako słowo:

MSB LSB
0000101101110110_2 (dopełniona do 16 cyfr zerami z lewej strony).

W zapisie binarnym skrajna lewa wartość to tzw. **najbardziej znaczący bit** (z ang. MSB – ***Most Significant Bit***), skrajna prawa – to tzw. **najmniej znaczący bit** (z ang. LSB – ***Least Significant Bit***).

Ćwiczenie 2. Obliczamy wartości liczb dwójkowych

Oblicz wartości dziesiętne liczb 1000000000_2 i 11111111_2.

3. Wyznaczanie rozwinięcia dwójkowego liczby dziesiętnej

Rozwinięcie dwójkowe liczby dziesiętnej można wyznaczyć na kilka sposobów. Przedstawimy jeden z nich.

Aby wyznaczyć postać dwójkową (zwaną rozwinięciem dwójkowym lub binarnym) liczby dziesiętnej, powtarzamy operację dzielenia z resztą tej liczby przez dwa i zapisujemy wszystkie reszty z dzieleń, aż do otrzymania ilorazu równego zero.

Rozwinięcie dwójkowe liczby to zapisane kolejno (**począwszy od ostatniej**) reszty z dzieleń.

Ćwiczenie 3. Obliczamy rozwinięcie dwójkowe liczby zapisanej w systemie dziesiętnym

Sprawdź, czy rozwinięcie dwójkowe liczby 1476 to 10111000100_2. Utwórz tabelę podobną do pokazanej na rysunku 1. i wykonaj brakujące obliczenia według wzoru.

Liczba	Reszta	Komentarz
1476	0	$1476 = 2 \cdot 738 + 0$
738	0	$738 = 2 \cdot 369 + 0$
...	...	...
5	1	$5 = 2 \cdot 2 + 1$
2	0	$2 = 2 \cdot 1 + 0$
1	1	$1 = 2 \cdot 0 + 1$
0		iloraz równy 0

Rys. 1. Wzór tabeli – ćwiczenie 3.

Ćwiczenie 4. Obliczamy rozwinięcia dwójkowe liczb zapisanych w systemie dziesiętnym

1. Oblicz rozwinięcia dwójkowe liczb 255 i 1024.
2. Sprawdź, jaki rysunek utworzą cyfry 1, gdy zapiszesz rozwinięcia dwójkowe liczb: 16, 16, 56, 124, 254, 16, 16, 56, tak aby ich środkowe cyfry znajdowały się jedna pod drugą.

Wskazówka: Polecenie z punktu 2. najlepiej wykonać na kartce w kratkę albo w tabeli. Można też w rozwinięciach dwójkowych zastąpić jedynki literami x, a zer nie wpisywać.

Warto zapamiętać

- Wszelkie informacje (np. liczby, znaki alfanumeryczne) przechowywane w komputerze zapamiętywane są w postaci binarnej.
- Systemem pozycyjnym nazywamy taki sposób prezentowania liczb, w którym wartość cyfry zależy od jej pozycji w ciągu cyfr określającym liczbę.
- Rozwinięcie dwójkowe (inaczej binarne) liczby to zapisanie liczby dziesiętnej w systemie dwójkowym, czyli za pomocą zer i jedynek.

Pytania i polecenia

1. Omów pozycyjne systemy liczbowe na przykładzie systemu dziesiętnego.
2. Jakie znasz niepozycyjne systemy liczbowe?
3. Dlaczego powszechnie używanym systemem liczbowym jest system dziesiętny?
4. Czy istnieje system jedynkowy? Jak wyglądałby zapis liczby w tym systemie?
5. W jaki sposób wyznaczamy rozwinięcie dwójkowe liczby dziesiętnej? Pokaż na przykładzie.
6. Jak oblicza się wartość dziesiętną liczby binarnej?

Zadania

1. Oblicz wartość dziesiętną następujących liczb: 1110011010100_2, 3210_4, 444_5.
2. Oblicz rozwinięcie dwójkowe liczby określającej bieżący rok kalendarzowy.
3. Oblicz rozwinięcie dwójkowe liczby określającej twój wiek.
4. Oblicz rozwinięcie dwójkowe liczb: 2222, 500, 100.

Dla zainteresowanych

5. Sprawdź, jaki rysunek utworzą cyfry 1, gdy zapiszesz rozwinięcia dwójkowe liczb: 41702, 45705, 45801, 43657, 43657, 42633, 42633, 41606, tak aby ich kolejne cyfry znajdowały się jedna pod drugą. Skorzystaj ze wskazówki umieszczonej w ćwiczeniu 4.

Temat 51. System szesnastkowy

1. Wartość liczby zapisanej w systemie szesnastkowym
2. Wyznaczanie rozwinięcia szesnastkowego liczby dziesiętnej
3. Zależność między systemem dwójkowym a szesnastkowym

Warto powtórzyć

1. Czym charakteryzuje się pozycyjny system liczbowy, a czym addytywny?
2. Jaką wartość dziesiętną ma liczba 101_2? Omów sposób wykonania obliczeń.
3. Jakie jest rozwinięcie dwójkowe liczby 10? Omów sposób wykonania obliczeń.

1. Wartość liczby zapisanej w systemie szesnastkowym

W praktyce posługiwanie się systemem dwójkowym jest bardzo niewygodne – zapis w nim jest długi i łatwo o pomyłkę (np. o pominięcie zera lub jedynki). Często więc w informatyce, zamiast systemu dwójkowego, wykorzystuje się system szesnastkowy, w którym do dyspozycji mamy 16 cyfr. Oprócz cyfr od 0 do 9 pojawiają się następujące cyfry:

A – o wartości 10, B – 11, C – 12, D – 13, E – 14, F – 15.

Przykłady liczb w systemie szesnastkowym: 100_{16}, AB_{16}, $FFFF_{16}$, $400D_{16}$.

Aby obliczyć wartość liczby zapisanej w systemie szesnastkowym, używamy tego samego wzoru, z którego korzystaliśmy przy obliczaniu wartości liczby zapisanej w systemie dwójkowym, przyjmując za podstawę systemu liczbę 16:

$(a_{n-1}a_{n-2}\ldots a_1a_0)_p = a_{n-1}\cdot p^{n-1} + a_{n-2}\cdot p^{n-2} + \ldots + a_1\cdot p^1 + a_0\cdot p^0$,

gdzie: $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ to kolejne cyfry danej liczby,

p – podstawa systemu liczbowego.

Przykład 1. Obliczanie wartości liczby zapisanej w systemie szesnastkowym

3 2 1 0

$\mathbf{A\,B\,C\,D}_{16} = \mathbf{A}\cdot 16^3 + \mathbf{B}\cdot 16^2 + \mathbf{C}\cdot 16^1 + \mathbf{D}\cdot 16^0 =$

$= \mathbf{10}\cdot 16^3 + \mathbf{11}\cdot 16^2 + \mathbf{12}\cdot 16^1 + \mathbf{13}\cdot 16^0 =$

$= \mathbf{10}\cdot 4096 + \mathbf{11}\cdot 256 + \mathbf{12}\cdot 16 + \mathbf{13}\cdot 1 = 40960 + 2816 + 192 + 13 = 43981$

Ćwiczenie 1. Obliczanie wartości liczb zapisanych w systemie szesnastkowym

Oblicz wartość dziesiętną liczb szesnastkowych: 100_{16}, $FFFF_{16}$, $C000_{16}$.

2. Wyznaczanie rozwinięcia szesnastkowego liczby dziesiętnej

Do zamiany liczby z systemu dziesiętnego na system szesnastkowy służy ten sam algorytm, z którego korzystaliśmy przy zamianie na system dwójkowy z tą tylko różnicą, że liczbę dzielimy przez podstawę systemu szesnastkowego. Podobnie jak w przypadku zamiany na system dwójkowy, wykonujemy dzielenia z resztą, aż do otrzymania ilorazu równego 0.

Ćwiczenie 2. Obliczamy rozwinięcie szesnastkowe liczby zapisanej w systemie dziesiętnym

Zamień liczbę dziesiętną 1476 na liczbę szesnastkową. Utwórz tabelę podobną do przedstawionej na rys. 1. w ćwiczeniu 3. z tematu 50.

Wskazówka: Pamiętaj, że cyfry o wartościach powyżej 9 są reprezentowane przez litery.

Ćwiczenie 3. Obliczamy rozwinięcia szesnastkowe liczb zapisanych w systemie dziesiętnym

1. Podaj swój wiek w systemie szesnastkowym.
2. Oblicz rozwinięcia szesnastkowe liczb dziesiętnych 100 i 64.

3. Zależność między systemem dwójkowym a szesnastkowym

Zauważmy, że systemu szesnastkowego można używać zamiast systemu dwójkowego, ponieważ $2^4 = 16$, a więc jedna cyfra szesnastkowa odpowiada **dokładnie** czterem cyfrom dwójkowym.

W przypadku systemu dziesiętnego nie ma takiej zależności pomiędzy cyframi dziesiętnymi a cyframi dwójkowymi, bowiem liczba 10 nie jest potęgą całkowitą liczby dwa.

Aby zamienić liczbę szesnastkową na dwójkową, wystarczy sprawdzić wartość dziesiętną każdej cyfry szesnastkowej i zamienić ją na system dwójkowy, pamiętając o ewentualnym dopełnieniu zerami z przodu, tak by każda liczba dwójkowa odpowiadająca cyfrze szesnastkowej miała dokładnie cztery cyfry. Otrzymane rozwinięcia dwójkowe zapisujemy w tej samej kolejności, co odpowiadające im cyfry szesnastkowe.

Przykład 2. Zamienianie liczby zapisanej w systemie szesnastkowym na liczbę dwójkową

Zapiszemy liczbę $5C4_{16}$ w systemie dwójkowym. Aby przyspieszyć zamianę, można posłużyć się wartościami podanymi w tabeli 1.

$5_{16} = 5_{10} = 10_{12} = 0101_2$ (dopełniamy do 4 cyfr jednym zerem z przodu)
$C_{16} = 12_{10} = 1100_2$
$4_{16} = 4_{10} = 100_2 = 0100_2$ (dopełniamy do 4 cyfr jednym zerem z przodu)
Zatem:

$$\begin{matrix} & \mathbf{5} & \mathbf{C} & \mathbf{4} \\ 5C4_{16} = & 0101 & 1100 & 0100_2 \end{matrix}$$

Wartość dziesiętna	Wartość szesnastkowa	Wartość dwójkowa
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Tabela 1. Wartości dziesiętne, szesnastkowe i dwójkowe liczb od 0 do 15

Zasadę, zgodnie z którą jedna cyfra szesnastkowa odpowiada czterem cyfrom dwójkowym, wykorzystujemy także przy zamianie liczb z systemu dwójkowego na system szesnastkowy. Liczbę dwójkową dzielimy na czterocyfrowe grupy, zaczynając od prawej strony. Następnie obliczamy wartość dziesiętną każdej grupy i wpisujemy odpowiadającą jej cyfrę szesnastkową. Dla ułatwienia obliczeń można posłużyć się wartościami podanymi w tabeli 1.

Przykład 3. Zamienianie liczby zapisanej w systemie dwójkowym na liczbę szesnastkową

Zapiszemy liczbę $11000111010111010111011_2$ w systemie szesnastkowym.
$11|0001|1101|0111|0101|1011_2$
11 0001 1101 0111 0101 1011 (dzielimy na grupy)
3 1 13 7 5 11 (obliczamy wartość dziesiętną każdej grupy)
3 1 D 7 5 B (wpisujemy odpowiednią cyfrę szesnastkową)
$11000111010111010111011_2 = 31D75B_{16}$

Ćwiczenie 4. Zamienianie liczb między różnymi systemami

Zapisz następujące liczby w systemach dwójkowym, dziesiętnym i szesnastkowym:

a. $ABC_{16} = ?_{10} = ?_{2}$
b. $2331_{10} = ?_{16} = ?_{2}$
c. $10011110101011011110110_{2} = ?_{16} = ?_{10}$

Warto zapamiętać

- W systemie szesnastkowym do dyspozycji mamy 16 cyfr: cyfry od 0 do 9 oraz A – o wartości 10, B – 11, C – 12, D – 13, E – 14, F – 15.
- Aby wyznaczyć rozwinięcie szesnastkowe liczby dziesiętnej, korzystamy z tego samego algorytmu, co przy zamianie liczby dziesiętnej na system dwójkowy z tą tylko różnicą, że dzielimy przez podstawę systemu szesnastkowego.
- Aby obliczyć wartość liczby zapisanej w systemie szesnastkowym, używamy tego samego wzoru, z którego korzystaliśmy przy obliczaniu wartości liczby zapisanej w systemie dwójkowym, przyjmując za podstawę systemu liczbę 16.
- Cyfra szesnastkowa odpowiada dokładnie czterem cyfrom dwójkowym.

Pytania i polecenia

1. Jak obliczamy wartość liczby zapisanej w systemie szesnastkowym? Wyjaśnij na przykładzie.
2. Jak wyznaczyć rozwinięcie szesnastkowe liczby dziesiętnej? Wyjaśnij na przykładzie.
3. Dlaczego systemu szesnastkowego można używać zamiast systemu dwójkowego? Wyjaśnij na przykładzie (innym niż podany w temacie).

Zadania

1. Oblicz wartość następujących liczb szesnastkowych: ACE_{16}, 2000_{16}, 1111_{16}.
2. Oblicz rozwinięcie szesnastkowe następujących liczb: 640, 32678, 1024.

Rozdział XI

Rozwiązywanie problemów i programowanie

Temat 52. Sytuacje problemowe i określanie specyfikacji zadania

Temat 53. Techniki algorytmiczne i programowanie

Temat 52.

Sytuacje problemowe i określanie specyfikacji zadania

1. Sytuacje problemowe
2. Określanie specyfikacji zadania (problemu)

Warto powtórzyć

1. Jakie problemy dotyczące edycji tekstu można rozwiązać, wykorzystując edytor tekstu? Podaj dwa przykłady.
2. Omów przykładowe zadanie z matematyki, podając dane do zadania, szukane oraz metodę rozwiązania.

1. Sytuacje problemowe

Jesteś sprzedawcą komputerów. Zastanawiasz się, jak przygotować kalkulację ceny. Cena komputera zależy m.in. od ceny płyty głównej – jeśli cena płyty wzrośnie, trzeba podnieść cenę komputera. O jaki procent możesz podnieść cenę, aby nie stracić klienta?

Programy komputerowe umożliwiają rozwiązywanie problemów.



Gdy mamy do czynienia z **problemem**, szukamy sposobu rozwiązania (**algorytmu**), a następnie dobieramy odpowiednie narzędzie – **program komputerowy** (np. użytkowy lub **język programowania**).

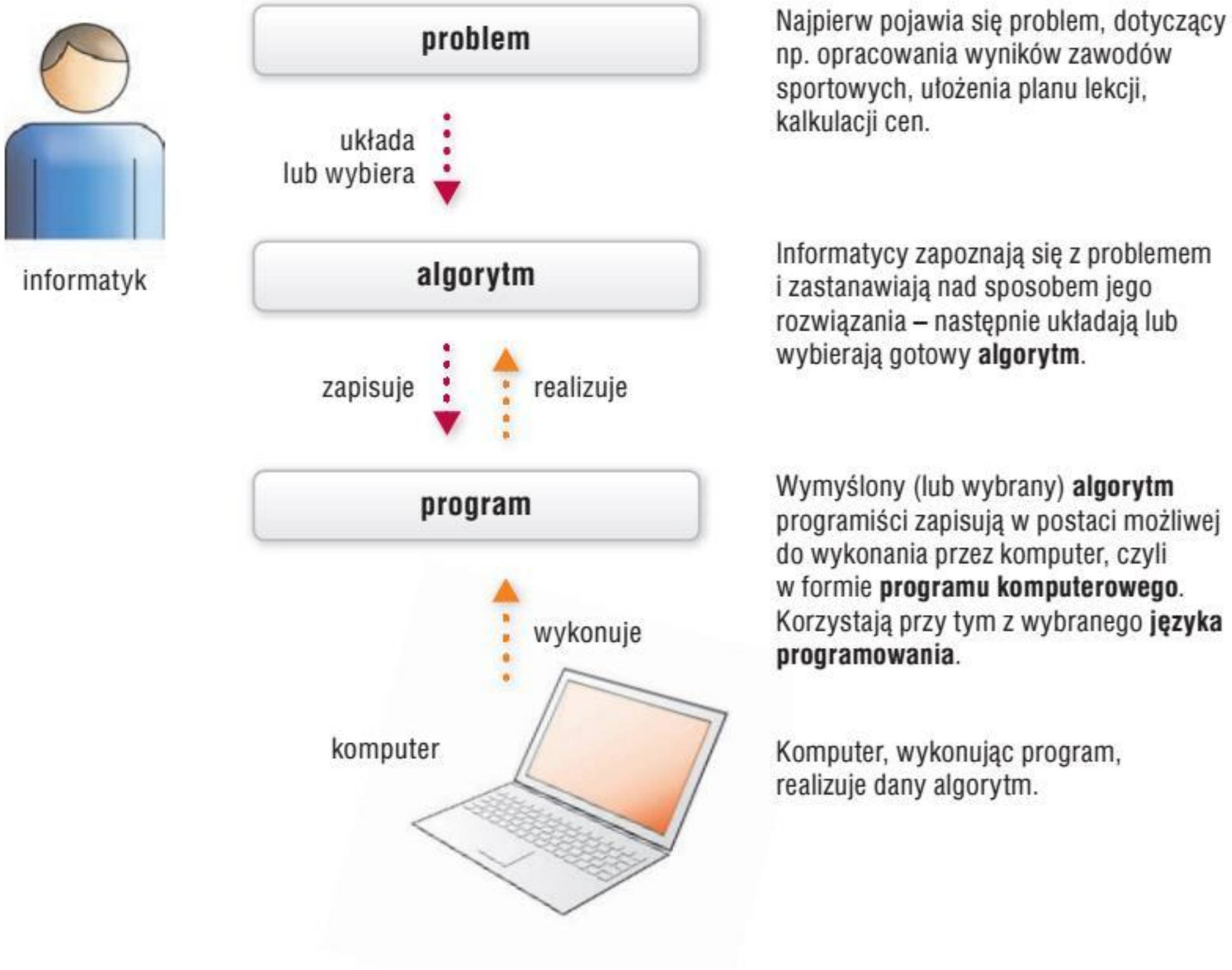
Najczęściej korzystamy z programów użytkowych (np. arkuszy kalkulacyjnych, edytorów tekstu lub grafiki, programów do odtwarzania muzyki), programów narzędziowych (np. antywirusowych, archiwizujących) i systemu operacyjnego. Nie jest dla nas ważne, w jaki sposób dany program został napisany. Wystarczy, że znamy jego działanie – niezbędne funkcje i możliwości. Zazwyczaj nie mamy w ogóle wglądu do „wnętrza" programu, czyli do tzw. **programu (kodu) źródłowego**.

Wybór algorytmu i sposób jego sformułowania zależą od problemu, który chcemy rozwiązać.

Na rysunku 1. pokazano uproszczony schemat postępowania w przypadku wyboru języka programowania do rozwiązania danego problemu.

A **Algorytm**

Uporządkowany i uściślony sposób rozwiązywania danego problemu, zawierający szczegółowy opis wykonywanych czynności w skończonej liczbie kroków.



Rys. 1. Relacje między problemem, algorytmem i programem komputerowym

Program komputerowy jest realizacją wybranego wcześniej algorytmu lub wielu algorytmów.

W rzeczywistości tworzenie programów jest złożonym procesem – tym bardziej, im trudniejszy jest problem. Skomplikowane zadania dzieli się na mniejsze części i często dla każdej z nich oddzielnie układa bądź dobiera algorytmy. W efekcie powstają duże projekty informatyczne, w których tworzeniu uczestniczy wiele osób, nie tylko programistów, ale również projektantów.

P **Program komputerowy** (program źródłowy, kod źródłowy)

Ciąg instrukcji języka programowania, realizujący algorytm.

Język programowania J

Specjalny język służący do pisania programów komputerowych. Jest on zbiorem określonych instrukcji i zasad składni.

Istniejące programy usprawnia się, opracowując ich kolejne wersje, oferujące nowe możliwości, które pozwalają rozwiązywać coraz więcej problemów. Ciągle powstają też nowe programy, dostosowane do zmieniających się potrzeb użytkowników.

Czasem tworzy się programy dla indywidualnych odbiorców (osób prywatnych, firm), jeśli mają oni wymagania użytkowe, których nie spełnia żaden z istniejących programów. Użytkownik może np. oczekiwać innych od oferowanych w gotowym programie sposobów wprowadzania danych (formularzy), odmiennych końcowych zestawień (raportów) czy dodatkowych wyliczeń. Możemy też wyobrazić sobie, że firma użytkownika to swoisty „magazyn osobliwości", zawierający artykuły, które trudno byłoby klasyfikować i przetwarzać według ogólnych kryteriów.

Nierzadko tworzy się też programy do zadań, które wprawdzie można wykonać za pomocą istniejącego oprogramowania, np. arkusza kalkulacyjnego, ale łatwiej i wygodniej jest napisać odrębny program. W nowym programie zadanie będzie wymagało jedynie kilku prostych instrukcji, podczas gdy w arkuszu kalkulacyjnym byłoby bardziej pracochłonne – wymagałoby większej liczby czynności, np. kopiowania tej samej formuły.

Przykład 1. Przykład sytuacji problemowej

W komputerze gromadzimy wiele danych. Pojawia się zatem problem ich porządkowania, np. w systemach zarządzania plikami: według dat utworzenia, według nazw lub rozszerzeń; w systemach baz danych: sortowanie artykułów według nazw, cen czy dat dostaw.

Aby użytkownik komputera mógł oglądać dane uporządkowane w wybranym przez siebie układzie, informatycy musieli wymyślić odpowiednie sposoby rozwiązania problemu (algorytmy sortowania). Dopiero potem mogli napisać programy do konkretnych zastosowań.

Ćwiczenie 1. Wybieramy algorytm do rozwiązania problemu

Jaki znany ci algorytm można zastosować do znalezienia największego wspólnego dzielnika (NWD) dla dwóch niezerowych liczb naturalnych?



W życiu codziennym pewne podobieństwo do algorytmów cechuje np. przepisy kulinarne, określające, z jakich składników i w jaki sposób należy przyrządzić potrawę. Przepisów nie można jednak nazwać algorytmami w sensie informatycznym, choćby z racji tego, że pojawiają się w nich określenia nie do końca precyzyjne, np. „gotować przez chwilę". Zależnie od tego, ile ta chwila będzie trwała, możemy otrzymać różne wyniki (np. budyń o różnej konsystencji).

W algorytmach informatycznych dane wejściowe powinny być precyzyjnie określone, a wszystkie operacje – dokładnie opisane. Dla tych samych danych powinniśmy otrzymać te same wyniki. W algorytmach nieinformatycznych wynik może zależeć od danych wejściowych i różnych czynników (np. od czasu gotowania potrawy).

2. Określanie specyfikacji zadania (problemu)

Postępowanie mające na celu rozwiązanie dowolnego problemu możemy podzielić na następujące etapy:

1. Sformułowanie zadania.
2. Określenie danych wejściowych.
3. Określenie celu, czyli wyniku.
4. Określenie metody rozwiązania, czyli wybór algorytmu.
5. Przedstawienie algorytmu w wybranej postaci: opisu słownego, listy kroków, schematu blokowego lub programu w wybranym języku programowania.
6. Analiza poprawności rozwiązania.
7. Testowanie rozwiązania dla różnych danych – ocena efektywności przyjętej metody.

W zadaniach z różnych dziedzin (np. z fizyki czy matematyki) określa się dane wejściowe i wynik (wyniki), a także związek pomiędzy danymi a wynikami, czyli warunki, jakie muszą spełniać wyniki – określamy w ten sposób specyfikację zadania. Potem szukamy odpowiednich rozwiązań, dobieramy wzory, twierdzenia, definicje. Podobnie na zajęciach z informatyki – opis sposobu rozwiązania problemu będziemy poprzedzać **specyfikacją zadania**.

Określając specyfikację zadania, warto nadać danym i wynikom nazwy, którymi będziemy się posługiwać w dalszych etapach rozwiązywania zadania.

S **Specyfikacja zadania** (problemu)

Szczegółowy opis zadania, w którym określa się dane wejściowe i oczekiwane wyniki oraz związek między danymi a wynikami (warunki, jakie muszą spełniać wyniki).

Przykład 2. Formułowanie zadania i określanie jego specyfikacji – przykłady

Zadanie 1: Oblicz wartość bezwzględną dowolnej liczby całkowitej.

Dane: dowolna liczba całkowita: *x*.

Wynik: wartość bezwzględna liczby *x* równa *w*.

Zadanie 2: Uporządkuj alfabetycznie rosnąco (od A do Ż) zbiór fikcyjnych nazwisk uczniów klas pierwszych fikcyjnej szkoły.

Dane: nieuporządkowany zbiór fikcyjnych nazwisk uczniów klas pierwszych.

Wynik: uporządkowany alfabetycznie zbiór fikcyjnych nazwisk uczniów klas pierwszych.

Zadanie 3: Sprawdź, czy dany wyraz (ciąg znaków składający się z liter) jest palindromem. Palindrom to wyraz, który czyta się tak samo od lewej do prawej i od prawej do lewej, np. *potop, kajak*.

Dane: dowolny wyraz.

Wynik: jeżeli wyraz jest palindromem, określenie „tak", w przeciwnym wypadku – określenie „nie".

Ćwiczenie 2. Piszemy specyfikację zadania

Napisz specyfikację do zadania: *Oblicz sumę liczb ujemnych i sumę liczb dodatnich w* n*-elementowym zbiorze liczb rzeczywistych różnych od zera*.

Sformułujemy teraz zadanie i specyfikację do sytuacji problemowej sprzedawcy komputerów podanej na początku tematu.

Przykład 3. Formułowanie zadania i określanie jego specyfikacji

Zadanie: Oblicz i wyprowadź aktualną cenę komputera w zależności od wartości procentowego wzrostu ceny płyty głównej (*w*). Jeśli wartość *w* mieści się w przedziale ⟨0%; 10%⟩, cenę komputera pozostaw bez zmian; jeśli mieści się w przedziale (10%; 20%⟩, cenę komputera zwiększ o 5% nowej ceny płyty głównej; jeśli $w > 20\%$, wyprowadź komunikat: „za wysoka cena".

Dane: cena komputera *cenak*, cena płyty głównej *cenap*, wartość *w* – procent, o jaki wzrosła cena płyty głównej. Przyjmujemy założenia: $cenak > 0$, $cenap > 0$, $w \geq 0$.

Wynik: aktualna wartość ceny komputera *cenak* (jeśli $0 \leq w \leq 20$) lub napis „za wysoka cena" (jeśli $w > 20$).

Przykład 4. Określenie metody rozwiązania problemu sprzedawcy komputerów, czyli wybór algorytmu

Przyjmiemy następujący algorytm kalkulacji ceny komputera:

a. jeśli *w* mieści się w przedziale ⟨0%; 10%⟩, cena komputera pozostaje bez zmian (zmniejszasz swój zysk, aby zadowolić klienta);
b. jeśli *w* mieści się w przedziale (10%; 20%⟩, cenę sprzedawanego komputera zwiększasz o 5% nowej ceny płyty głównej (przewidujesz, że klient zaakceptuje wzrost ceny komputera);
c. jeśli $w > 20\%$, nie ustalasz ceny tego komputera (szukasz innego producenta płyty głównej) i wyprowadzasz komunikat: „za wysoka cena".

Przykład 5. Projektowanie rozwiązania i wybór narzędzi komputerowych

W przyjętym w przykładzie 4. rozwiązaniu występują sytuacje warunkowe. Kolejne kroki algorytmu będą zatem realizowane zależnie od spełnienia warunku.

Zaczynamy od sprawdzenia, czy $0 \leq w \leq 10$ (gdzie *w* oznacza procent, o jaki wzrośnie cena płyty głównej). Jeśli warunek jest spełniony, wyprowadzamy wynik: aktualną niezmienioną cenę komputera (*cenak*).

W przeciwnym wypadku sprawdzamy, czy $w \leq 20$. Jeśli warunek jest spełniony, obliczamy aktualną cenę płyty głównej

$$cenap = cenap \cdot (1 + \frac{w}{100})$$

i aktualną cenę komputera według wzoru:

$$cenak = cenak + \frac{5}{100} \cdot cenap$$

i wyprowadzamy wynik: aktualną cenę komputera (*cenak*). W przeciwnym wypadku (czyli dla $w > 20$) wyprowadzamy komunikat „za wysoka cena".

Po opisaniu algorytmu należy sprawdzić, czy wszystkie sformułowania i wartości podane w specyfikacji zadania i opisie algorytmu są wystarczająco precyzyjnie określone, by można było zrealizować rozwiązanie na komputerze.

Do rozwiązania zadania wykorzystamy arkusz kalkulacyjny, ponieważ możemy w nim realizować algorytmy z warunkami za pomocą funkcji JEŻELI. Można również stosować zagnieżdżoną funkcję JEŻELI, w której argumentem funkcji JEŻELI jest wartość będąca wynikiem kolejnej funkcji JEŻELI.

Uwaga: Zadanie można również rozwiązać w wybranym języku programowania – zrealizujemy je w językach C++ i Python w następnych tematach.

Ćwiczenie 3. Rozwiązywanie zadania za pomocą arkusza kalkulacyjnego

1. Korzystając z opisu przedstawionego w przykładach 3-5, zaprojektuj w arkuszu kalkulacyjnym tabelę do wprowadzania danych umożliwiającą rozwiązanie problemu sprzedawcy komputerów. Utwórz formułę obliczającą i wyprowadzającą odpowiedni wynik.
2. Zapisz plik pod nazwą *Kalkulacja*.

Każde rozwiązanie należy przetestować dla różnych danych. Danymi do zadania rozwiązywanego w ćwiczeniu 3. są liczby, określające odpowiednio: cenę komputera, cenę płyty głównej i procent, o jaki zostanie zmieniona cena płyty głównej. Określiliśmy też związek między danymi a wynikami (przykład 3.) oraz przyjęliśmy odpowiedni algorytm ustalania ceny komputera (przykład 4.).

Testując rozwiązanie, będziemy oceniać **poprawność** zastosowanego algorytmu.

Algorytm jest **poprawny**, jeśli rozwiązuje problem zgodnie ze specyfikacją (czyli dla poprawnych danych daje poprawne wyniki) oraz dla poprawnych danych zawsze kończy swoje działanie (nie zapętla się).

Ćwiczenie 4. Testowanie rozwiązania

Przetestuj rozwiązanie ćwiczenia 3. dla różnych danych. Pokaż na przykładach, że dla poprawnych danych (różnych wartości *w*, spełniających podane warunki) otrzymujesz poprawne wyniki.

Na zakończenie pracy nad omawianym problemem możemy przeprowadzić prezentację rozwiązania zadania i omówić zastosowane narzędzie. Odpowiadamy m.in. na pytanie, czy zastosowaliśmy właściwe narzędzie. Wybór arkusza kalkulacyjnego jest właściwy, ponieważ arkusz posiada mechanizmy obliczeń zawierających sytuacje warunkowe (funkcję JEŻELI).

Ćwiczenie 5. Zaprezentowanie rozwiązania

Przedstaw i omów rozwiązanie zastosowane w ćwiczeniu 3., korzystając z projektora multimedialnego.

Warto zapamiętać

- Komputer wykonuje programy komputerowe, a program realizuje algorytmy (rys. 1.).
- Aby rozwiązać dowolny problem (zadanie), trzeba go poprawnie sformułować, określić dane i wyniki oraz warunki, jakie muszą zostać spełnione. Następnie należy ułożyć lub wybrać odpowiedni algorytm i określić sposób jego realizacji oraz wybrać narzędzie, które to umożliwi.

Pytania i polecenia

1. Omów zależności przedstawione na rysunku 1.
2. Wyjaśnij pojęcia algorytmu i specyfikacji problemu (zadania).
3. Jakie są cechy algorytmu informatycznego?

Zadania

1. Napisz specyfikacje do zadań:
   a. Oblicz średnią arytmetyczną trzech liczb.
   b. Zebrano dane o wzroście uczniów klas pierwszych w twojej szkole. Uporządkuj informacje o wzroście malejąco.
   c. Znajdź wśród danych o wzroście uczniów w twojej klasie najmniejszą i największą liczbę.
2. Wybierz jedno z zadań od 1a do 1c i wykonaj je w arkuszu kalkulacyjnym. Przetestuj rozwiązanie dla różnych danych. Czy arkusz kalkulacyjny jest odpowiednim narzędziem do wykonania każdego zadania? Uzasadnij odpowiedź.

Dla zainteresowanych

3. Opisz (w edytorze tekstu lub zeszycie przedmiotowym) przykładową sytuację problemową, w której występują sytuacje warunkowe (możesz wzorować się na opisie problemu sprzedawcy komputerów). Sformułuj zadanie i zapisz jego specyfikację. Zaprojektuj rozwiązanie zadania, wzorując się na przykładzie 5.
4. Wyjaśnij pochodzenie słowa *algorytm*. Informacje znajdź w Internecie.

Temat 53. Techniki algorytmiczne i programowanie

1. Języki programowania
2. Na czym polega programowanie?
3. Sytuacje warunkowe
4. Iteracja
5. Stosowanie iteracji do sumowania liczb

Warto powtórzyć

1. Jakie znasz rodzaje oprogramowania komputerowego?
2. Z jakich programów użytkowych korzystaliśmy? Jakie było ich przeznaczenie?
3. Jakie są zastosowania języków programowania?
4. Z jakich środowisk programowania korzystaliśmy w szkole podstawowej?
5. Na czym polega programowanie w środowiskach poznanych w szkole podstawowej?

1. Języki programowania

Żeby przedstawić algorytm w postaci programu komputerowego, należy zapisać go jako ciąg instrukcji języka programowania. Każda instrukcja odpowiada określonej operacji lub ciągowi operacji.

Każdy język programowania posiada swój zbiór instrukcji, w tym **słowa kluczowe**. Podobnie jak języki naturalne, którymi posługują się ludzie, języki programowania posiadają odpowiednie **zasady składni** oraz właściwe **słownictwo**.

Każdy język naturalny dysponuje własnymi słowami, służącymi określaniu czynności i nazywaniu rzeczy. Podobnie jest z językami programowania.

Instrukcje każdego języka programowania realizują czynności takie, jak: wprowadzanie danych, wyprowadzanie wyników, wykonywanie obliczeń, sprawdzanie warunków czy powtarzanie operacji.

S **Słowo kluczowe**

Słowo mające szczególne znaczenie w danym języku programowania, np. oznaczające określony rozkaz, instrukcję lub polecenie.

Niezależnie od stosowanego języka programowania należy pamiętać, że:

1. Język programowania jest językiem formalnym, co oznacza, że podlega jednoznacznym regułom. Języki naturalne nie posiadają tej cechy, ponieważ czasem sens zdania zależy od kontekstu, w którym zostało ono użyte, lub od naszej interpretacji.
2. Postać instrukcji, w tym słów kluczowych, musi być bardzo precyzyjna – zgodna z zasadami składni. W programie nie może zabraknąć ani jednego koniecznego znaku (np. dwukropka, przecinka, średnika, nawiasu). Tłumaczenie programu na kod maszynowy połączone jest ze sprawdzaniem poprawności składniowej zapisanych instrukcji. Instrukcje błędnie zapisane nie zostaną przetłumaczone, a tym samym komputer nie będzie mógł wykonać programu.
3. Kolejność zapisywania instrukcji powinna odpowiadać kolejności operacji realizujących dany algorytm.



Języki programowania można podzielić na:

- **języki wysokiego poziomu**, np. Java, C, C++, C# (C Sharp), Python, Visual Basic, Pascal, PHP, JavaScript;
- **języki niskiego poziomu** (wewnętrzne), np. assemblery.

2. Na czym polega programowanie?



Zapisanie algorytmu w postaci ciągu instrukcji języka programowania wysokiego poziomu nazywamy **implementacją**. Powstaje wówczas tzw. **program (kod) źródłowy**.

Implementacja I

Zapisanie algorytmu w postaci kodu źródłowego.

Kod maszynowy (program maszynowy, kod wynikowy, program wynikowy) K

Program napisany w języku wewnętrznym (maszynowym), wykonywany przez procesor.

Kod źródłowy można zapisać w dowolnym edytorze tekstu, najwygodniej jest jednak skorzystać ze specjalnych edytorów dla programistów lub z edytora wbudowanego w środowisko programistyczne danego języka.

Program źródłowy (zapisany jako dokument tekstowy) nie może być wykonywany przez procesor, który potrafi wykonywać jedynie **kod maszynowy**. Pisząc program w języku programowania, przygotowujemy jedynie źródło do utworzenia kodu maszynowego, czyli programu rozumianego przez komputer. Komputer, a dokładniej – procesor, wykonuje program w kodzie maszynowym (w języku wewnętrznym procesora).

Program komputerowy może występować w dwóch postaciach:

- jako **program** (**kod**) **źródłowy** – program zrozumiały dla programisty, zapisany jako ciąg instrukcji tekstowych języka programowania wysokiego poziomu, np. C++ (rys. 3.), Python (rys. 4.),
- jako **program** (**kod**) **wynikowy** (**maszynowy**) – program zrozumiały dla komputera, napisany w języku wewnętrznym komputera w postaci ciągu instrukcji dla procesora (rys. 5.).

Program napisany w języku wysokiego poziomu (np. Python, C++) musi zostać przetłumaczony na język niskiego poziomu (język wewnętrzny komputera). Proces ten nazywamy **translacją**. Może ona przebiegać w formie **kompilacji** lub **interpretacji**, wykonywanych w **translatorze**, czyli programie do tłumaczenia programu na kod maszynowy.

Przykładami języków kompilowanych są: C, C++, C# (C Sharp), Pascal.

Przykładami języków interpretowanych są Scratch oraz Python. W zadaniach rozwiązywanych w tym podręczniku będziemy używać języków C++ i Python.

K **Kompilacja**

Przetłumaczenie programu w całości na język zrozumiały dla procesora, tak by mógł go wykonać komputer. Jeśli kompilacja przebiegnie poprawnie, można uruchomić program. Raz skompilowany program nie wymaga już powtórnej operacji tłumaczenia.

I **Interpretacja**

Tłumaczenie programu utworzonego w języku programowania instrukcja po instrukcji, tak aby komputer mógł wykonać każdą z nich. Tłumaczenie następuje każdorazowo przy uruchomieniu programu.

Przykład 1. Przedstawienie algorytmu obliczania sumy dwóch liczb w języku programowania

Zadanie: Napisz w wybranym języku programowania program umożliwiający wprowadzenie z klawiatury dwóch liczb całkowitych, obliczający ich sumę i wyprowadzający wynik obliczeń na ekran. Przed wprowadzeniem każdej danej wyświetlaj napis „Podaj liczbę”, a przed wyprowadzeniem wyniku – „Suma wynosi:”.

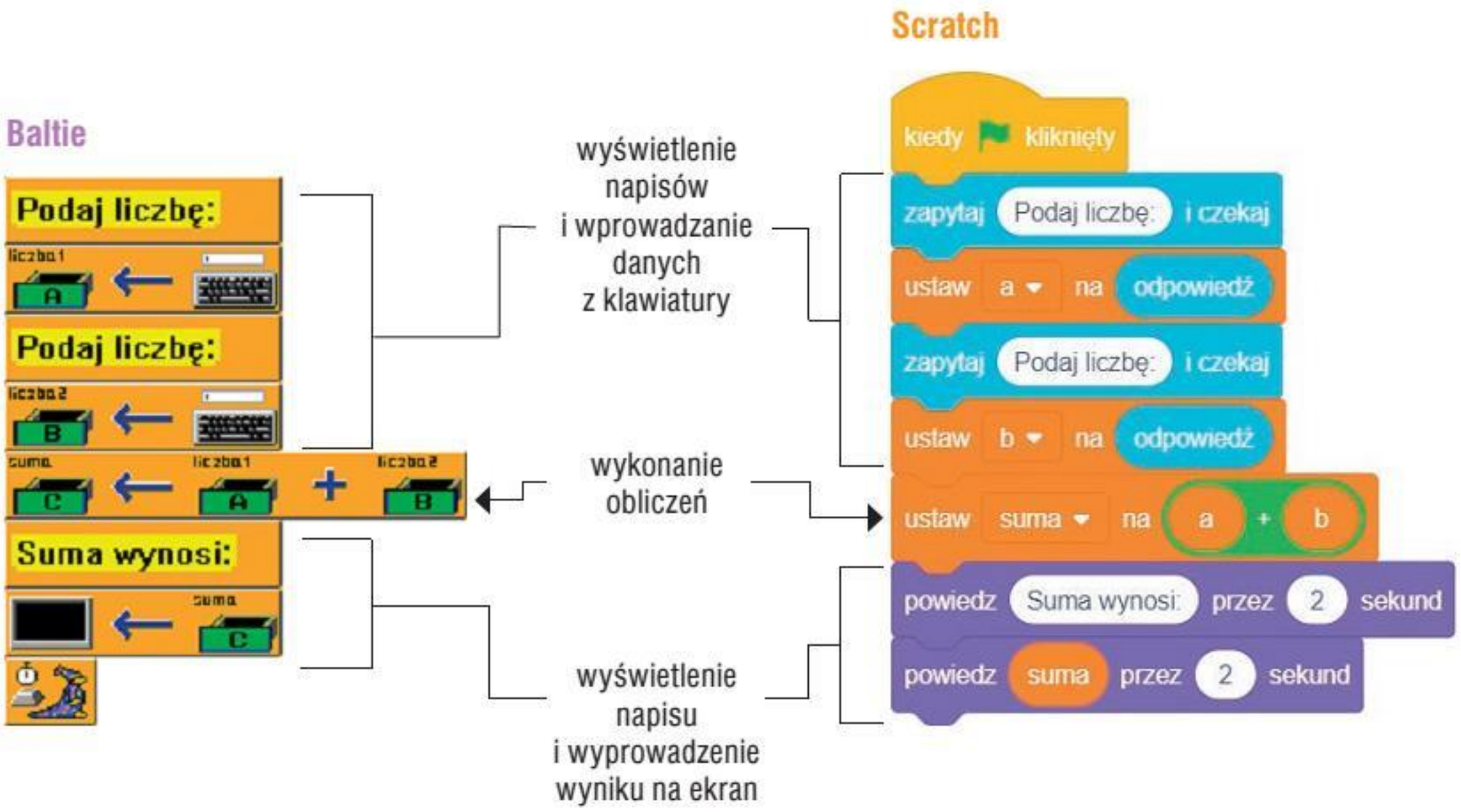
Dane: dwie liczby całkowite: *a*, *b*.

Wynik: wartość sumy liczb *a* i *b*: *suma*.

Na rysunkach 1. i 2. pokazano realizację zadania w środowiskach dydaktycznych Baltie i Scratch, a na rysunkach 3. i 4. – w językach C++ i Python. Na pierwszy rzut oka wydaje się, że są to cztery różne programy. Każde środowisko dydaktyczne ma inaczej wyglądające elementy graficzne, inny zapis poleceń, a każdy z języków programowania może mieć inną postać instrukcji, odmienne zasady składni, w tym stosowanie znaków interpunkcyjnych itp.

Zauważmy jednak, że w każdym rozwiązaniu występują te same części programu, ułożone w takiej samej kolejności:

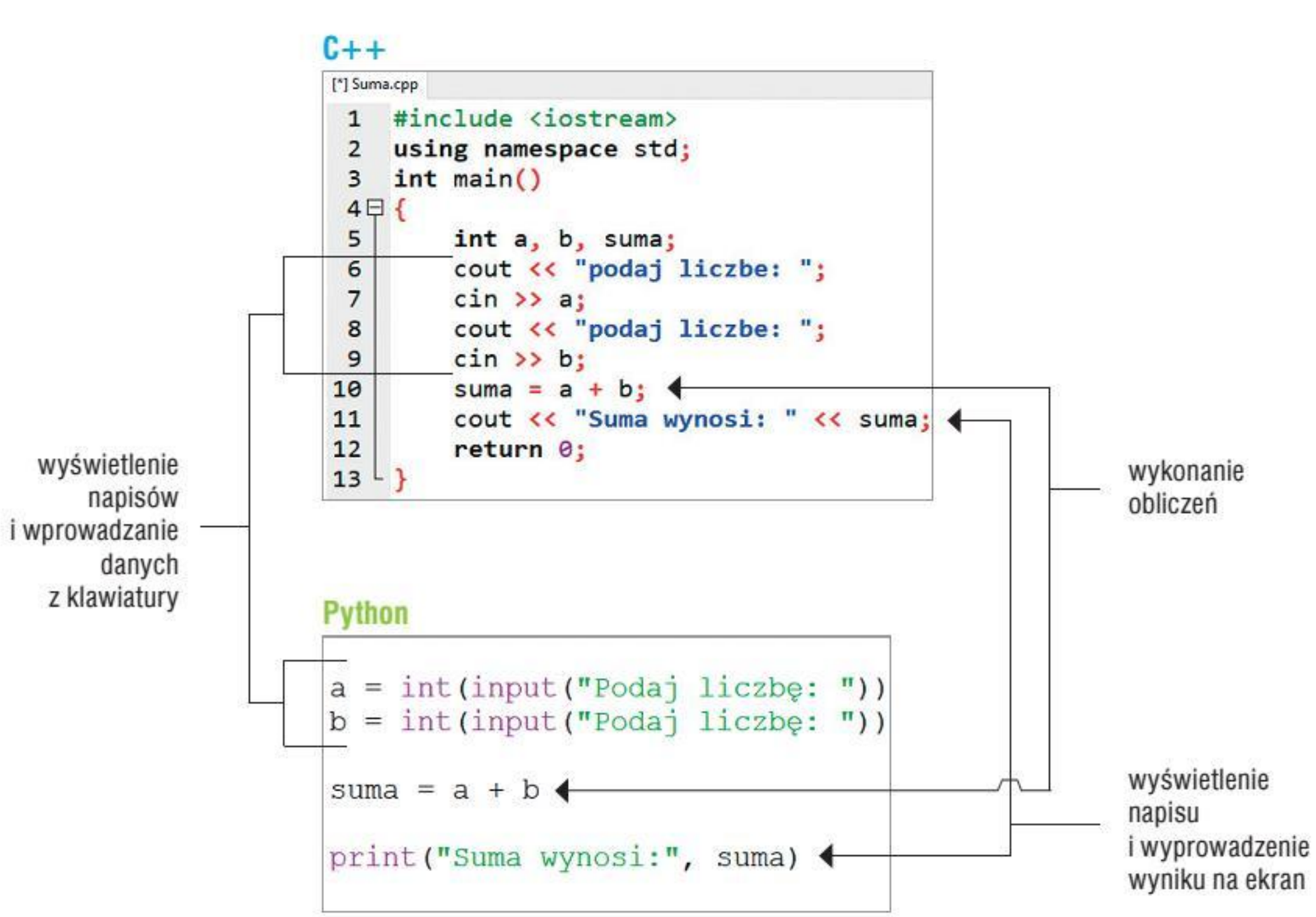
1. wyświetlenie napisu i wprowadzanie danych z klawiatury,
2. wykonanie obliczeń,
3. wyświetlenie napisu i wyprowadzenie wyniku na ekran.



Rys. 1. Program w środowisku Baltie

Rys. 2. Program w środowisku Scratch

Rys. 3. Program w języku C++



Rys. 4. Program w języku Python

```
0x00401374      call    0x416610 <__main>
0x00401379      lea     -0x20(%ebp),%eax
0x0040137C      mov     %eax,(%esp)
0x0040137F      movl    $0x1,-0x58(%ebp)
0x00401386      mov     $0x477900,%ecx
0x0040138B      call    0x446984 <std::istream::operator>>(int&)>
0x00401390      sub     $0x4,%esp
0x00401393      lea     -0x24(%ebp),%eax
0x00401396      mov     %eax,(%esp)
0x00401399      mov     $0x477900,%ecx
0x0040139E      call    0x446984 <std::istream::operator>>(int&)>
```

Rys. 5. Program (kod) maszynowy, czyli wynik przetłumaczenia programu napisanego w języku C++ pokazany w symbolicznej formie

Ćwiczenie 1. Porównujemy rozwiązania tego samego zadania w różnych narzędziach

1. Porównaj programy pokazane na rysunkach 1. i 2. – podaj różnice i podobieństwa. Wskaż odpowiadające sobie polecenia i elementy (przedmioty). Na przykład: *Jakiemu elementowi w języku Scratch odpowiada przedmiot z klawiaturą w środowisku Baltie?*
2. Porównaj programy pokazane na rysunkach 3. i 4. – podaj różnice i podobieństwa. Na przykład: *Czym różni się zapis instrukcji obliczającej sumę liczb* a *i* b *w obydwu językach?* Wskaż odpowiadające sobie instrukcje.
3. W wybranym środowisku programowania (Baltie lub Scratch) utwórz program realizujący zadanie opisane w przykładzie 1.
4. Zapisz program w pliku pod nazwą *Suma*.

Program komputerowy może zostać napisany na podstawie **listy kroków**, **schematu blokowego** lub samej specyfikacji problemu.

L **Lista kroków**

Przedstawienie algorytmu w kolejnych punktach (krokach). Każdy punkt takiej listy zawiera opis wykonywanej czynności. Kolejność punktów nie jest przypadkowa – musi być zgodna z działaniem algorytmu.

S **Schemat blokowy**

Przedstawienie poszczególnych operacji algorytmu za pomocą odpowiednio połączonych figur (bloków). Połączenia określają, w jakiej kolejności i w jaki sposób będą wykonywane operacje realizujące dany algorytm.

W kolejnych tematach algorytmy będziemy opisywać i przedstawiać głównie w postaci programów komputerowych (w językach C++ i Python), a na lekcjach realizowanych w zakresie rozszerzonym – w postaci list kroków, schematów blokowych i programów komputerowych.

3. Sytuacje warunkowe

Sytuacja warunkowa występuje wtedy, gdy wynik lub dalsze działanie zależą od spełnienia (lub niespełnienia) pewnego warunku. Algorytm zawierający sytuacje warunkowe nazywamy **algorytmem z warunkami** (inaczej: **algorytmem z rozgałęzieniami**).

Sytuacje warunkowe występują w matematyce i fizyce, gdy wykonanie obliczeń zależy od warunku, który muszą spełniać dane, np. powinny być różne od zera lub dodatnie.

Przykład 2. Specyfikacja zadania i lista kroków algorytmu z warunkami

Zadanie: Sprawdź, czy dana liczba rzeczywista jest dodatnia czy niedodatnia.

Dane: liczba rzeczywista: *x*.

Wynik: komunikat „liczba dodatnia" lub „liczba niedodatnia" – zależnie od wartości wprowadzonych danych.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź liczbę *x*.
3. Jeśli $x > 0$,
   wyprowadź komunikat „liczba dodatnia";
   w przeciwnym wypadku
   wyprowadź komunikat „liczba niedodatnia".
4. Zakończ algorytm.

Ćwiczenie 2. Podajemy przykład sytuacji warunkowej

Podaj przykład zadania, w którym występuje sytuacja warunkowa. Napisz jego specyfikację i listę kroków.

4. Iteracja

Iteracja, czyli technika powtarzania tych samych operacji, pojawia się w wielu zadaniach.

Często trzeba wykonać takie same działania (ciągi operacji) na wielu różnych danych, co jest pracochłonne i wymaga sporo czasu.

Jednym z pierwszych zadań komputera było zastępowanie człowieka w wykonywaniu powtarzających się obliczeń na wielu liczbach. W tym celu powstały też pierwsze maszyny liczące. Rozwiązywanie problemów iteracyjnych stało się ważnym zastosowaniem komputera, zwłaszcza że tworzenie programów komputerowych ma sens przede wszystkim w przypadku czynności powtarzalnych.

Prawie wszystkie wcześniej omawiane problemy można rozwiązać, korzystając z takich narzędzi, jak arkusz kalkulacyjny czy kalkulator. Jednak, w celu wielokrotnego wykonania (tysiące, a nawet miliony razy) tych samych operacji, o wiele szybciej i efektywniej jest skorzystać z programu komputerowego utworzonego specjalnie w tym celu.

Iteracja polega na powtarzaniu tej samej operacji (ciągu operacji).

Iterację **implementujemy**, stosując tzw. **pętlę**. Z pętlą mamy do czynienia, gdy w pewnym kroku algorytmu wracamy do jednego z wcześniejszych kroków, co powoduje, że kroki te zostaną wykonane wiele razy.

5. Stosowanie iteracji do sumowania liczb

Zastanówmy się, jak dodajemy w pamięci lub na kalkulatorze, np. siedem liczb. Zadanie wydaje się banalne i zazwyczaj odpowiadamy: „po prostu dodajemy do siebie wszystkie liczby i podajemy wynik". Ale co to znaczy „dodać do siebie wszystkie liczby"?

Wykonując to zadanie w pamięci lub na kalkulatorze, realizujemy pewien algorytm: dodajemy dwie pierwsze liczby do siebie, zapamiętujemy ich sumę, następnie do tej sumy dodajemy trzecią liczbę, znów zapamiętujemy wynik itd. Opis tego algorytmu może wyglądać jak w przykładzie 3.

Przykład 3. Obliczenie sumy siedmiu liczb

1. Zacznij algorytm.
2. Suma jest równa zero.
3. Wprowadź pierwszą liczbę.
4. Do poprzedniego wyniku sumowania dodaj wprowadzoną liczbę.
5. Wprowadź drugą liczbę.
6. Do poprzedniego wyniku sumowania dodaj wprowadzoną liczbę.
7. Wprowadź trzecią liczbę.
8. Do poprzedniego wyniku sumowania dodaj wprowadzoną liczbę.
9. Wprowadź czwartą liczbę.
10. Do poprzedniego wyniku sumowania dodaj wprowadzoną liczbę.
11. Wprowadź piątą liczbę.
12. Do poprzedniego wyniku sumowania dodaj wprowadzoną liczbę.
13. Wprowadź szóstą liczbę.
14. Do poprzedniego wyniku sumowania dodaj wprowadzoną liczbę.
15. Wprowadź siódmą liczbę.
16. Do poprzedniego wyniku sumowania dodaj wprowadzoną liczbę.
17. Wyprowadź wynik sumowania.
18. Zakończ algorytm.

Zauważmy, że operacje zapisane w krokach 3. i 4. z przykładu 3. powtarzają się w kolejnych parach kroków (5-6, 7-8, 9-10, 11-12, 13-14 i 15-16). Gdybyśmy mieli zsumować np. kilka tysięcy liczb, opis algorytmu byłby bardzo długi. W przedstawionej sytuacji możemy powtarzające się ciągi operacji (tu kroki 3. i 4.) zapisać tylko raz i zastosować pętlę, czyli po wykonaniu czwartego kroku wrócić do trzeciego, to znaczy wykonać kroki 3. i 4. siedem razy.

W przykładzie 3. dodaliśmy siedem liczb. Zazwyczaj jednak zapisujemy algorytm w uogólnionej postaci – dla dowolnej liczby danych. Taki algorytm jest uniwersalny.

Przykład 4. Specyfikacja zadania i lista kroków algorytmu sumowania *n* liczb

Zadanie: Oblicz sumę *n* liczb całkowitych.

Dane: liczba naturalna *n* większa od zera, oznaczająca, ile liczb ma być sumowanych, *n* dowolnych liczb całkowitych wprowadzanych kolejno i zapamiętywanych w zmiennej *a*.

Wynik: wartość sumy: *suma*.

Lista kroków:

1. Zacznij algorytm.
2. Zmiennej *suma* przypisz wartość 0.
3. Zmiennej *i* przypisz wartość 0.
4. Podaj, ile liczb ma być zsumowanych. Zapamiętaj wprowadzoną liczbę w zmiennej *n*.
5. Wprowadź liczbę do sumowania – zapamiętaj ją w zmiennej *a*.
6. Zmiennej *suma* przypisz wartość zmiennej *suma* powiększoną o wartość zmiennej *a*: $suma = suma + a$.
7. Zmiennej *i* przypisz wartość zmiennej *i* powiększoną o 1: $i = i + 1$.
8. Jeśli $i < n$, wróć do kroku 5.
9. Wyprowadź wynik: *suma*.
10. Zakończ algorytm.

W przykładzie 4. wartość sumy liczb dla kolejno wprowadzanych danych jest zapamiętywana w zmiennej *suma*, której na początku algorytmu przypisujemy wartość zero (krok 2.).

Każdą z kolejno wprowadzanych liczb zapamiętujemy w zmiennej *a* (w przypadku algorytmu iteracyjnego zwykle nie stosujemy różnych nazw zmiennych dla kolejnych liczb).

Przypisanie *suma = suma + a* oznacza „pod zmienną *suma* podstaw poprzednią wartość zmiennej *suma* zwiększoną o wartość kolejnej liczby *a*". Nie jest to równość w sensie matematycznym. Zapis ten odpowiada **instrukcji przypisania** stosowanej w językach programowania. Jeśli w tym samym programie zapiszemy w tej samej zmiennej nową wartość (wykonamy instrukcję przypisania), to poprzednia wartość zostanie usunięta. Na przykład po wykonaniu kolejno następujących instrukcji przypisania:

C++

```
suma = 7;
suma = suma + 20;
```

Python

```
suma = 7
suma = suma + 20
```

wartość zmiennej *suma* wyniesie 27.

Przykład 5. Analiza działania algorytmu dodawania *n* liczb

Prześledzimy działanie algorytmu dla $n = 7$ i dla kolejnych wartości zmiennej *a*: 7, 20, 13, 15, 8, 11, 33.

Zmienna *i* oznacza w tym przykładzie numer wprowadzonej liczby. W analizie przedstawiamy wartości zmiennych po wykonaniu każdego cyklu iteracji.

Na początku przyjmujemy: $suma = 0$, $i = 0$, $n = 7$.

cykl 1.	$a = 7$	$suma = 0 + 7 = 7$	$i = 0 + 1 = 1$	czy $i < n$, $i < 7$?	TAK
cykl 2.	$a = 20$	$suma = 7 + 20 = 27$	$i = 1 + 1 = 2$	czy $i < n$, $i < 7$?	TAK
cykl 3.	$a = 13$	$suma = 27 + 13 = 40$	$i = 2 + 1 = 3$	czy $i < n$, $i < 7$?	TAK
cykl 4.	$a = 15$	$suma = 40 + 15 = 55$	$i = 3 + 1 = 4$	czy $i < n$, $i < 7$?	TAK
cykl 5.	$a = 8$	$suma = 55 + 8 = 63$	$i = 4 + 1 = 5$	czy $i < n$, $i < 7$?	TAK
cykl 6.	$a = 11$	$suma = 63 + 11 = 74$	$i = 5 + 1 = 6$	czy $i < n$, $i < 7$?	TAK
cykl 7.	$a = 33$	$suma = 74 + 33 = 107$	$i = 6 + 1 = 7$	czy $i < n$, $i < 7$?	NIE

Wyprowadzenie wartości zmiennej *suma*: 107.

Ćwiczenie 3. Analizujemy działanie algorytmu dodawania liczb

1. Korzystając z przykładów 4. i 5., prześledź działanie algorytmu dodawania *n* liczb dla $n = 5$ i dla kolejnych wartości zmiennej *a*: 23, -35, 40, -7, 25.
2. Wykonaj to zadanie, korzystając z programu *Kalkulator*. Zwróć uwagę, że zadanie zawsze wykonywane jest według tego samego algorytmu.

Warto zapamiętać

- Programowanie polega na utworzeniu kodu źródłowego, czyli zapisaniu (implementacji) algorytmu w postaci programu w wybranym języku programowania.
- Aby program był zrozumiały dla komputera, musi zostać przetłumaczony z języka wysokiego poziomu na język wewnętrzny komputera. Powstaje wówczas program (kod) maszynowy (wynikowy).
- Proces tłumaczenia kodu źródłowego na kod maszynowy nazywamy translacją. Może ona przebiegać w formie kompilacji lub interpretacji.
- Wyróżniamy następujące etapy programowania: implementację, kompilację (lub interpretację), uruchomienie i wykonanie oraz testowanie.
- Każdy język programowania posiada odpowiedni zestaw instrukcji, w tym słów kluczowych, umożliwiający zapisanie algorytmów opracowanych przez programistów.
- W algorytmie z warunkami występują sytuacje warunkowe – wynik lub dalsze działanie algorytmu zależą od spełnienia określonego warunku.
- Technika iteracji ma zastosowanie do czynności powtarzalnych. Iteracja oznacza powtarzanie tej samej operacji lub ciągu operacji.
- Iterację implementujemy, stosując tzw. pętlę. Z pętlą mamy do czynienia, gdy w pewnym kroku algorytmu wracamy do jednego z wcześniejszych kroków, co powoduje, że określone kroki algorytmu zostaną wykonane wiele razy.

Pytania i polecenia

1. Wymień te cechy języka programowania, które odróżniają go od języka naturalnego.
2. Czym jest implementacja?
3. Czym różni się kod źródłowy od kodu maszynowego?
4. Przedstaw i omów na konkretnym przykładzie etapy programowania.
5. Na czym polega kompilacja programu? Czym różni się od interpretacji?
6. Wyjaśnij, na czym polega sytuacja warunkowa. Podaj przykład sytuacji warunkowej.
7. Na czym polega iteracja?
8. Kiedy mamy do czynienia z pętlą?
9. Omów zastosowanie iteracji na przykładzie sumowania ośmiu liczb.

Zadania

1. W wybranym środowisku dydaktycznym (Baltie lub Scratch) utwórz program obliczający średnią z trzech liczb wprowadzanych z klawiatury. Zapisz plik pod nazwą *Srednia*.
2. Określ, czy w zadaniu „Oblicz iloraz dwóch liczb *a* i *b* dla *b* różnego od zera" występuje sytuacja warunkowa. Uzasadnij odpowiedź. Napisz specyfikację do tego zadania.
3. Podaj przykład zadania, w którym występuje iteracja. Napisz specyfikację tego zadania.

Dla zainteresowanych

4. Przypomnij ze szkoły podstawowej, w jaki sposób przedstawia się algorytm z warunkami w postaci schematu blokowego. Narysuj schemat blokowy przykładowego algorytmu z warunkami.
5. Przypomnij ze szkoły podstawowej, w jaki sposób przedstawia się algorytm iteracyjny w postaci schematu blokowego. Narysuj schemat blokowy przykładowego algorytmu iteracyjnego.

Przeczytaj, jeśli chcesz wiedzieć więcej…

Pierwszy język wysokiego poziomu powstał w 1954 roku. Nazywał się FORTRAN (nazwa pisana wielkimi literami). Kompilator FORTRAN-u był również pierwszym kompilatorem języka wysokiego poziomu. Język ten doczekał się wielu wersji.

Do wersji 77. nazywał się FORTRAN (nazwa pisana wielkimi literami), a od wersji 90. do dziś – **Fortran** (nazwa jedynie zaczyna się wielką literą). Co ciekawe, język ten nadal jest rozwijany i używany. Uznaje się go za najpopularniejszy język do obliczeń numerycznych, naukowo-inżynierskich, symulacji. Kolejne wersje umożliwiają programowanie strukturalne, obiektowe, modularne i równoległe.

Rozdział XII

Programowanie w języku C++

Temat 54. Środowisko programistyczne języka C++, stosowanie zmiennych i wyprowadzanie wyników

Temat 55. Algorytmy z warunkami w języku C++

Temat 56. Sprawdzanie poprawności danych i zapisywanie rozwiązania problemu w języku C++

Temat 57. Instrukcje warunkowe zagnieżdżone i instrukcja wyboru w języku C++

Temat 58. Algorytmy iteracyjne w języku C++

Temat 59. Instrukcje iteracyjne zagnieżdżone w języku C++

Temat 60. Stosowanie instrukcji iteracyjnych `for`, `while` i `do ... while` w języku C++

Temat 61. Funkcje zwracające wartość w języku C++

Temat 62. Funkcje niezwracające wartości w języku C++

Temat 63. Deklarowanie i stosowanie tablicy w języku C++

Temat 64. Wykonywanie operacji na elementach tablicy w języku C++

Temat 65. Dane tekstowe w języku C++

Temat 54.

Środowisko programistyczne języka C++, stosowanie zmiennych i wyprowadzanie wyników

1. **Środowisko programistyczne języka C++**
   1.1. Struktura programu w języku C++
   1.2. Etapy tworzenia programu w języku C++
2. **Stosowanie zmiennych**
   2.1. Deklarowanie zmiennych
   2.2. Nadawanie zmiennym wartości
   2.3. Wykonywanie obliczeń
3. **Wyprowadzanie komunikatów i wyników na ekran monitora**

Warto powtórzyć

1. Jakie informacje należy umieścić w specyfikacji zadania?
2. Czym jest program komputerowy?
3. Czym jest język programowania?
4. Wyjaśnij pojęcie *słowo kluczowe*.
5. Dlaczego o języku programowania mówimy, że jest językiem formalnym?
6. Czym jest kod źródłowy?
7. Z czego powinna wynikać kolejność instrukcji programu?
8. Na czym polega kompilacja programu?

1. Środowisko programistyczne języka C++

Aby pisać programy w języku C++, należy zainstalować wybraną aplikację zawierającą edytor i kompilator języka C++, np. Code::Blocks, Dev-C++. Są one dostępne bezpłatnie w Internecie. Można także skorzystać z rozwiązań komercyjnych, np. Visual Studio, C++Builder.

Wyspecjalizowany pakiet programistyczny zawiera edytor kodu źródłowego, kompilator i inne narzędzia wspomagające programowanie. Taką aplikację (lub zespół aplikacji) określa się też jako **zintegrowane środowisko programistyczne**.

W tym podręczniku będziemy korzystać ze środowiska Dev-C++.

Z **Zintegrowane środowisko programistyczne**
(IDE – z ang. *Integrated Development Environment*)

Aplikacja lub zespół aplikacji do tworzenia, modyfikowania, testowania i utrzymywania oprogramowania.

1.1. Struktura programu w języku C++

Prosty program w języku C++ składa się z **poleceń preprocesora** (tzw. **dyrektyw**) oraz **funkcji głównej** `main()`. W programie musi wystąpić funkcja `main()`, od której rozpoczyna się wykonanie programu. Funkcja `main()` zawiera główny program. Przed nazwą funkcji powinno się znajdować słowo kluczowe **`int`**, informujące, że funkcja `main()` zwraca wartość liczbową. Instrukcja **`return`** `0` na końcu funkcji powoduje zwrócenie do systemu operacyjnego wartości 0, informującej o prawidłowym wykonaniu programu. Pozostałe elementy programu są opcjonalne.

Przykładowe elementy programu, ułożone w zwykle stosowanej kolejności, pokazujemy na rysunku 1.

P **Preprocesor**

Program, który przetwarza wstępnie kod źródłowy przed rozpoczęciem zasadniczej kompilacji. Polecenia preprocesora (zwane dyrektywami) rozpoczynają się od znaku **#**. Np. dyrektywa **`#include`** nakazuje dołączenie do kodu źródłowego kodu z innego pliku źródłowego.

dołączenie biblioteki standardowej

informacja o korzystaniu z biblioteki standardowej

```
#include <iostream>
using namespace std;
// deklaracja zmiennych globalnych (jeśli występują)
// deklaracja funkcji (jeśli występują)

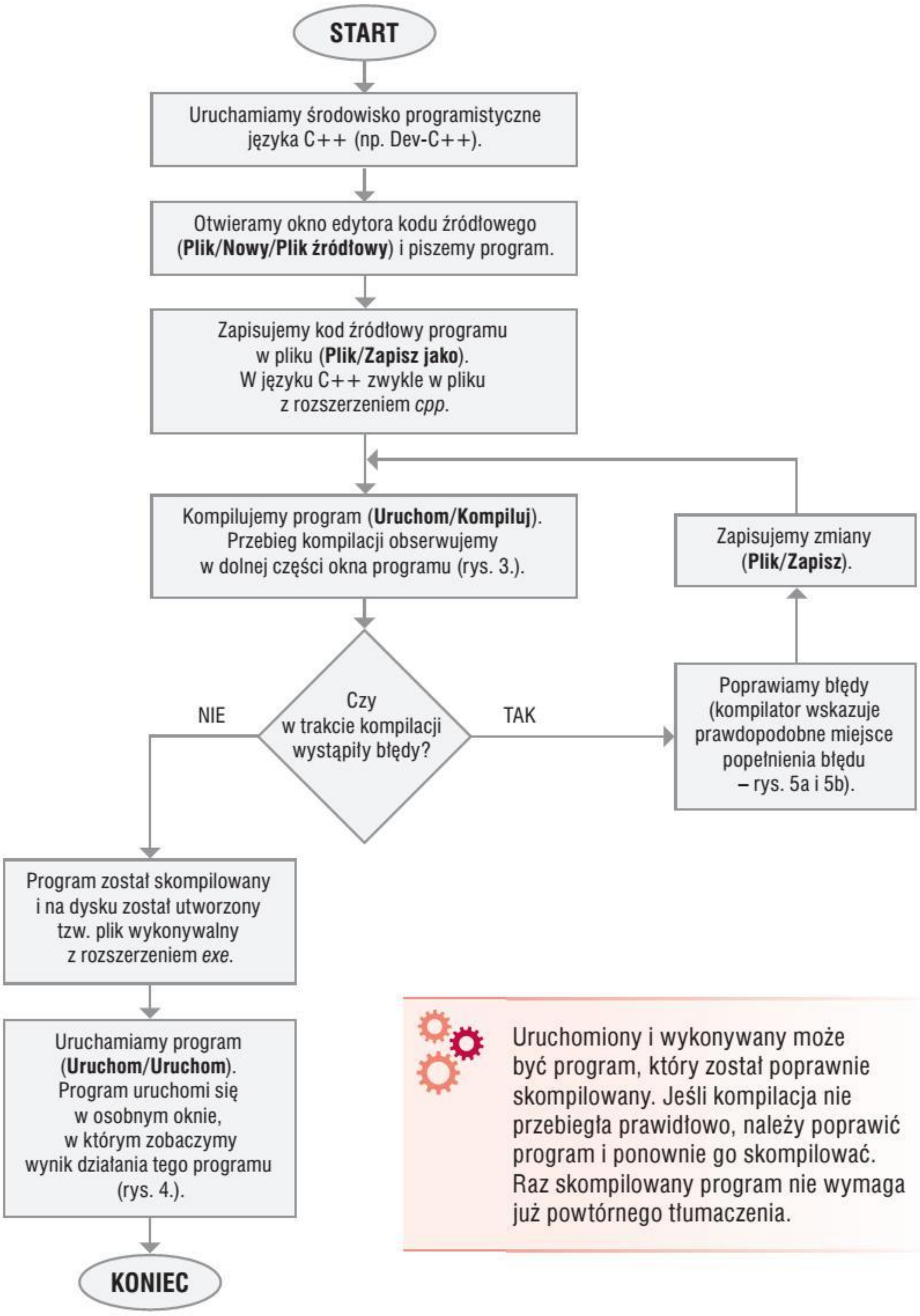
int main()
{
      // deklaracja zmiennych lokalnych (jeśli występują)
      // część wykonawcza programu
   return 0;
}
```

Rys. 1. Ogólna struktura programu w języku C++

1.2. Etapy tworzenia programu w języku C++

Po uruchomieniu środowiska programistycznego języka C++ zobaczymy okno z głównym menu programu. Okno edytora wbudowanego do środowiska programistycznego otwieramy w menu **Plik** – możemy otworzyć istniejący plik (plik źródłowy) lub utworzyć nowy.

W każdym języku kompilowanym możemy wyróżnić etapy tworzenia programu pokazane na rysunku 2.

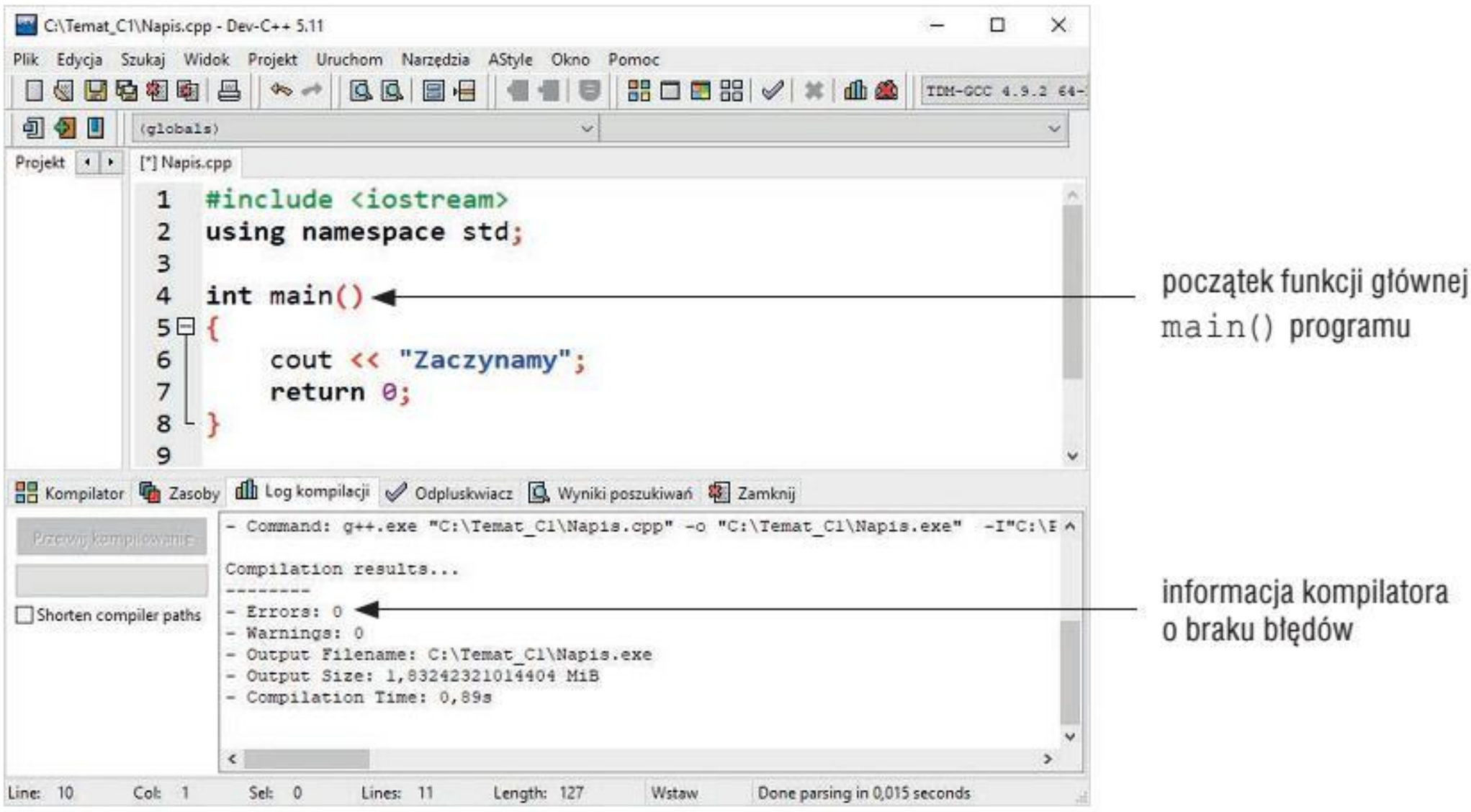


Uruchomiony i wykonywany może być program, który został poprawnie skompilowany. Jeśli kompilacja nie przebiegła prawidłowo, należy poprawić program i ponownie go skompilować. Raz skompilowany program nie wymaga już powtórnego tłumaczenia.

Rys. 2. Etapy tworzenia programu w kompilowanym języku programowania

Kilka zasad pisania programu w języku C++

- W programowaniu w języku C++, jak w każdym języku, bardzo istotna jest prawidłowa kolejność zapisu poleceń.
- Instrukcje języka C++, podobnie jak innych języków programowania, realizują takie czynności, jak: wprowadzanie danych, wyprowadzanie wyników, wykonywanie obliczeń, sprawdzanie warunków czy powtarzanie operacji.
- Każda instrukcja programu powinna być zakończona średnikiem.
- W języku C++ **blok programu** wyznaczają nawiasy klamrowe {}. **Komentarze** (ignorowane przez kompilator) umieszczamy po znakach // w tej samej linii lub wewnątrz znaków /* (początek bloku komentarza) i */ (koniec bloku komentarza).



Rys. 3. Okno środowiska programistycznego języka C++ z programem źródłowym – kompilacja przebiegła prawidłowo

Ćwiczenie 1. Poznajemy etapy tworzenia programu w języku C++

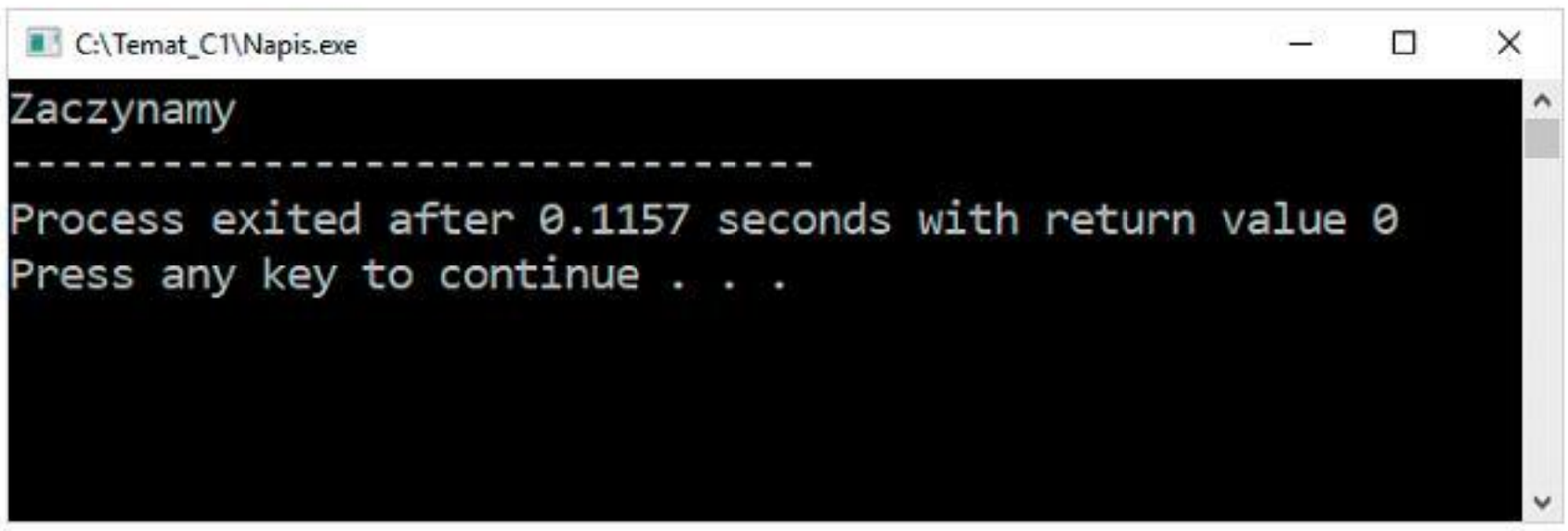
1. Utwórz nowy plik źródłowy i napisz program wyświetlający na ekranie napis „Zaczynamy" – przepisz instrukcje pokazane na rysunku 3.
2. Zapisz program w pliku pod nazwą *Napis*.
3. Skompiluj program.
4. Uruchom program.
5. Jeśli kompilator wykrył błędy, popraw je. Zapisz program w pliku pod tą samą nazwą. Ponownie skompiluj i uruchom program.

W trakcie wykonywania programu procesor rozpoznaje i wykonuje instrukcje swojego wewnętrznego języka. Po wykonaniu programu wyniki pojawią się na wybranym urządzeniu zewnętrznym, np. na ekranie monitora (rys. 4.).

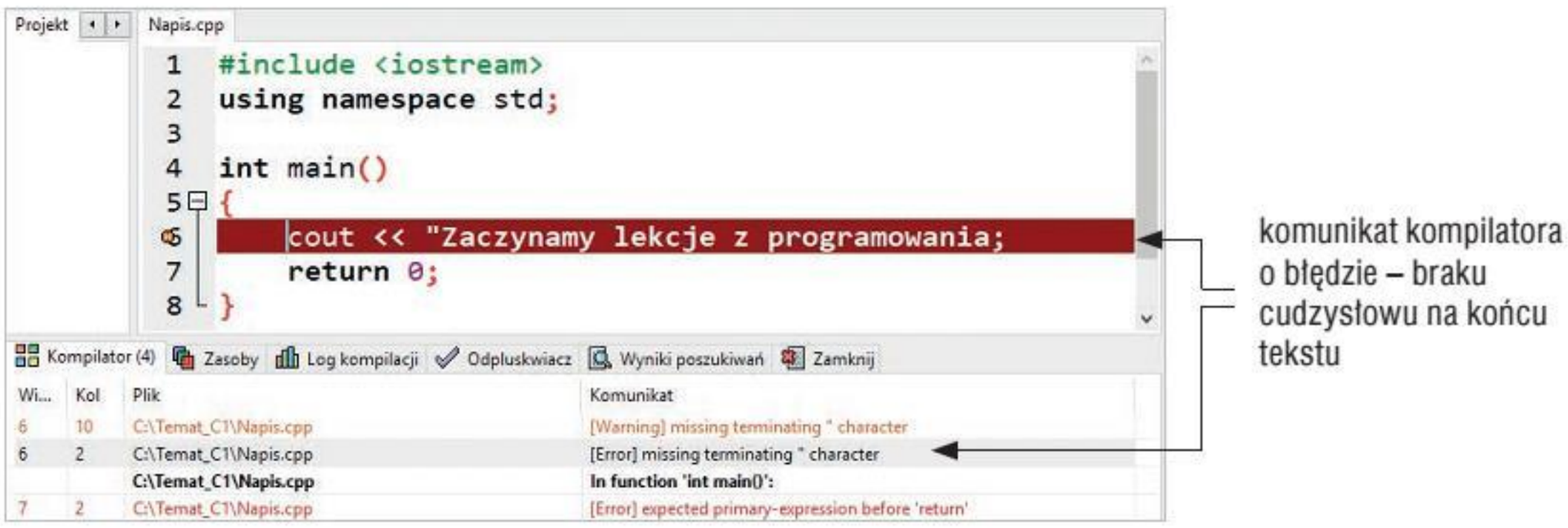
Aby powiększyć czcionkę w oknie wykonania programu, należy kliknąć przycisk w lewym górnym rogu okna i wybrać polecenie **Właściwości/Czcionka**.

Dobra rada

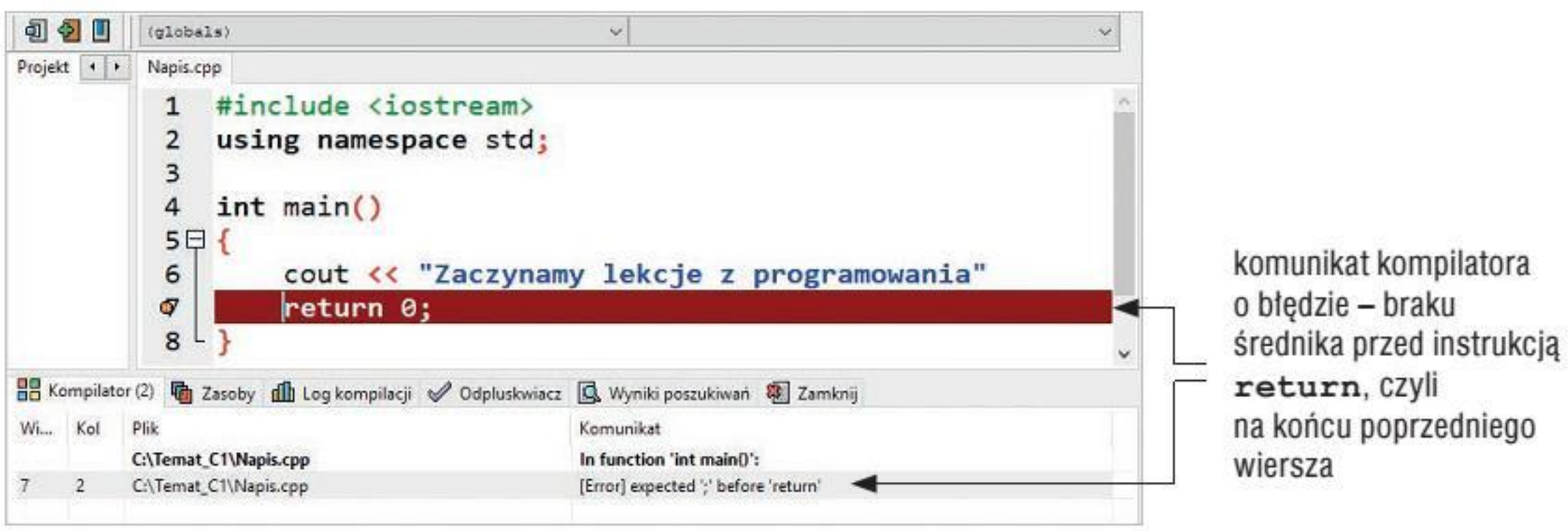
Staraj się samodzielnie poprawiać program do momentu, gdy kompilacja przebiegnie prawidłowo i program będzie można uruchomić. W nauce programowania umiejętność wyszukiwania i poprawiania błędów w programie jest bardzo ważna.



Rys. 4. Wynik działania programu – ćwiczenie 1.



Rys. 5a. Okno środowiska programistycznego języka C++ z programem źródłowym – kompilator wykrył błąd w programie



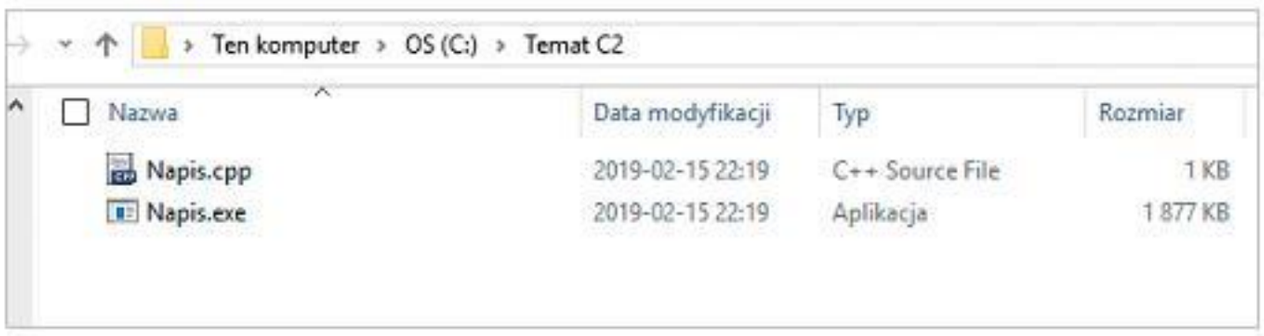
Rys. 5b. Okno środowiska programistycznego języka C++ z programem źródłowym – kompilator wykrył błąd w programie

Ćwiczenie 2. Modyfikujemy program

1. Zmodyfikuj program utworzony w ćwiczeniu 1., dodając dalszą część napisu: „lekcje z programowania".
2. Zapisz plik pod tą samą nazwą.
3. Jeśli kompilator wykrył błędy, popraw je. Ponownie zapisz plik i skompiluj program.
4. Jeśli kompilacja przebiegła pomyślnie, uruchom program.

Wskazówka: Na rysunkach 5a i 5b pokazano przykładowe błędy wykryte przez kompilator. Oba błędy są w szóstym wierszu programu, ale na rysunku 5b kompilator podświetlił wiersz siódmy, ponieważ brakuje średnika na końcu poprzedniego wiersza.

Po skompilowaniu programu źródłowego zapisanego w pliku *Napis.cpp* zostanie utworzony plik wykonywalny *Napis.exe* (aplikacja) o takiej samej nazwie własnej (*Napis*), ale z rozszerzeniem *exe* (rys. 6.). Aby uruchomić program po zamknięciu okna środowiska programistycznego, wystarczy kliknąć dwukrotnie nazwę lub ikonę *Napis.exe*. Program uruchomi się w oknie pokazanym na rysunku 4.



Rys. 6. W oknie Eksploratora plików widać dwa pliki – źródłowy i wykonywalny

2. Stosowanie zmiennych

Chcemy obliczyć iloczyn dwóch liczb całkowitych. Jak napisać program w języku C++ umożliwiający wprowadzenie dwóch liczb całkowitych z klawiatury, obliczający ich iloczyn i wyprowadzający wynik obliczeń na ekran?

Program, który utworzyliśmy w ćwiczeniach 1. i 2., wyświetlał na ekranie napis – nie używaliśmy w nim zmiennych i nie wykonywaliśmy obliczeń. Struktura kolejnych programów będzie bardziej rozbudowana, m.in. dodamy **deklarację zmiennych**.

2.1. Deklarowanie zmiennych

W wielu językach programowania, w tym w języku C++, zmienne przed użyciem należy **zadeklarować**. Deklaruje się zmienne, w których będą przechowywane dane wejściowe dla programu, wyniki działania programu, a także dane pomocnicze niezbędne do wykonania obliczeń. W deklaracji zmiennej podaje się jej **typ** oraz nazwę.

Komórka pamięci K

Część pamięci operacyjnej komputera (pamięci RAM). Pamięć operacyjna jest podzielona logicznie na komórki pamięci. Każda komórka pamięci ma swój unikatowy adres. W komórkach pamięci mogą być przechowywane dane reprezentujące instrukcje procesora lub dane dla tych instrukcji.

Typ zmiennej T

Określa rodzaj danych przechowywanych w zmiennej – mogą to być np. liczby, znaki, wartości logiczne (**typy proste**) lub teksty, tablice, rekordy i obiekty (**typy złożone**).

Wykorzystywanym w programie zmiennym typu prostego przyporządkowane są określone **komórki pamięci**.

Dla każdej zadeklarowanej zmiennej rezerwowany jest fragment pamięci o określonym adresie i wielkości odpowiadającej danemu typowi danych. Na przykład dla zmiennych typu całkowitego **`int`** to cztery bajty lub osiem bajtów (zależnie od używanego kompilatora), dla zmiennych typu rzeczywistego **`float`** – cztery bajty, a wartość zmiennej znakowej **`char`** pamiętana jest w jednym bajcie lub w dwóch bajtach.



Użycie w programie niezadeklarowanej zmiennej zostanie zgłoszone podczas kompilacji jako błąd (program nie zostanie skompilowany).

Języki programowania oferują różne typy danych, które dobieramy zależnie od zadania, np. liczby całkowite, gdy mamy obliczyć sumę liczb całkowitych, lub znaki, gdy mamy sprawdzić, czy wprowadzona litera jest samogłoską.

Deklarowanie zmiennych	*opis_typu lista_zmiennych;*



Przykład 1. Deklarowanie zmiennych w języku C++

Deklarowanie zmiennych typu całkowitego **`(int)`**

```
int liczba;
int a, b, Suma;
int a, b, iloczyn;
int liczba_sztuk, dlugosc;
```

Deklarowanie zmiennych typu rzeczywistego **`(float)`**

```
float minimum;
float a, b, c, suma, srednia;
float Cena_jedn, Dlugosc;
```

Zasady dotyczące nazw zmiennych w języku C++

1. Wielkie i małe litery w nazwach są traktowane odmiennie (np. *suma* i *Suma* będą oznaczać różne zmienne). Pisząc program w języku C++, należy zwracać uwagę na poprawne używanie małych i wielkich liter.
2. W nazwach zmiennych można stosować litery (bez polskich znaków diakrytycznych), cyfry i znak podkreślenia. Nazwa nie może zaczynać się od cyfry.
3. W nazwach zmiennych nie wolno stosować spacji.

2.2. Nadawanie zmiennym wartości

Zmiennej stosowanej w programie możemy nadać wartość za pomocą **instrukcji przypisania** lub wprowadzić wartość ze standardowego wejścia (zwykle klawiatury), stosując polecenie `cin >>`.

Wprowadzanie danych z klawiatury	`cin >> nazwa_zmiennej;`

W języku C++ `cin` jest **obiektem** reprezentującym standardowe wejście. **Operator** >> oznacza wprowadzenie danych do odpowiedniej zmiennej, podanej po prawej stronie. Jeżeli chcemy wprowadzić dane do kilku zmiennych, należy użyć operatora >> wielokrotnie.

Przykład 2. Wprowadzanie danych

```
cin >> dlugosc;
cin >> a >> b;
cin >> Liczba1 >> Liczba2 >> Liczba3;
```

Instrukcja przypisania	*zmienna = wyrażenie;*

W instrukcji przypisania zmiennej podanej po lewej stronie instrukcji zostanie przypisana obliczona przez komputer wartość wyrażenia znajdującego się po prawej stronie instrukcji. *Wyrażeniem* mogą być m.in. konkretna wartość, zmienna, wyrażenie algebraiczne.

Przykład 3. Stosowanie instrukcji przypisania w języku C++

```
Rok = 2019;
a = 7.58;
x = y;
Obwod = 2 * a + 2 * b;
```

Jeśli w tym samym programie umieścimy ciąg instrukcji:

```
suma = 100;
a = 20;
suma = 100 + a;
```

to po wykonaniu tych instrukcji wartość zmiennej *suma* będzie wynosić 120.

Operacji przypisania można użyć wielokrotnie w ramach jednej instrukcji, np. zapis:

```
a = b = c = 150;
```

oznacza, że wartość 150 zostanie przypisana najpierw zmiennej *c*, potem wartość zmiennej *c* zostanie przypisana zmiennej *b*, a na koniec wartość zmiennej *b* zostanie przypisana zmiennej *a*. Po wykonaniu instrukcji zmienne *a*, *b* i *c* będą miały wartość 150.

Uwaga

Jeśli w programie w instrukcji przypisania nadamy zmiennej nową wartość, to poprzednia wartość zostanie usunięta.

2.3. Wykonywanie obliczeń

W języku C++ obliczenia możemy zapisywać, korzystając z instrukcji przypisania oraz **operatorów arytmetycznych**. W obliczeniach stosujemy następujące podstawowe operatory arytmetyczne: + (dodawania), – (odejmowania), * (mnożenia), / (dzielenia) oraz % (reszty z dzielenia). Operator / oznacza dzielenie, w zależności od typu danych – całkowite (z obcięciem części ułamkowej) lub zmiennoprzecinkowe (z zachowaniem części ułamkowej), a operator % – obliczenie reszty z dzielenia dwóch liczb całkowitych.

Operator	Działanie	Przykład instrukcji przypisania	Wynik działania dla danych: a = 11 i b = 4
+	dodawanie	`suma = a + b;`	11 + 4 = 15
–	odejmowanie	`roznica = a - b;`	11 – 4 = 7
*	mnożenie	`iloczyn = a * b;`	11 * 4 = 44
/	dzielenie w zależności od typu danych: • całkowite (z obcięciem części ułamkowej) – dla wartości typu całkowitego	`iloraz = a / b;`	11 / 4 = 2
	• zmiennoprzecinkowe (z zachowaniem części ułamkowej) – jeśli przynajmniej jedna liczba jest wartością typu rzeczywistego	`iloraz = a / b;`	11 / 4 = 2.75
%	obliczenie reszty z dzielenia dwóch liczb całkowitych	`reszta = a % b;`	11 % 4 = 3

Tabela 1. Podstawowe operatory arytmetyczne w języku C++

Ćwiczenie 3. Tworzymy program z wykorzystaniem zmiennych

1. Na rysunku 7. pokazano program obliczający iloczyn dwóch liczb całkowitych *a* i *b* wprowadzanych z klawiatury.
2. Utwórz nowy plik źródłowy i przepisz program. Wskaż w programie: deklarację zmiennych, wprowadzanie danych z klawiatury, wykonywanie obliczeń.
3. Zapisz plik pod nazwą *Iloczyn*. Skompiluj i uruchom program.

[*] Iloczyn.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a, b, iloczyn;
6
7      cin >> a;
8      cin >> b;
9      iloczyn = a * b;
10
11     return 0;
12 }
```

Rys. 7. Program obliczający iloczyn dwóch liczb wprowadzanych z klawiatury – ćwiczenie 3.

3. Wyprowadzanie komunikatów i wyników na ekran monitora

W ćwiczeniach 1. i 2. wyprowadzaliśmy komunikaty na ekran monitora, używając polecenia: `cout <<`. Za pomocą tego polecenia można wyprowadzać również wyniki obliczeń.

Wyprowadzanie wyników i komunikatów na ekran monitora	`cout << wartość;`

W języku C++ `cout` jest **obiektem** reprezentującym standardowe wyjście programu (zwykle okno systemu operacyjnego na ekranie monitora).

Operator << oznacza wyprowadzenie wartości podanej po prawej stronie. Jeżeli chcemy wyprowadzić kilka wartości, należy użyć operatora << wielokrotnie.

W instrukcji wyprowadzania wyników i komunikatów na ekran monitora *wartość* może być np. zmienną, wyrażeniem, a także napisem. Napis umieszczany w programie, który ma zostać wyświetlony na ekranie monitora, musi być ujęty w górne cudzysłowy "".

Przykład 4. Wyprowadzanie komunikatów i wyników

```
cout << "Zaczynamy lekcje z programowania";
cout << P;
cout << a + b;
cout << "Obwod = " << 2 * a + 2 * b;
cout << "Obwod wynosi: " << obwod;
```

Umieszczenie w wierszu z instrukcją wyprowadzania danych zapisu `endl` lub znaków `"\n"` powoduje przejście kursora na początek następnego wiersza po wyświetleniu komunikatu lub wyniku, za którym umieszczono te znaki.

Polecenia:

```
cout << "Liczba dni wolnych" << endl;
cout << liczba;
```

lub

```
cout << "Liczba dni wolnych" << endl << liczba;
```

lub

```
cout << "Liczba dni wolnych" << "\n";
cout << liczba;
```

lub

```
cout << "Liczba dni wolnych\n" << liczba;
```

wyświetlą w taki sam sposób te same wyniki.

Uwaga

Jeśli wprowadzamy z klawiatury podwójny cudzysłów, edytor środowiska Dev-C++ od razu wypisuje obydwa górne cudzysłowy i ustawia pomiędzy nimi kursor: `""`. Podobnie dzieje się, gdy wprowadzamy otwierający nawias okrągły po nazwie funkcji, np. `main`: `int main()`, czy otwierający nawias klamrowy `{` zaczynający blok.

Ćwiczenie 4. Wyprowadzamy napisy i wynik na ekran

1. Do programu zapisanego w ćwiczeniu 3. dodaj wyświetlanie komunikatów i wyniku, podobnie jak pokazano na rysunku 8a.
2. Zapisz plik pod tą samą nazwą. Skompiluj i uruchom program.
3. Co sądzisz o programach pokazanych na rysunkach 8a i 8b? Wskaż kilka zasadniczych różnic.

Uwaga

Aby uniknąć problemów z poprawnym wyświetlaniem komunikatów, nie będziemy w nich używać polskich znaków diakrytycznych.

DOBRZE

[*] Iloczyn.cpp

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, iloczyn;

    cout << "podaj pierwsza liczbe: ";
    cin >> a;
    cout << "podaj druga liczbe: ";
    cin >> b;
    iloczyn = a * b;
    cout <<"Iloczyn wynosi: "<< iloczyn;

    return 0;
}
```

Rys. 8a. Program napisany czytelnie i przejrzyście – ćwiczenie 4.

ŹLE

[*] Iloczyn.cpp

```
#include <iostream>
using namespace std;
int main()
{int a , b,iloczyn ;
cout << "podaj pierwsza liczbe: " ;cin>> a;
    cout << "podaj druga liczbe: ";
cin >>b;iloczyn =a * b;cout <<"Iloczyn wynosi: "<<iloczyn;
        return 0;
}
```

Rys. 8b. Program mniej czytelny i nieprzejrzysty – ćwiczenie 4.

Ćwiczenie 5. Testujemy działanie programu obliczającego średnią arytmetyczną dla danych różnego typu

1. Do programu zapisanego w ćwiczeniu 4. dodaj obliczanie średniej arytmetycznej liczb *a* i *b*. Zadeklaruj zmienną *srednia* typu całkowitego **`int`**. Wynik wyprowadź na ekran. Dodaj odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Srednia_c*. Skompiluj go i uruchom. Przetestuj program dla par liczb (wartości zmiennych *a* i *b*): (2; 3), (14; 8); (7; 6).
3. Zmień typ wszystkich danych na **`float`**. Zapisz program pod nazwą *Srednia_r*, skompiluj i uruchom. Przetestuj program dla tych samych danych. Porównaj otrzymane wyniki z wynikami działania programu *Srednia_c*. Wyjaśnij, dlaczego dla tych samych danych otrzymujemy różne wyniki. Dodatkowo przetestuj program *Srednia_r* dla liczb: (342,3; 25,7) (3763,82; 109,87).

Wskazówka: Pamiętaj, aby liczby dziesiętne wprowadzać z kropką, a nie z przecinkiem.

Warto zapamiętać

- Aby utworzyć program w języku C++, piszemy kod źródłowy w specjalnym edytorze wbudowanym w środowisko programistyczne, zapisujemy program w pliku i kompilujemy. Jeśli kompilacja przebiegła prawidłowo, uruchamiamy program. Wynik działania programu pokaże się w osobnym oknie i zostanie utworzony plik wykonywalny z rozszerzeniem *exe*.
- Funkcja główna `main()`, rozpoczynająca działanie programu, musi wystąpić w każdym programie w języku C++. Inne elementy są opcjonalne.
- W języku C++ każda zmienna używana w programie musi zostać zadeklarowana przed pierwszym użyciem poprzez określenie jej typu i nazwy.
- Wartość zmiennej możemy wprowadzić z klawiatury (stosując polecenie `cin >> zmienna`) lub nadać zmiennej wartość za pomocą instrukcji przypisania.
- Obliczenia zapisujemy, korzystając z instrukcji przypisania. Zmiennej po lewej stronie znaku = zostanie przypisana wartość wyrażenia podanego po prawej stronie.
- Komunikaty i wyniki wyprowadzamy na ekran monitora, stosując polecenie `cout <<`.

Pytania i polecenia

1. Przedstaw i omów na konkretnym przykładzie etapy tworzenia programu komputerowego.
2. Jaka jest ogólna struktura programu w języku C++?
3. Czym jest komórka pamięci?
4. Na czym polega deklaracja zmiennych w programie komputerowym? Wyjaśnij sposób deklarowania zmiennych w języku C++.
5. Przedstaw na przykładzie sposób wprowadzania danych z klawiatury.
6. Do czego służy instrukcja przypisania?
7. Jaka jest rola zapisów `endl` i `"\n"` umieszczonych w instrukcji wyprowadzania danych?
8. Wyjaśnij zapisy:
   a. `int liczba1, liczba2;`
   b. `float iloraz;`
   c. `cin >> liczba1;`
   d. `iloraz = liczba1 / liczba2;`
   e. `cout << "Iloraz wynosi: " << iloraz << endl;`
9. Wskaż nieprawidłowo zapisane instrukcje. Wyjaśnij, na czym polegają błędy:
   a. `cin >> 78;`
   b. `cin >> droga >> czas;`
   c. `cout >> "Podaj liczbę: ";`
   d. `cout << "Suma wynosi: " << a + b;`

Zadania

Uwagi:

- W edytorze języka C++ można kopiować (**Edycja**/**Kopiuj**) i wklejać (**Edycja**/**Wklej**) fragmenty kodu programu – w tym samym dokumencie i pomiędzy dokumentami (tak jak w edytorze tekstu). Ta możliwość jest przydatna, gdy ponownie wykorzystujemy te same lub podobne fragmenty kodu.
- Staraj się dodawać do programu komunikaty (również komentarze) oraz pisać przejrzyste programy.
- Każdy program należy skompilować, uruchomić i przetestować dla różnych danych, nawet jeśli w zadaniu nie ma takiego polecenia.

1. Napisz specyfikację zadania i program obliczający pole powierzchni prostokąta dla danych długości boków *a* i *b*, wprowadzanych z klawiatury. Po uruchomieniu programu na ekranie powinny pojawiać się w kolejnych wierszach komunikaty: „Podaj dlugosc boku a:", „Podaj dlugosc boku b:". Po podaniu danych, w trzecim wierszu powinny wyświetlić się: napis „Pole prostokąta wynosi" oraz wartość pola. Zapisz program w pliku pod nazwą *Pole*.
2. Napisz specyfikację zadania i program realizujący algorytm obliczania kwadratu i sześcianu wprowadzanej z klawiatury liczby rzeczywistej. Zapisz program w pliku pod nazwą *Potegi*.
3. Napisz specyfikację zadania i program obliczający drogę *S* przebytą w czasie *t* przez pojazd poruszający się ze średnią prędkością *v*. Zapisz program w pliku pod nazwą *Droga*.
4. Otwórz plik przekazany przez nauczyciela. W pliku zapisano program zawierający błędy, pokazany na rysunku 9. Po wprowadzeniu z klawiatury liczby program powinien wyświetlić na ekranie liczbę o 2 większą od wprowadzonej. Skompiluj program. Jakie błędy pojawiły się w zapisie programu? Zmodyfikuj program, aby działał poprawnie. Zapisz ponownie plik pod tą samą nazwą, skompiluj program, uruchom i sprawdź jego działanie.

[*] TC1_z4.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int liczba1;
7      cin >> liczba1
8      cout << liczba2
9      liczba2 = liczba1 + 2;
10     return 0;
11 }
```

Rys. 9. Program z błędami – zadanie 4.

5. Napisz specyfikację i program do zadania: *Jaki procent liczby* a *stanowi liczba* b? Zapisz program w pliku pod nazwą *Procenty*.

Dla zainteresowanych

6. Dane są suma i iloczyn dwóch liczb całkowitych. Napisz specyfikację zadania i program znajdujący te liczby. Zapisz program w pliku pod nazwą *Liczby*.

Temat 55.

Algorytmy z warunkami w języku C++

1. **Instrukcja warunkowa w języku C++**
2. **Stosowanie warunków logicznych**
   - 2.1. Proste warunki logiczne
   - 2.2. Złożone warunki logiczne

Warto powtórzyć

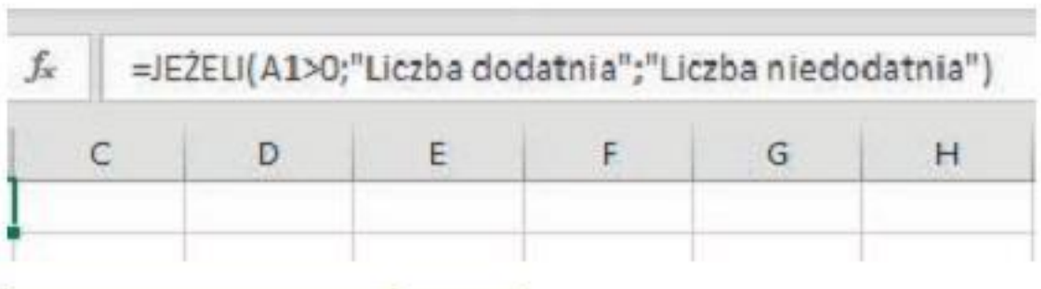
1. Na czym polega sytuacja warunkowa? Podaj dwa przykłady.
2. Co określa typ zmiennej?
3. Czy w języku C++ można użyć w programie zmiennej niezadeklarowanej? Uzasadnij odpowiedź.

1. Instrukcja warunkowa w języku C++

Chcemy napisać program sprawdzający, czy wprowadzona z klawiatury liczba jest dodatnia czy niedodatnia (ujemna albo równa zero). Jak zrealizować ten algorytm w arkuszu kalkulacyjnym? W jaki sposób napisać program realizujący sytuację warunkową w języku C++?

Aby w wybranym języku programowania (w tym w języku C++) napisać program realizujący algorytm z warunkami, korzystamy z **instrukcji warunkowej**, której postać i działanie są zbliżone do funkcji JEŻELI arkusza kalkulacyjnego: JEŻELI(*test_logiczny*;*wartość_jeżeli_prawda*;*wartość_jeżeli_fałsz*).

Arkusz kalkulacyjny Excel



Baltie



C++

```
if(a > 0)
    cout << "Liczba dodatnia";
else
    cout << "Liczba niedodatnia";
```

Rys. 1. Zastosowanie funkcji JEŻELI w arkuszu kalkulacyjnym i instrukcji warunkowej w dydaktycznym środowisku programowania Baltie oraz w języku programowania C++ do realizacji tego samego algorytmu z warunkiem

Ćwiczenie 1. Analizujemy sposób realizacji sytuacji warunkowej w różnych narzędziach

Porównaj sposób zapisu sytuacji warunkowej w różnych narzędziach. Przeanalizuj rysunek 1., zwracając szczególną uwagę na postać instrukcji warunkowej. Wskaż podobieństwa i różnice.

Instrukcja warunkowa	`if(warunek)` `    lista_instrukcji1;` `else` `    lista_instrukcji2;` `kolejna_instrukcja;`
Uproszczona postać instrukcji warunkowej	`if(warunek)` `    lista_instrukcji1;` `kolejna_instrukcja;`

Działanie instrukcji warunkowej jest w większości języków programowania podobne. Sprawdzany jest *warunek* logiczny (po słowie kluczowym `if`) – jeśli jest prawdziwy, wykonywana jest instrukcja (instrukcje) umieszczona (umieszczone) po warunku (*lista_instrukcji1*); jeśli fałszywy – wykonywana jest instrukcja (instrukcje) po słowie `else` (*lista_instrukcji2*), a jeśli brak tej części instrukcji – wykonywana jest od razu *kolejna_instrukcja* zapisana po instrukcji warunkowej.

Jako *lista_instrukcji1* i *lista_instrukcji2* mogą wystąpić również kolejne instrukcje warunkowe lub **blok instrukcji** (**blok kodu**). W języku C++ blok instrukcji wyznaczają nawiasy klamrowe `{}`. *Warunek* po słowie kluczowym `if` może być prosty lub złożony.

Operator	Określenie	Przykład warunku	Interpretacja wyrażenia
==	równy	`a == b`	*a* równe *b*
!=	różny	`a != b`	*a* różne od *b*
<	mniejszy	`a < b`	*a* mniejsze od *b*
>	większy	`a > b`	*a* większe od *b*
>=	większy lub równy	`a >= b`	*a* większe lub równe *b*
<=	mniejszy lub równy	`a <= b`	*a* mniejsze lub równe *b*

Tabela 1. Podstawowe operatory porównania w języku C++

Do porównywania wartości używamy operatora porównania ==. Nie stosujemy w tym celu operatora przypisania =. W języku C++ zapisanie warunku za pomocą operatora przypisania, np. `if (x = 0)`, nie spowoduje błędu kompilacji, ale program nie będzie działał poprawnie.

2. Stosowanie warunków logicznych

W instrukcji warunkowej warunek po słowie kluczowym **`if`** może być:

- prostym warunkiem logicznym, np.:
  `(a == 9)`

lub

- złożonym warunkiem logicznym, np.:
  `(x > -5 && x <=0)`.

2.1. Proste warunki logiczne

Przykład 1. Zapisywanie algorytmu z warunkiem logicznym prostym w języku C++

Zadanie: Oblicz wartość bezwzględną dowolnej liczby całkowitej.

Dane: dowolna liczba całkowita: *x*.

Wynik: wartość bezwzględna liczby *x* równa *w*.

[*] Wartosc_bezw.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int x, w;
6      cout << "Podaj liczbe x: ";
7      cin >> x;
8      if(x >=0)
9          w = x;
10     else
11         w = -x;
12     cout << "Wartosc bezwzgledna liczby x = " << w;
13
14     return 0;
15 }
```

Ćwiczenie 2. Realizujemy algorytm obliczania wartości bezwzględnej liczby w języku C++

1. Przepisz program podany w przykładzie 1. Sprawdź, czy program jest zgodny ze specyfikacją zadania. Wyjaśnij, która instrukcja zostanie wykonana, gdy warunek będzie prawdziwy, a która – gdy warunek będzie fałszywy.
2. Zapisz program w pliku pod nazwą *Wart_bezwzgl*. Skompiluj program i wykonaj dla kilku różnych wartości *x*.

Ćwiczenie 3. Realizujemy algorytm z warunkiem logicznym prostym w języku C++

1. Napisz specyfikację zadania i program realizujący algorytm sprawdzania, czy wprowadzona liczba jest dodatnia czy niedodatnia. Wykorzystaj zapis instrukcji warunkowej z rysunku 1.
2. Zapisz program w pliku pod nazwą *Liczby*. Skompiluj program i wykonaj dla kilku różnych wartości zmiennych.

2.2. Złożone warunki logiczne

Wszystkie klasy twojej szkoły rywalizują w tzw. lidze klas. Zdobycie największej liczby punktów oznacza wygraną. Liczba punktów zależy m.in. od frekwencji i średniej ocen w ostatnim półroczu.

W jaki sposób napisać program, w którym będziemy mogli obliczać punkty zależnie od spełnienia obydwu warunków?

Jest to przykład sytuacji warunkowej z warunkiem logicznym złożonym. Jeśli frekwencja i średnia ocen spełniają podane warunki, to klasa uzyskuje punkty. Mamy tu do czynienia z logiczną koniunkcją. W wybranym języku programowania zastosujemy zatem instrukcję warunkową z warunkiem złożonym. W instrukcjach warunkowych warunki złożone zapisujemy, stosując odpowiednie operatory logiczne (tabela 2.).

Operator	Określenie	Przykład warunku logicznego	Interpretacja wyrażenia
`||`	alternatywa logiczna (lub)	`a < -5 || a > 5`	*a* mniejsze od –5 lub *a* większe od 5
`&&`	koniunkcja logiczna (i)	`a > 0 && a < 10`	*a* większe od zera i *a* mniejsze od 10

Tabela 2. Operatory logiczne w języku C++

Przykład 2. Zapisywanie algorytmu z warunkiem logicznym złożonym w języku C++

Zadanie: Pewna klasa uczestniczy w lidze klas. Napisz program obliczający, ile wynosi liczba punktów (*pkt*) uzyskanych przez daną klasę zależnie od frekwencji (*f*) i średniej ocen (*so*) w ostatnim półroczu. Klasa otrzymuje dodatkowe 20 punktów, jeśli frekwencja jest powyżej 94% ($f > 94\%$) i średnia ocen nie jest mniejsza niż 4,0 ($so \geq 4{,}0$). Zdobytą wcześniej liczbę punktów (w innych kategoriach), średnią ocen i frekwencję wprowadzaj z klawiatury, a aktualną liczbę punktów wyświetlaj na ekranie.

Dane: liczba całkowita $pkt > 0$, oznaczająca punkty zdobyte przez klasę w innych kategoriach, liczba rzeczywista $f > 0$, oznaczająca procent frekwencji, liczba rzeczywista *so*, oznaczająca średnią ocen.

Wynik: liczba całkowita *pkt*, oznaczająca aktualną liczbę punktów danej klasy.

Opis rozwiązania: W rozwiązaniu zastosujemy instrukcję warunkową (w wersji uproszczonej) z warunkiem złożonym zawierającym operator koniunkcji `&&`. Zastosujemy wersję uproszczoną, ponieważ niezależnie od tego, czy warunek zostanie spełniony czy nie, na ekranie ma się wyświetlać aktualna liczba punktów.

Uwaga

Na razie w programach nie sprawdzamy poprawności danych wprowadzanych przez użytkownika. Zakładamy, że użytkownik podaje poprawne dane. Problem poprawności danych omówimy w temacie 56.

[*] Liga.cpp

```cpp
#include <iostream>
using namespace std;
int main()
{
    int pkt;
    float f, so;

    cout << "Podaj liczbe punktow zdobytych przez klase: ";
    cin >> pkt;
    cout << "Podaj frekwencje klasy: ";
    cin >> f;
    cout << "Podaj srednia ocen klasy: ";
    cin >> so;
    if(f > 94 && so >= 4.0)
        pkt = pkt + 20;
    cout << "Aktualna liczba punktow wynosi: " << pkt;

    return 0;
}
```

Ćwiczenie 4. Realizujemy algorytm z warunkiem logicznym złożonym w języku C++

1. Przepisz program podany w przykładzie 2. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Dlaczego zastosowano instrukcję warunkową w wersji uproszczonej (bez części z **`else`**)? Uzasadnij odpowiedź.
2. Wykonaj program dla kilku różnych wartości zmiennych.

Ćwiczenie 5. Realizujemy algorytm z warunkiem logicznym złożonym w języku C++

1. Klasa uczestniczy w lidze klas. Jeśli klasa w ostatnim semestrze uzyskała frekwencję powyżej 93% lub średnią ocen powyżej 4,2, to w nagrodę pojedzie na wycieczkę. Napisz program sprawdzający, czy klasa zakwalifikuje się na wycieczkę. Frekwencję i średnią ocen wprowadzaj z klawiatury, a komunikat „nagroda” lub „brak nagrody” wyświetlaj na ekranie.
2. Zapisz program w pliku pod nazwą *Wycieczka*.

! Uwaga

W języku C++ musimy skompilować program przed uruchomieniem. Nie będziemy o tym przypominać w każdym ćwiczeniu i zadaniu.

Warto zapamiętać

- W algorytmie z warunkami występują sytuacje warunkowe – wynik lub dalsze działanie algorytmu zależą od spełnienia lub niespełnienia warunku.
- W językach programowania do zapisania algorytmów z warunkami stosujemy instrukcje warunkowe. Działanie tych instrukcji jest podobne w większości języków programowania.
- Warunek w instrukcji warunkowej może być prosty lub złożony. W warunkach złożonych w języku C++ stosujemy operatory alternatywy (`||`) i koniunkcji logicznej (`&&`).

Pytania i polecenia

1. Wyjaśnij, na czym polega sytuacja warunkowa. Podaj przykład sytuacji warunkowej.
2. W jaki sposób zapisuje się w językach programowania algorytmy zawierające sytuacje warunkowe?
3. Omów działanie instrukcji warunkowej w wybranym języku programowania. Porównaj działanie tej instrukcji do działania funkcji JEŻELI w arkuszu kalkulacyjnym.
4. Podaj dwa przykłady sposobu zapisania złożonych warunków logicznych.

Zadania

1. Kiedy zostanie wykonana instrukcja `cout << x - y;` w programach z podpunktów a i b? Rozważ przypadki: $x > y$, $x = y$, $x < y$. Czym różni się efekt działania instrukcji **`if`** w punktach a i b?

a.
```
cin >> x >> y;
if(x > y)
    x = x - y;
cout << x - y;
```

b.
```
cin >> x >> y;
if(x > y)
    x = x - y;
else
    cout << x - y;
```

2. Napisz specyfikację zadania i program realizujący algorytm sprawdzania, czy wprowadzona z klawiatury liczba jest parzysta czy nieparzysta. Zapisz program w pliku pod nazwą *Parzyste_nieparzyste*.
   Wskazówka: Oblicz resztę z dzielenia wprowadzonej liczby przez 2.
3. Napisz specyfikację zadania i program realizujący algorytm sprawdzania, czy liczba jest podzielna przez 3. Zapisz program w pliku pod nazwą *Podzielna_przez_3*.
   Wskazówka: Oblicz resztę z dzielenia wprowadzonej liczby przez 3.
4. Klasa ma być podzielona na dwie grupy, w zależności od wyniku testu kompetencji językowych z wybranego języka obcego lub oceny na świadectwie ukończenia szkoły podstawowej. Jeśli dany uczeń uzyskał z testu wynik powyżej 90% punktów lub jeśli na świadectwie ukończenia szkoły podstawowej miał z danego języka ocenę nie niższą niż 5, to kwalifikuje się do grupy zaawansowanej. W przeciwnym wypadku kwalifikuje się do grupy podstawowej. Napisz program sprawdzający, do której grupy zakwalifikuje się dany uczeń. Liczbę punktów uzyskanych z testu i ocenę ze świadectwa wprowadzaj z klawiatury, a komunikat: „Grupa zaawansowana" lub „Grupa podstawowa" wyświetlaj na ekranie. Zapisz program w pliku pod nazwą *Grupy*.

Dla zainteresowanych

5. Napisz specyfikację zadania i program sprawdzający, czy wprowadzone trzy liczby naturalne są bokami trójkąta prostokątnego. Zależnie od rezultatu program ma wyprowadzać komunikaty „Tak" lub „Nie". Zapisz program w pliku pod nazwą *Trojkat_prostokat*.

Temat 56. Sprawdzanie poprawności danych i zapisywanie rozwiązania problemu w języku C++

1. Sprawdzanie poprawności danych
2. Zapisywanie rozwiązania przykładowego problemu

Warto powtórzyć

1. W jaki sposób możemy nadać wartości zmiennym w języku C++?
2. Jak działa instrukcja warunkowa w języku C++?
3. Jak wyróżnić blok instrukcji (blok kodu) w języku C++?

1. Sprawdzanie poprawności danych

Chcemy napisać program obliczający pierwiastek kwadratowy danej liczby. Pierwiastki kwadratowe w zbiorze liczb rzeczywistych można jednak obliczyć tylko dla liczb nieujemnych. Jak sprawdzić w programie, czy wprowadzona przez użytkownika liczba jest nieujemna?

Podczas pisania programów ważne jest sprawdzanie poprawności wprowadzanych danych. Dzięki temu jesteśmy zabezpieczeni przed niespodziewanym przerwaniem pracy programu oraz zapewniamy zgodność danych wejściowych ze specyfikacją zadania.

Poprawny program powinien spełniać kilka kryteriów:

- rozwiązywać problem, dla którego został utworzony;
- rozwiązywać problem dla wszystkich danych określonych w specyfikacji i odpowiednio reagować na wprowadzanie niepoprawnych danych;
- rozwiązywać problem w sposób jak najbardziej efektywny, optymalnie wykorzystując zasoby komputera.

Z punktu widzenia wygody obsługi programu wskazane byłoby, aby użytkownik był natychmiast powiadamiany o wprowadzeniu niepoprawnych danych i mógł je od razu poprawić.

Przykład 1. Sprawdzanie poprawności danych

Zadanie: Napisz program obliczający pierwiastek kwadratowy *p* z liczby rzeczywistej nieujemnej *x*, wprowadzanej z klawiatury. Program ma sprawdzać, czy wprowadzona przez użytkownika liczba jest poprawna (nieujemna) i wyprowadzać na ekran wynik obliczenia lub komunikat o błędnych danych.

Dane: liczba rzeczywista *x*.

Wyniki: pierwiastek kwadratowy *p* z liczby *x* lub komunikat „Wprowadzono liczbe mniejsza od zera".

Opis rozwiązania: Aby sprawdzić, czy wprowadzona przez użytkownika liczba spełnia warunki określone w zadaniu, zastosujemy instrukcję warunkową. Pierwiastki kwadratowe w zbiorze liczb rzeczywistych można obliczyć tylko dla liczb nieujemnych. W programie sprawdzamy, czy wprowadzona liczba *x* jest ujemna. Jeśli tak, wyprowadzamy komunikat o błędnych danych, w przeciwnym wypadku wykonujemy obliczenia i wyprowadzamy wynik na ekran.

Wykorzystamy standardową funkcję matematyczną `sqrt()`, służącą do obliczania pierwiastka kwadratowego.

[*] Pierwiastek.cpp

```
2  #include <cmath>        /* sqrt */
3  using namespace std;
4  int main()
5  {
6      float x, p;
7
8      cout << "Podaj x: ";
9      cin >> x;
10     if(x < 0)
11         cout << "Wprowadzono liczbe mniejsza od zera";
12     else
13     {
14         p = sqrt(x);
15         cout << "Pierwiastek z " << x << " wynosi " << p;
16     }
17     return 0;
18 }
```

dołączenie biblioteki standardowej `cmath` zawierającej funkcje matematyczne, m.in. funkcję `sqrt()` zwracającą pierwiastek kwadratowy z nieujemnej liczby rzeczywistej

wywołanie funkcji `sqrt()`

Ćwiczenie 1. Analizujemy gotowy program

1. Przepisz program podany w przykładzie 1. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Dlaczego zastosowano instrukcję warunkową w pełnej wersji (z **`else`**)? Wykonaj program dla kilku różnych wartości zmiennych.
2. Sprawdź, czy program jest zgodny ze specyfikacją zadania. Wyjaśnij, która instrukcja zostanie wykonana, gdy warunek będzie prawdziwy, a która – gdy warunek będzie fałszywy.

Ćwiczenie 2. Sprawdzamy poprawność wprowadzanych danych liczbowych

1. Napisz specyfikację zadania i program obliczający pole trójkąta. W programie uwzględnij sprawdzanie poprawności wprowadzanych danych (podstawy i wysokości trójkąta).
2. Zapisz program w pliku pod nazwą *Pole_trojkata*.

2. Zapisywanie rozwiązania przykładowego problemu

Chcesz przygotować tradycyjną sałatkę jarzynową na imprezę urodzinową. W Internecie są przepisy, ale w proporcjach dla dwóch, pięciu lub dziesięciu osób. Nie możesz znaleźć przepisu dla trzynastu osób – uczestników spotkania. Jak napisać program w języku C++, który obliczy, ile potrzebujemy danego składnika sałatki i wyświetli wynik na ekranie?

Przykład 2. Sformułowanie zadania, specyfikacja i projekt rozwiązania

Zadanie: Napisz program obliczający, ile danego składnika sałatki potrzeba dla *o2* osób, gdy znasz ilość składnika dla *o1* osób.

Dane: liczba całkowita $o1 > 0$, oznaczająca liczbę osób uwzględnioną w przepisie,
liczba rzeczywista $s1 > 0$, określająca ilość składnika dla *o1* osób,
liczba całkowita $o2 > 0$, oznaczająca liczbę uczestników spotkania.

Wynik: liczba rzeczywista $s2 > 0$, oznaczająca ilość składnika dla *o2* osób.



W rozwiązaniu skorzystamy z zależności:

jeśli

$s1$ ⟶ dla $o1$ osób
$s2$ ⟶ dla $o2$ osób

to

$$s2 = \frac{s1 \cdot o2}{o1}$$

Aby zapisać te obliczenia w języku C++, zastosujemy instrukcję przypisania:

```
s2 = (s1 * o2) / o1;
```

Uwaga: Z matematycznego punktu widzenia nawiasy są zbędne – umieściliśmy je w programie dla zwiększenia czytelności zapisu.

Ćwiczenie 3. Realizujemy algorytm obliczania ilości składnika sałatki w języku C++

1. Utwórz program realizujący zadanie sformułowane w przykładzie 2. Zadeklaruj zmienne zgodnie ze specyfikacją zadania. Dodaj komunikaty dotyczące wprowadzania zmiennych oraz komunikat przed wyprowadzeniem wyniku. Komunikaty i wynik wyprowadź na ekran monitora, jak pokazano na rysunku 1.
2. Zapisz program w pliku pod nazwą *Przepis*. Skompiluj i uruchom program.
   Przetestuj go dla następujących trójek liczb (wartości zmiennych *o1, o2*, *s1*): (10; 13; 2,5), (3; 12; 0,45), (15; 4; 3,20).
3. Przetestuj program dla trzech innych trójek liczb.

! Uwaga

W przykładach i ćwiczeniach dotyczących obliczeń nie uwzględniamy przypadków wprowadzenia danych innych niż liczba. Takie przypadki omówimy na informatyce realizowanej w zakresie rozszerzonym.

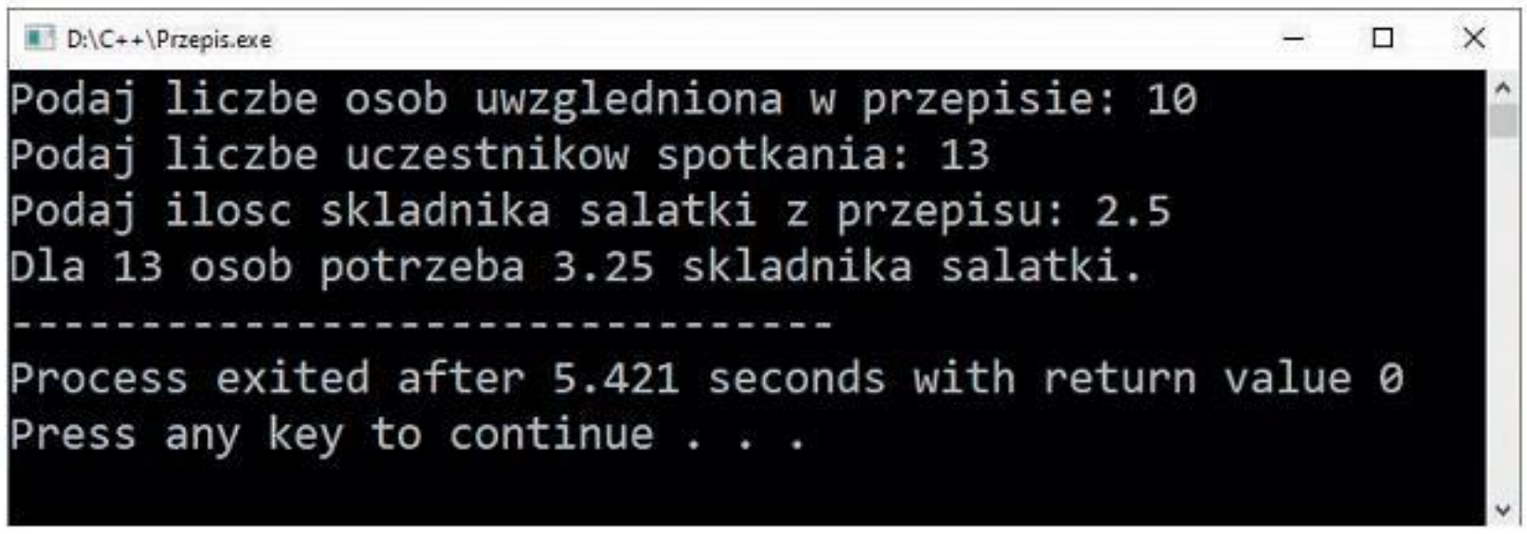
Wskazówki:

- Jeśli musimy zadeklarować zmienne różnych typów, warto dla czytelności programu zapisać deklaracje w dwóch kolejnych wierszach, np.:

```
int o1, o2;
float s1, s2;
```

- Aby wyświetlić wyniki, tak jak pokazano na rysunku 1., należy wpisać instrukcję:

```
cout << "Dla " << o2 << " osob potrzeba " << s2
<< " skladnika salatki.";
```



Rys. 1. Wynik działania programu – ćwiczenie 3.

Zauważmy, że program utworzony w ćwiczeniu 3. oblicza ilość tylko jednego składnika sałatki. Jeśli chcemy obliczyć ilość kolejnego składnika, musimy ponownie uruchomić program. W temacie 58. zmodyfikujemy ten program tak, aby obliczał i wyświetlał na ekranie kolejno ilości wszystkich składników.

Ćwiczenie 4. Dodajemy do programu sprawdzanie poprawności danych

1. Otwórz plik *Przepis* zapisany w ćwiczeniu 3. Dodaj do programu sprawdzanie poprawności wprowadzanych danych, zgodnie ze specyfikacją.
2. Zapisz plik pod tą samą nazwą.

Warto zapamiętać

- Program powinien efektywnie rozwiązywać problem dla wszystkich danych określonych w specyfikacji.
- Należy dbać o zgodność danych wejściowych ze specyfikacją i sprawdzać w programie, czy użytkownik wprowadził właściwe dane.
- Do sprawdzenia poprawności danych możemy zastosować w programie instrukcję warunkową.

Pytania i polecenia

1. Dlaczego ważne jest sprawdzanie poprawności danych w programie? Uzasadnij odpowiedź.
2. Jakie kryteria powinien spełniać poprawny program?
3. Podaj przykład sytuacji, w której należy sprawdzić poprawność danych.
4. Jakiej instrukcji możemy użyć do sprawdzenia poprawności danych w programie napisanym w języku C++?

Zadania

1. Napisz specyfikację zadania i program obliczający wysokość trójkąta, gdy podane są jego pole i podstawa. Zadbaj o poprawność wprowadzanych danych, zgodnie z zapisaną wcześniej specyfikacją. Zapisz program w pliku pod nazwą *Wysokosc_trojkata*.
2. Napisz specyfikację zadania i program realizujący algorytm obliczania objętości walca o wysokości h i promieniu podstawy r, gdzie h i r są dodatnimi liczbami rzeczywistymi. Zadbaj o poprawność wprowadzanych danych, zgodnie z zapisaną wcześniej specyfikacją. Dla liczb niedodatnich powinien być wyprowadzany napis „bledne dane". Zapisz program w pliku pod nazwą *Objetosc_walca*.
3. Napisz specyfikację zadania i program obliczający, ile litrów wody spadło na plac o powierzchni P metrów kwadratowych, jeśli pokryła go warstwa wody o grubości d milimetrów. Zadbaj o poprawność wprowadzanych danych, zgodnie z zapisaną wcześniej specyfikacją. Zapisz program w pliku pod nazwą *Deszcz*.
4. Napisz specyfikację zadania i opisz rozwiązanie: *Dany jest średni wiek trzech osób: S. Najmłodsza osoba ma* x *lat, najstarsza:* y. *Ile lat ma trzecia osoba? S, x, y to liczby naturalne różne od zera*. Napisz program na podstawie zapisanej specyfikacji. Przetestuj program dla następujących wartości zmiennych (S, x, y): (15, 10, 20), (27, 20, 40), (24, 14, 33). Dodaj kilka swoich propozycji danych. Zadbaj o poprawność wprowadzanych danych, zgodnie ze specyfikacją. Jakie powinny być wartości zmiennych: S oraz x i y, aby otrzymany wynik był sensowny? Uwzględnij te warunki w opisie algorytmu. Zapisz program w pliku pod nazwą *Wiek_3_osoby*.
5. Opisz wymyślony samodzielnie problem podobnie jak w punkcie 2. tego tematu. Ułóż zadanie, zapisz specyfikację i przygotuj rozwiązanie podobnie jak w przykładzie 2. Na koniec napisz program realizujący to zadanie. Zapisz program w pliku pod nazwą *Problem*.

Dla zainteresowanych

6. Napisz specyfikację i program do zadania: *Dane są: suma długości przekątnych rombu i długość jednego boku. Oblicz długości przekątnych rombu*. Zadbaj o poprawność wprowadzanych danych, zgodnie ze specyfikacją. Zapisz program w pliku pod nazwą *Romb*.
7. Napisz program obliczający pierwiastki równania kwadratowego $ax^2 + bx + c = 0$. Program powinien poprawnie reagować na błędne dane ($a = 0$) oraz rozróżniać sytuacje: $\Delta = 0$ i $\Delta > 0$. Zapisz program w pliku pod nazwą *Rownanie_kwadratowe*.

Temat 57.

Instrukcje warunkowe zagnieżdżone i instrukcja wyboru

1. Algorytmy z warunkami zagnieżdżonymi
2. Stosowanie instrukcji wyboru

Warto powtórzyć

1. Jak działa instrukcja iteracyjna?
2. Jak wyróżnić blok instrukcji (blok kodu) w języku C++?
3. Dla jakiej wartości warunku logicznego zostaną wykonane instrukcje umieszczone w instrukcji warunkowej po słowie `else`?

1. Algorytmy z warunkami zagnieżdżonymi

Chcemy napisać program realizujący algorytm wyboru najmniejszej z trzech różnych liczb całkowitych wprowadzanych z klawiatury. W jaki sposób po sprawdzeniu, czy pierwsza liczba jest mniejsza od drugiej, sprawdzać, czy pierwsza jest też mniejsza od trzeciej oraz druga od trzeciej?

Instrukcje warunkowe, podobnie jak funkcję JEŻELI w arkuszu kalkulacyjnym, można **zagnieżdżać**, czyli po warunku oraz po słowie `else` może wystąpić kolejna instrukcja warunkowa.

Ćwiczenie 1. Piszemy specyfikację problemu

Sformułuj zadanie i napisz specyfikację zadania do sytuacji problemowej podanej na początku tematu. Przyjmij następujące nazwy zmiennych: *a*, *b*, *c* (dla sprawdzanych liczb), *najmniejsza* (dla zapamiętania najmniejszej liczby).

Napiszemy program realizujący algorytm wyboru najmniejszej z trzech różnych liczb całkowitych. W rozwiązaniu zastosujemy zagnieżdżone instrukcje warunkowe.

[*] Najmniejsza_z_trzech.cpp

```cpp
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, najmniejsza;

    cout << "Podaj trzy liczby: " << endl;
    cin >> a >> b >> c;
    if(a < b)
    {
        if(a < c)
            najmniejsza = a;
        else
            najmniejsza = c;
    }
    else
    {
        if(b < c)
            najmniejsza = b;
        else
            najmniejsza = c;
    }
    cout << "Najmniejsza liczba wynosi: " << najmniejsza << endl;
    return 0;
}
```

Uwaga

W przypadku instrukcji zagnieżdżonych zaleca się pisać słowo kluczowe **`else`** od tej samej kolumny, co odpowiadające mu słowo **`if`** i stosować dodatkowe klamry przy zewnętrznej instrukcji **`if`** (dla zachowania czytelności programu).

Rys. 1. Zapisywanie algorytmu z warunkami zagnieżdżonymi w języku C++

Ćwiczenie 2. Stosujemy zagnieżdżone instrukcje warunkowe

1. Przepisz program pokazany na rysunku 1. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Sprawdź, czy program jest zgodny ze specyfikacją zadania napisaną w ćwiczeniu 1. Jeśli nie, popraw program. Zwróć uwagę na zapis zagnieżdżonych instrukcji warunkowych. Dlaczego w tym programie je zastosowano?
2. Zapisz program w pliku pod nazwą *Najmniejsza_z_trzech*. Uruchom program i przetestuj dla następujących trójek liczb (wartości zmiennych *a*, *b*, *c*): (3; -56; 0), (76; 123; -2), (-70; 40; -40), (56; 68; 88), (100; 0; 2).
3. Wyjaśnij, co dzieje się w poszczególnych wierszach programu. W zeszycie przedmiotowym dokończ opis: *Sprawdzamy, czy pierwsza liczba jest mniejsza od drugiej. Jeśli tak, to…*
4. Odpowiedz na pytanie: *Kiedy zostanie wykonane przypisanie* `najmniejsza = b`*?*

Ćwiczenie 3. Zapisujemy problem sprzedawcy komputerów w języku programowania

1. Przypomnij sobie problem sprzedawcy komputerów i, korzystając ze specyfikacji zadania i projektu rozwiązania z tematu 52. (przykłady 3-5), napisz program realizujący przedstawione rozwiązanie. Zadbaj o sprawdzanie poprawności wprowadzanych danych (zgodnie ze specyfikacją).
2. Zapisz program w pliku pod nazwą *Cena_komputera*, uruchom i przetestuj dla różnych wartości zmiennych (*w*, *cenak*, *cenap*).

2. Stosowanie instrukcji wyboru



Chcemy napisać program „Prosty kalkulator" umożliwiający wybór rodzaju obliczenia z wykorzystaniem menu. Po wprowadzeniu z klawiatury np. liczby 1 program powinien obliczać sumę dwóch liczb, a po wprowadzeniu liczby 2 – różnicę itd. Jak to zrobić w języku C++?

Jeśli chcemy utworzyć menu ekranowe, w którym użytkownik ma wybrać jedną z czterech opcji, możemy zastosować kilkakrotnie instrukcję warunkową **`if`**. We fragmencie programu pokazanym na rysunku 2. wybór rodzaju obliczeń zależy od spełnienia warunku (od wprowadzonej wartości zmiennej *opcja*).

```
12      cout << "Wybierz opcje menu: 1 - suma, 2 - roznica, 3 - iloczyn, 4 - iloraz: ";
13      cin >> opcja;
14      if(opcja == 1)
15          cout << "suma = " << x + y;
16      if(opcja == 2)
17          cout << "roznica = " << x - y;
18      if(opcja == 3)
19          cout << "iloczyn = " << x * y;
20      if(opcja == 4)
21          cout << "iloraz = " << x / y;
22      if(opcja != 1 && opcja != 2 && opcja != 3 && opcja != 4)
23          cout << "Nie ma takiej opcji";
```

Rys. 2. Stosowanie instrukcji warunkowej **`if`** do wykonania menu programu

Do tworzenia menu ekranowego, w którym użytkownik musi wybrać jedną z opcji, w języku C++ można, zamiast kilku instrukcji **`if`**, użyć instrukcji **`switch`**.

```
13      cin >> wybor;
14      switch(wybor)
15      {
16      case 1:
17          cout << x + y << endl;
18          break;
19      case 2:
20          cout << x - y << endl;
21          break;
22      case 3:
23          cout << x * y << endl;
24          break;
25      case 4:
26          cout << x / y << endl;
27          break;
28      default:
29          cout << "Nie ma takiej opcji" << endl;
30          break;
31      }
```

Rys. 3. Stosowanie instrukcji wyboru **`switch`** do wykonania menu programu

Instrukcja wyboru	**switch**(*wyrażenie*) { **case** *etykieta1*: *instrukcja1*; **break**; **case** *etykieta2*: *instrukcja2*; **break**; … **default**: *instrukcjan*; **break**; } *kolejna_instrukcja*;

W instrukcji **switch** *wyrażenie* musi mieć typ całkowity lub znakowy, a wszystkie *etykiety* muszą być stałymi: liczbami całkowitymi lub znakami. Najpierw obliczana jest wartość *wyrażenia*, po czym, zależnie od tego, której *etykiecie* będzie równa, zostanie wykonana dana *instrukcja*. Jeśli wynikiem *wyrażenia* nie będzie żadna z wartości podanych w *etykietach*, zostanie wykonana *instrukcjan* znajdująca się po opcjonalnym słowie **default**. Jeśli nie ma tego słowa, program przejdzie do *kolejnej_instrukcji*.

Ćwiczenie 4. Stosujemy instrukcję wyboru do utworzenia menu programu

1. Otwórz plik przekazany przez nauczyciela. W pliku zapisano rozwiązanie problemu „Prosty kalkulator". Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Sprawdź działanie wszystkich opcji programu (podaj również inne liczby niż 1, 2, 3 lub 4).
2. Zmodyfikuj program, stosując instrukcję wyboru **switch** (wzór na rysunku 3.). Zapisz program w pliku pod nazwą *Prosty _kalkulator_switch*.
3. Porównaj obydwa rozwiązania (z kilkoma instrukcjami **if** i z instrukcją **switch**). Wskaż różnice.

Warto zapamiętać

- W programach komputerowych można stosować zagnieżdżone instrukcje warunkowe.
- W przypadku gdy wykonywanie operacji jest uzależnione od różnych wartości zmiennej, w języku C++ można zastosować instrukcję wyboru.
- W instrukcji wyboru wykonywana jest jedna z instrukcji – oznaczona etykietą równą wartości wyrażenia umieszczonego w nawiasach po słowie kluczowym **switch**.
- Instrukcja wyboru umożliwia np. tworzenie menu programu, w którym musimy wybrać jedną z opcji.

Pytania i polecenia

1. Podaj przykład sytuacji problemowej, w której występują zagnieżdżające się sytuacje warunkowe.
2. Omów sposób zapisywania zagnieżdżonej instrukcji warunkowej.
3. Podaj przykład problemu, dla którego możemy zastosować instrukcję wyboru **`switch`**.
4. Na czym polega działanie instrukcji **`switch`**?

Zadania

1. Napisz specyfikację zadania i program realizujący algorytm sprawdzania, która z dwóch liczb rzeczywistych jest większa. Uwzględniaj przypadek, gdy liczby są równe. Zapisz program w pliku pod nazwą *Wieksza_z_dwoch*.
2. Napisz specyfikację zadania: *Wprowadź dowolną liczbę rzeczywistą. Gdy liczba jest dodatnia, obliczaj pole koła o promieniu równym tej liczbie. Dla liczby ujemnej podawaj jej wartość bezwzględną, a dla równej zero – wyprowadź napis „rowna zero”*. Napisz program na podstawie zapisanej specyfikacji. Zapisz program w pliku pod nazwą *Zadanie2*.
3. Napisz specyfikację zadania i program obliczający prędkość samochodu *v*, gdy dane są droga *s* i czas *t*, podane odpowiednio w km i h. Jeśli prędkość samochodu przekracza 90 km/h, wyświetlaj komunikat „za szybko”. Jeśli prędkość będzie równa 90 km/h, wyświetlaj komunikat „w sam raz”, a jeśli mniejsza, wyświetlaj komunikat „za wolno”. Program napisz na podstawie przygotowanej wcześniej specyfikacji. Zapisz program w pliku pod nazwą *Predkosc*.
4. Napisz program, który dla wprowadzonej liczby 1 oblicza pole kwadratu, dla 2 – pole prostokąta, dla 3 – pole trójkąta, dla 4 – pole trapezu. Dla innej liczby – komunikat „Nie ma takiej opcji”. Zapisz program w pliku pod nazwą *Pola_figur*.
5. Napisz program obliczający wartość funkcji przedstawionej na rysunku 4. Zapisz program w pliku pod nazwą *Wartosc_funkcji*.

$$f(x) = \begin{cases} -1 & dla\ x < -1 \\ x & dla\ -1 \le x \le 1 \\ 1 & dla\ x > 1 \end{cases}$$

Rys. 4. Wzór funkcji – zadanie 5.

Dla zainteresowanych

6. Napisz specyfikację zadania i program sprawdzający, czy z odcinków o długościach wprowadzonych przez użytkownika z klawiatury można zbudować trójkąt (przypomnij sobie warunek istnienia trójkąta z lekcji matematyki). Program ma sprawdzać dziesięć trójek liczb i zależnie od rezultatu wyprowadzać komunikaty: „mozna zbudowac trojkat” lub „nie mozna zbudowac trojkata”. Dodaj do programu sprawdzanie poprawności wprowadzanych danych. Zapisz plik pod nazwą *Trojkat*.
7. Otwórz plik *Najmniejsza_z_trzech* zapisany w ćwiczeniu 2. Zmodyfikuj program, aby był uwzględniony przypadek, gdy liczby są równe. Zapisz plik pod tą samą nazwą.
8. Wymyśl samodzielnie problem, w którym można wykorzystać instrukcję wyboru **`switch`** do utworzenia menu. Napisz w języku C++ program realizujący ten problem.

Temat 58.

Algorytmy iteracyjne w języku C++

1. Instrukcja iteracyjna `for` w języku C++
2. Zapisywanie rozwiązania problemu iteracyjnego w języku C++

Warto powtórzyć

1. Czym jest iteracja? Podaj przykład problemu, w którym występuje iteracja.
2. Jak działa instrukcja iteracyjna w wybranym dydaktycznym środowisku programowania (Baltie, Scratch)?
3. Czym jest blok instrukcji (blok kodu) i jak go wyróżniamy?
4. Jak działa iteracyjny algorytm sumowania *n* liczb?

1. Instrukcja iteracyjna `for` w języku C++

Chcemy napisać program obliczający sumę *n* liczb. W algorytmie mamy do czynienia z powtarzaniem poleceń wprowadzania liczby i dodawania. W jaki sposób napisać program realizujący iterację w języku C++?

W temacie 53. omówiliśmy iterację na przykładzie algorytmu sumowania *n* liczb. Zapisaliśmy specyfikację zadania i listę kroków tego algorytmu. W tym temacie zapiszemy go w postaci programu w języku C++.

Aby w języku programowania napisać program realizujący algorytm iteracyjny, stosujemy **instrukcje iteracyjne** (zwane też **instrukcjami pętli**).

W języku C++ są dostępne trzy instrukcje iteracyjne: **`for`**, **`while`** i **`do ... while`**. W tym temacie pokażemy stosowanie instrukcji **`for`** (stosowanie pozostałych instrukcji pętli omówimy w temacie 60.).

Instrukcję **`for`** możemy stosować, gdy liczba powtórzeń (iteracji) jest z góry określona lub wprowadzana z klawiatury po uruchomieniu programu. Do określania kroków iteracji (powtórzeń) w instrukcji **`for`** stosujemy zwykle tzw. **zmienną sterującą**.

Instrukcja iteracyjna `for`	**`for`**(*`wyrażenie_początkowe; warunek; wyrażenie_pętli`*) *`lista_instrukcji;`*

Najpierw wykonywane jest *wyrażenie_początkowe* (np. `i = 0`), zwykle ustalające wartość początkową zmiennej sterującej. Następnie, dopóki spełniony jest *warunek*, wykonywana jest *lista_instrukcji*, a potem obliczana wartość *wyrażenia_pętli*, służącego zazwyczaj do zwiększenia (lub zmniejszenia) wartości zmiennej sterującej. Jako *lista_instrukcji* może wystąpić pojedyncza instrukcja (w tym kolejna instrukcja pętli) lub więcej instrukcji ujętych w blok {}.

Zwyczajowo początkową wartość zmiennej sterującej przyjmuje się jako równą 0, a warunek zapisuje jako np. `i < n` (jeśli chcemy, aby liczba iteracji wynosiła *n*).

Wyrażenie `i++` oznacza „zmiennej *i* przypisz wartość *i* zwiększoną o 1". Jest to skrót instrukcji przypisania: `i = i + 1`.

Rodzaj operatora	Przykład zastosowania	Odpowiednik	Znaczenie
operator inkrementacji	`i++` lub `++i`	`i = i + 1`	przypisanie zmiennej *i* poprzedniej wartości tej zmiennej, powiększonej o 1
operator dekrementacji	`i--` lub `--i`	`i = i - 1`	przypisanie zmiennej *i* poprzedniej wartości tej zmiennej, pomniejszonej o 1

Tabela 1. Operatory inkrementacji i dekrementacji w języku C++

Przykład	Opis
`for(i = 0; i < 7; i++)` `  cout << "*";`	Zmienna sterująca *i* na początku ma wartość 0. Instrukcja `cout << "*";` zostanie wykonana siedem razy dla zmiennej *i* mniejszej od 7 (czyli równej kolejno: 0, 1, 2, 3, 4, 5, 6). Na ekranie wyświetli się siedem gwiazdek w jednym wierszu.
`for(i = 0; i <= n; i++)` `  cout << "Witaj" << endl;`	Instrukcja `cout "Witaj" << endl;` zostanie wykonana *n* + 1 razy dla zmiennej *i* mniejszej od *n* lub równej *n* (czyli równej kolejno: 0, 1, 2, 3,… *n*). Program wypisze w kolejnych wierszach *n* + 1 razy napis „Witaj".
`for(i = 0; i < k; i++)` `{` `  cout << "Podaj liczbe: ";` `  cin >> liczba;` `}`	Instrukcje ujęte w blok `{}` zostaną wykonane *k* razy dla zmiennej *i* równej kolejno: 0, 1, 2, 3, … *k* – 1. Program wyświetli napis „Podaj liczbe: ", po czym zaczeka na wprowadzenie wartości zmiennej *liczba*. Te czynności powtórzą się *k* razy.
`for(i = 20; i >= 1; i--)` `  cout << i << endl;`	Instrukcja `cout << i << endl;` zostanie wykonana 20 razy dla zmiennej *i* równej kolejno: 20, 19, 18, … 1. Program wypisze w kolejnych wierszach wartości zmiennej *i*.

Tabela 2. Przykłady stosowania instrukcji `for` w języku C++

Ćwiczenie 1. Stosujemy instrukcję `for` w języku C++

Korzystając z tabeli 2., napisz cztery programy:

a. Program wyświetlający piętnaście razy znak „@" w tym samym wierszu.
b. Program wyświetlający na ekranie w kolejnych wierszach *n* napisów „Na lekcji informatyki poznajemy zasady programowania" (wartość zmiennej *n* wprowadzaj z klawiatury).
c. Program wprowadzający *k* liczb rzeczywistych z klawiatury i zapamiętujący je kolejno w zmiennej *liczba* (wartości zmiennych *k* i *liczba* wprowadzaj z klawiatury).
d. Program wyświetlający w kolumnie liczby całkowite od 10 do -10.

Zmienną sterującą *i* zadeklaruj jako `int`. Zadeklaruj również inne używane zmienne, dobierając odpowiednio ich typ. Każdy program zapisz w pliku, skompiluj i uruchom.

2. Zapisywanie rozwiązania problemu iteracyjnego w języku C++

Przykład 1. Zapisywanie algorytmu iteracyjnego sumowania *n* liczb w języku C++

Zadanie: Oblicz sumę *n* liczb całkowitych.

Dane: liczba naturalna *n* większa od zera, oznaczająca, ile liczb ma być sumowanych, *n* dowolnych liczb całkowitych.

Wynik: wartość sumy liczb: *suma*.

[*] Suma_n.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int i, a, suma, n;

    cout << "Ile liczb: ";
    cin >> n;
    suma = 0;
    for(i = 0; i < n; i++)
    {
        cout << "Podaj liczbe: ";
        cin >> a;
        suma = suma + a;
    }
    cout << "Suma wynosi: " << suma;

    return 0;
}
```

W języku C++ instrukcję przypisania, w której danej zmiennej przypisujemy jej poprzednią wartość powiększoną o wartość innej zmiennej, np. `suma = suma + a;` można zapisać krócej: `suma += a;`.

W podobny sposób można zapisać instrukcje przypisania z użyciem pozostałych operatorów (tabela 3.).

Przykład zastosowania	Odpowiednik	Znaczenie
`s += a`	`s = s + a`	przypisanie zmiennej *s* poprzedniej wartości tej zmiennej, powiększonej o *a*
`s -= a`	`s = s - a`	przypisanie zmiennej *s* poprzedniej wartości tej zmiennej, pomniejszonej o *a*
`s *= a`	`s = s * a`	przypisanie zmiennej *s* poprzedniej wartości tej zmiennej, pomnożonej *a* razy
`s /= a`	`s = s / a`	przypisanie zmiennej *s* poprzedniej wartości tej zmiennej, podzielonej przez *a*

Tabela 3. Operatory przypisania w języku C++

Ćwiczenie 2. Zapisujemy algorytm sumowania *n* liczb w języku C++

1. Przypomnij listę kroków algorytmu sumowania *n* liczb z tematu 53.
2. Otwórz plik przekazany przez nauczyciela. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Sprawdź, czy program jest zgodny ze specyfikacją zadania i listą kroków. Uruchom i przetestuj program dla kilku różnych wartości zmiennych.

Ćwiczenie 3. Dodajemy do programu sprawdzanie poprawności danych

1. Dodaj do programu z ćwiczenia 2. sprawdzanie poprawności wprowadzanej wartości zmiennej *n*, zgodnie ze specyfikacją zadania (podaną w przykładzie 1.). Sumowanie wprowadzanych liczb powinno być wykonywane dla poprawnej wartości *n*. Jeśli użytkownik wprowadzi błędne dane, powinien wyświetlić się komunikat „Niepoprawne dane".
2. Zapisz program w pliku pod nazwą *Suma_n* i uruchom program dla różnych wartości zmiennej *n*.

Ćwiczenie 4. Stosujemy instrukcję iteracyjną `for`

1. Napisz specyfikację zadania i program obliczający sumę *n* kolejnych liczb naturalnych (począwszy od 1). Sprawdzaj poprawność wprowadzanej wartości zmiennej *n*.
2. Zapisz program w pliku pod nazwą *Naturalne_suma_n*, uruchom i sprawdź jego działanie dla różnych wartości zmiennej *n*.

Wskazówka: Możesz zmodyfikować kod programu z ćwiczenia 3.

Warto zapamiętać

- W językach programowania dostępne są instrukcje iteracyjne (instrukcje pętli), z których korzystamy, aby zapisać algorytm iteracyjny w postaci programu.
- Jeśli instrukcje wewnątrz pętli mają zostać wykonane zadaną liczbę razy, stosujemy pętlę **`for`**.
- W języku C++ zmienna sterująca określa liczbę kroków iteracji (powtórzeń).

Pytania i polecenia

1. Wyjaśnij na przykładach stosowanie instrukcji iteracyjnej **`for`** w języku C++.
2. Do czego służy zmienna sterująca w instrukcji **`for`** w języku C++?
3. Omów zastosowanie iteracji na przykładzie sumowania ośmiu liczb.
4. Na podstawie następujących fragmentów programów napisz, jakie są wyniki ich działania:

   a.

```
for(i = 0; i <= 9; i++)
    cout << "$";
```

   b.

```
for(i = 0; i <= 9; i++)
    cout << "$" << endl;
```

5. Ile razy będzie wykona instrukcja `cout << i;`? Dla jakich wartości zmiennej *i* zostanie wykonana instrukcja `cout << i;`?

```
for(i = 0; i <= 20; i++)
    cout << i;
```

Zadania

1. Zmodyfikuj program *Przepis* zapisany w ćwiczeniu 4. z tematu 56. tak, aby obliczenia wykonywane były dla każdego z *n* składników sałatki, gdzie *n* jest wprowadzane z klawiatury. Zapisz plik pod tą samą nazwą.
2. Napisz program zliczający, ile jest liczb parzystych wśród dziesięciu liczb wprowadzanych z klawiatury i na koniec wyprowadzający ich liczbę. Zapisz program w pliku pod nazwą *Parzyste*.
3. Napisz specyfikację zadania, utwórz listę kroków oraz napisz program obliczania sumy liczb podzielnych przez 7 dla *n* liczb wprowadzanych z klawiatury. Zapisz program w pliku pod nazwą *Podzielne_7_suma*.
4. Napisz program realizujący algorytm obliczania iloczynu *n* dowolnych liczb całkowitych. Wynik mnożenia zapamiętuj w zmiennej *iloczyn* i wyprowadź na ekran. Zapisz program w pliku pod nazwą *Iloczyn_n*.
   Wskazówka: Zmiennej *iloczyn* na początku programu przypisz wartość 1.

5. Napisz specyfikację zadania i program obliczający osobno sumę liczb dodatnich i sumę liczb ujemnych dla *n* liczb całkowitych wprowadzanych z klawiatury. Zapisz program w pliku pod nazwą *Dodatnie_ i_ ujemne*.
6. W firmie X miesięczna płaca podstawowa jest zwiększana m.in. o kwotę za przepracowane nadgodziny. Jeśli liczba nadgodzin przekroczy 30, to stawka za kolejne nadgodziny jest zwiększana o 50%. Wprowadzaj liczbę nadgodzin przepracowanych przez jednego pracownika oraz stawkę za jedną nadgodzinę. Oblicz i wyprowadź płacę za przepracowane nadgodziny dla *n* pracowników. Program napisz na podstawie specyfikacji podanej we wskazówce. Zapisz program w pliku pod nazwą *Place*.
   Wskazówka:
   Dane: liczba całkowita *n* – liczba pracowników, liczba całkowita nieujemna *lng* – liczba nadgodzin, liczba rzeczywista dodatnia *s* – stawka za jedną nadgodzinę.
   Wynik: wartość płacy za nadgodziny: *placa*.

Dla zainteresowanych

7. Napisz program obliczający sumę: 1 + 11 + 111 + 1111 + … + 1…1 (*n* składników), gdzie *n* wprowadzane jest z klawiatury. Zapisz program w pliku pod nazwą *Jedynki*.

Temat 59. Instrukcje iteracyjne zagnieżdżone w języku C++

1. Zagnieżdżone instrukcje iteracyjne `for`
2. Stosowanie zagnieżdżonej instrukcji iteracyjnej `for`

Warto powtórzyć

1. Czym jest iteracja?
2. Jak działa instrukcja iteracyjna **`for`** w języku C++?
3. Co oznacza wyrażenie `i++`?

1. Zagnieżdżone instrukcje iteracyjne `for`

Chcemy napisać program, który umożliwi wyświetlenie na ekranie monitora „prostokąta" utworzonego ze znaków „x" o wymiarach określonych z góry oraz wypełnionego tymi znakami. W jaki sposób zapisać ten program w języku C++?

W problemie mamy do czynienia z powtarzaniem poleceń wyświetlania określonej z góry liczby znaków „x" w określonej z góry liczbie wierszy. Występuje tu pętla w pętli. Do zapisania tego problemu w języku C++ możemy zastosować **zagnieżdżone** instrukcje iteracyjne **`for`**.

Instrukcje iteracyjne **`for`** mogą być zagnieżdżone, czyli instrukcją powtarzaną w pętli może być kolejna instrukcja pętli. Liczba kroków tej iteracji jest określona przez iloczyn: $n \cdot m$, gdzie n i m to liczby powtórzeń odpowiednio w każdej pętli.

```
xxxxxxxxxx
xxxxxxxxxx
xxxxxxxxxx
xxxxxxxxxx
```

Rys. 1. Układ znaków „x" w wyświetlonym na ekranie prostokącie dla $n = 4$ i $m = 10$

Przykład 1. Pętla w pętli w języku C++

Zadanie: Zapisz w języku C++ algorytm, który umożliwi wyświetlenie na ekranie monitora prostokąta utworzonego ze znaków „x” o bokach *n* i *m*, gdzie *m* – liczba znaków „x” w wierszu, *n* – liczba wierszy. Wnętrze prostokąta ma być wypełnione znakami „x”.

Uwaga: Liczba znaków w wierszu określa długość jednego boku prostokąta, a liczba wierszy określa długość drugiego boku.

Prostokat.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int i, j, m, n;
    cout << "Podaj liczbe znakow w wierszu: ";
    cin >> m;
    cout << "Podaj liczbe wierszy: ";
    cin >> n;
    for(i = 0; i < n; i++ )
    {
        for(j = 0; j < m; j++)
            cout << "x";
        cout << endl;
    }
    return 0;
}
```

Ćwiczenie 1. Stosujemy zagnieżdżone instrukcje iteracyjne `for`

1. Napisz specyfikację zadania z przykładu 1.
2. Przepisz program podany w przykładzie 1. Sprawdź, czy program jest zgodny z napisaną specyfikacją zadania. Zapisz program w pliku pod nazwą *Prostokat*. Uruchom i przetestuj program dla kilku różnych wartości zmiennych *m* i *n*.
3. Objaśnij, co się dzieje w poszczególnych wierszach programu.

Wskazówka: Aby objaśnić działanie pętli zagnieżdżonych, możesz prześledzić zmianę wartości zmiennych sterujących pętli zewnętrznej i wewnętrznej:

- $i = 0$ uruchamia się pętla zewnętrzna:
  - $j = 0$ uruchamia się pętla wewnętrzna,
  - dopóki $j < m$ program wyświetla znaki „x”,
  - gdy $j = m$, czyli program wyświetlił *m* znaków „x” w jednym wierszu, następuje wyjście z pętli wewnętrznej, kursor przechodzi do nowego wiersza (instrukcja `cout << endl;`).
- $i = 1$ uruchamia się pętla zewnętrzna:
  - $j = 0$ uruchamia się pętla wewnętrzna itd.

Można tak rozpisywać działanie programu, aż dojdzie się do sytuacji, w której pętla zewnętrzna zostanie zakończona (czyli spełniony będzie warunek $i = n$), po czym nastąpi zakończenie programu.

W języku C++ należy stosować odrębne nazwy zmiennych sterujących określających liczbę iteracji dla pętli zewnętrznej i wewnętrznej.

Ćwiczenie 2. Sprawdzamy stosowanie takich samych zmiennych określających liczbę iteracji

Sprawdź, co się stanie w programie z ćwiczenia 1., jeśli zamiast zmiennej *j* w wewnętrznej pętli użyjesz również zmiennej *i*. Zmodyfikuj program i go uruchom. Co zauważasz?

2. Stosowanie zagnieżdżonej instrukcji iteracyjnej `for`

Ćwiczenie 3. Wyświetlamy na ekranie choinkę z gwiazdek

1. Napisz program, który umożliwi wyprowadzenie na ekran monitora choinki (rys. 2.) składającej się z gwiazdek „*". Liczba gwiazdek tworzących podstawę i wysokość choinki jest taka sama – wprowadzamy ją z klawiatury.
2. Zapisz program w pliku pod nazwą *Choinka*.

Wskazówka: Pamiętaj, że w pętli `for` można zamiast operatora inkrementacji `i++` użyć operatora dekrementacji `i--`.



Rys. 2. Wynik działania programu dla $n = 9$ – ćwiczenie 3.



Rys. 3. Wynik działania programu dla $n = 6$ – ćwiczenie 4.



Ćwiczenie 4. Wyświetlamy na ekranie figurę z gwiazdek

1. Zmodyfikuj program zapisany w ćwiczeniu 3., aby wyświetlił figurę podobną do pokazanej na rysunku 3.
2. Zapisz program w pliku pod nazwą *Figura*.

Warto zapamiętać

- W programach komputerowych można stosować zagnieżdżone instrukcje iteracyjne.
- W przypadku dwóch zagnieżdżonych instrukcji `for` liczba kroków iteracji to iloczyn liczby iteracji zewnętrznej pętli i liczby iteracji wewnętrznej pętli.
- W języku C++ należy stosować odrębne nazwy zmiennych sterujących określających liczbę iteracji dla pętli zewnętrznej i wewnętrznej.

Pytania i polecenia

1. Jaką wartość zwyczajowo przyjmuje się jako początkową wartość zmiennej sterującej w pętli `for`?.
2. Przedstaw na przykładzie, na czym polega zagnieżdżanie się instrukcji pętli.
3. Jak określa się liczbę kroków iteracji w zagnieżdżonych instrukcjach pętli?

Zadania

1. Napisz program, który umożliwi wyprowadzenie na ekran monitora prostokąta o bokach *n* i *m*, składającego się z cyfr 1 i 0, występujących w co drugim wierszu, gdzie *m* – liczba cyfr 1 lub 0 w wierszu, *n* – liczba wierszy. Zapisz program w pliku pod nazwą *Zero_jeden*.

```
1111111111
0000000000
1111111111
0000000000
1111111111
```

Rys. 4. Wynik działania programu dla $m = 10$ i $n = 5$ – zadanie 1.

2. Napisz program tworzący na ekranie obraz składający się z gwiazdek, innych znaków lub cyfr – według własnego pomysłu.

Dla zainteresowanych

3. Napisz program realizujący algorytm, który umożliwi wyprowadzenie na ekran monitora szachownicy składającej się z zer i jedynek. Zapisz program w pliku pod nazwą *Szachownica*.
4. Napisz program wypisujący na ekranie tabliczkę mnożenia liczb naturalnych od 1 do 10. Zapisz program w pliku pod nazwą *Tabliczka*.

Temat 60. Stosowanie instrukcji iteracyjnych `for`, `while` i `do ... while` w języku C++

1. Stosowanie instrukcji iteracyjnej `for` w języku C++
2. Stosowanie instrukcji `while` w języku C++
3. Stosowanie instrukcji `do ... while` w języku C++

Warto powtórzyć

1. Jakie instrukcje wykorzystujemy do zapisania powtarzających się poleceń w językach programowania?
2. Kiedy możemy zastosować instrukcję **`for`**? Podaj przykład problemu iteracyjnego, do rozwiązania którego można zastosować tę instrukcję.
3. Jak określamy liczbę kroków iteracji w instrukcji **`for`** w języku C++ ?
4. W jaki sposób oznaczamy blok instrukcji w języku C++?

1. Stosowanie instrukcji iteracyjnej `for` w języku C++

Zbierasz pieniądze na zakup nowego komputera. Wpłacasz co miesiąc dowolną kwotę na swoje konto. Chcesz obliczyć sumę pieniędzy wpłaconych w ciągu roku. Jak zapisać rozwiązanie tego zadania w wybranym języku programowania?

W przedstawionym problemie powtarzają się dwie czynności: wprowadzania wpłacanej kwoty i powiększania sumy pieniędzy o tę kwotę. Mamy do czynienia z algorytmem iteracyjnym, w którym liczba powtórzeń (iteracji) jest z góry określona (pieniądze wpłacamy przez rok co miesiąc, czyli dwanaście razy).



Ćwiczenie 1. Obliczamy sumę dwunastu liczb wprowadzanych z klawiatury

1. Napisz specyfikację zadania i program, który obliczy sumę dwunastu comiesięcznych wpłat na zakup komputera i wyświetli wynik na ekranie.
2. Zapisz program w pliku pod nazwą *Suma_12*.

2. Stosowanie instrukcji `while` w języku C++

Chcesz dalej oszczędzać na zakup nowego komputera, ponieważ potrzebna jest większa kwota. Zamierzasz wpłacać pieniądze, dopóki suma wpłat nie przekroczy potrzebnej kwoty. Jak przedstawić ten problem w wybranym języku programowania?

Nie wiemy z góry, ilu wpłat musimy dokonać, aby osiągnąć lub przekroczyć kwotę potrzebną na zakup komputera. Liczba iteracji zależy w tym przypadku od spełnienia warunku przekroczenia założonej kwoty.

Do zapisywania algorytmów iteracyjnych, w których liczba iteracji nie jest z góry określona, możemy zastosować instrukcje iteracyjne:

- **`while`**,
- **`do … while`**.

Instrukcja iteracyjna `while`	**`while`**(*warunek*) *lista_instrukcji;* *kolejna_instrukcja;*

Działanie pętli **`while`**: najpierw sprawdzany jest *warunek*; jeśli jest spełniony, to wykonywana jest *lista_instrukcji*. Wewnątrz bloku *lista_instrukcji* powinna być zawsze umieszczona instrukcja, która zmienia wartość *warunku* – w przeciwnym wypadku pętla nigdy się nie zakończy. W szczególnej sytuacji, gdy *warunek* od razu nie jest spełniony, *lista_instrukcji* w ogóle nie zostanie wykonana. *Lista_instrukcji* może zawierać jedną lub wiele instrukcji.

W realizacji algorytmów iteracyjnych ważne jest prawidłowe określenie sposobu zakończenia działań. Nieprawidłowe określenie tych warunków może spowodować, że obliczenia nigdy się nie zakończą, czyli nastąpi **zapętlenie programu**.

Przykład 1. Stosowanie instrukcji **`while`** w języku C++

Zadanie: Napisz program obliczający sumę wpłat aż do osiągnięcia lub przekroczenia założonej kwoty.

Dane: liczba rzeczywista dodatnia *kwota*, oznaczająca kwotę potrzebną na zakup komputera, ciąg dowolnych liczb rzeczywistych dodatnich *wplata*, oznaczających kolejne comiesięczne wpłaty.

Wynik: liczba rzeczywista dodatnia *suma_wplat*, oznaczająca wartość sumy wszystkich wpłat.

Zbiorka.cpp

```cpp
#include <iostream>
using namespace std;

int main()
{
    float kwota, wplata, suma_wplat;

    cout << "Podaj potrzebna kwota: ";
    cin >> kwota;
    suma_wplat = 0;
    while(suma_wplat < kwota)
    {
        cout << "Wprowadz wplate: ";
        cin >> wplata;
        suma_wplat = suma_wplat + wplata;
    }
    cout << "Suma wplat wynosi: " << suma_wplat << " zl";

    return 0;
}
```

Ćwiczenie 2. Analizujemy, testujemy i modyfikujemy gotowy program

1. Przepisz program podany w przykładzie 1. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Sprawdź, czy program jest zgodny ze specyfikacją zadania. Uruchom i przetestuj program dla kilku różnych wartości zmiennych. Objaśnij działanie instrukcji **`while`**, m.in. odpowiedz na pytanie: *Kiedy zostanie wykonana instrukcja* `cout` *z 17. wiersza?*
2. Zmodyfikuj program, aby na ekranie wyświetlała się również bieżąca suma, uzyskana po każdej wpłacie, a na koniec – różnica pomiędzy kwotą wpłaconą a założoną (nadpłata). Dodaj odpowiednie komunikaty. Zapisz plik pod nazwą *Zbiorka*.

Ćwiczenie 3. Sprawdzamy poprawność danych

1. Dodaj do programu zapisanego w ćwiczeniu 2. sprawdzanie poprawności wprowadzanych wartości zmiennych *kwota* i *wplata*, zgodnie ze specyfikacją zadania (podaną w przykładzie 1.). Wprowadzanie i sumowanie liczb powinny być wykonywane dla poprawnych wartości zmiennych *kwota* i *wplata*:
   a. jeśli użytkownik wprowadzi niepoprawną wartość zmiennej *kwota*, wyświetl na ekranie komunikat „Wprowadzono bledna liczbe” i zakończ program,
   b. jeśli użytkownik wprowadzi niepoprawną wartość zmiennej *wplata*, wyświetl na ekranie komunikat „Wprowadzono bledna liczbe” i zignoruj wprowadzoną wartość.
2. Zapisz plik pod tą samą nazwą. Uruchom program i sprawdź jego działanie dla różnych wartości zmiennych.

Ćwiczenie 4. Stosujemy instrukcję iteracyjną `while`

1. Napisz specyfikację zadania i program, który będzie obliczał sumę liczb całkowitych wprowadzanych z klawiatury aż do wprowadzenia zera. Wprowadzenie zera kończy zliczanie. Wynik sumowania wyprowadź na ekran.
2. Zapisz program w pliku pod nazwą *Suma_while*.

Wskazówka: Na początku programu przypisz zmiennej *liczba* wartość różną od zera, np. 1, a zmiennej *suma* – 0. Początkową wartość można przypisać zmiennej podczas jej deklaracji: `int liczba = 1, suma = 0;`

3. Stosowanie instrukcji `do ... while` w języku C++

Drugą instrukcją iteracyjną, którą możemy zastosować w przypadku, gdy nie znamy liczby iteracji jest instrukcja: **`do ... while`**.

Instrukcja iteracyjna **`do ... while`**	**`do`** *`lista_instrukcji;`* **`while`**`(`*`warunek`*`);` *`kolejna_instrukcja;`*

W instrukcji **`do ... while`** najpierw jest wykonywana *lista_instrukcji*, a dopiero potem jest sprawdzany warunek. Instrukcje w pętli (*lista_instrukcji*) są wykonywane, dopóki warunek jest prawdziwy. Jeśli warunek jest fałszywy – wykonywana jest *kolejna_instrukcja*. Niezależnie od wartości początkowej warunku, *lista_instrukcji* zostanie wykonana przynajmniej raz. Jako *lista_instrukcji* może wystąpić pojedyncza instrukcja (w tym instrukcja **`do ... while`**) lub więcej instrukcji, jednak zawsze ujętych w blok `{}`.

Przykład 2. Stosowanie instrukcji `do ... while` w języku C++

Zadanie: Napisz program zliczający znaki wprowadzane z klawiatury aż do wprowadzenia kropki. Wprowadzenie kropki kończy zliczanie. Wyprowadź na ekran odpowiedni komunikat i liczbę wprowadzonych znaków łącznie z kropką.

Dane: ciąg znaków wprowadzanych z klawiatury, zapisywanych kolejno w zmiennej *znak*.

Wynik: liczba całkowita *liczba_znakow*, oznaczająca liczbę znaków wprowadzonych z klawiatury.

Zliczanie_znakow.cpp

```
#include <iostream>
using namespace std;

int main()
{
    char znak;
    int liczba_znakow = 0;

    do
    {
        cout << "Wprowadz znak: ";
        cin >> znak;
        liczba_znakow++;
    }
    while(znak != '.');
    cout << "Liczba wprowadzonych znakow wynosi: " << liczba_znakow;
    return 0;
}
```

Uwagi:

- Pojedynczy znak umieszczamy w programie w pojedynczych górnych cudzysłowach.
- Instrukcja `liczba_znakow++;`
  to odpowiednik instrukcji: `liczba_znakow = liczba_znakow + 1;`

Ćwiczenie 5. Analizujemy, testujemy i modyfikujemy gotowy program

1. Przepisz program podany w przykładzie 2. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Sprawdź, czy program jest zgodny ze specyfikacją zadania. Uruchom i przetestuj program dla kilku różnych wartości zmiennych. Objaśnij działanie instrukcji **`do ... while`**, m.in. odpowiedz na pytanie: *Kiedy zostanie wykonana instrukcja* `cout` *z 16. wiersza?*
2. Zmodyfikuj program, aby nie zliczał kropki. Zapisz program w pliku pod nazwą *Zliczanie_znakow_bez_kropki*.

Ćwiczenie 6. Stosujemy instrukcję iteracyjną `do ... while` w języku C++

1. Napisz specyfikację zadania i program, który będzie obliczał różnicę dwóch liczb całkowitych (*odjemna*, *odjemnik*) wprowadzanych z klawiatury, dopóki wynik odejmowania (*roznica*) będzie dodatni. Wynik niedodatni kończy obliczenia. Zliczaj wykonane działania odejmowania, których wynik jest dodatni, i wyprowadź ich liczbę na ekran.
2. Zapisz program w pliku pod nazwą *Roznice*.

Wskazówka: W warunku logicznym instrukcji **`do ... while`** sprawdzaj, czy wynik odejmowania (*roznica*) jest dodatni.

Warto zapamiętać

- W językach programowania występują instrukcje iteracyjne, które umożliwiają zapis algorytmów iteracyjnych, np.: w języku C++ instrukcje **`for`**, **`while`** i **`do ... while`**. Zwykle gdy znamy liczbę iteracji, stosujemy instrukcję **`for`**; jeśli nie wiemy, ile będzie powtórzeń – instrukcje **`while`** lub **`do ... while`**.
- W instrukcji **`while`** najpierw jest sprawdzany warunek logiczny, a potem (gdy warunek jest prawdziwy) są wykonywane instrukcje znajdujące się wewnątrz instrukcji **`while`**. Gdy warunek od razu nie jest spełniony, instrukcje wewnątrz instrukcji **`while`** nie zostaną wykonane ani razu.
- Instrukcja **`do ... while`** różni się od **`while`** tym, że warunek jest sprawdzany po bloku instrukcji umieszczonym pomiędzy słowami kluczowymi **`do`** i **`while`**, czyli te instrukcje zostaną wykonane przynajmniej raz.
- Zapętlenie programu nastąpi wtedy, gdy nie określimy prawidłowo warunku zakończenia algorytmu.

Pytania i polecenia

1. Jakie instrukcje w językach programowania umożliwiają zapis algorytmów iteracyjnych?
2. Wyjaśnij na przykładzie wybranego języka programowania, jak działa instrukcja **`while`**.
3. Czy zostaną wykonane instrukcje wewnątrz instrukcji **`while`**, jeśli warunek jest prawdziwy? Uzasadnij odpowiedź.
4. Czym różni się działanie instrukcji **`while`** od **`do ... while`**?

Zadania

1. Znajdź w Internecie wartość średniej temperatury w Polsce w poszczególnych miesiącach – w dzień i w nocy. Napisz program obliczający na podstawie tych danych średnie roczne temperatury w dzień i w nocy. Wyniki wyświetl na ekranie. Zapisz program w pliku pod nazwą *Temperatury*.
2. Napisz program obliczający i wyświetlający na ekranie iloraz dwóch liczb całkowitych (*dzielna*, *dzielnik*) wprowadzanych z klawiatury aż do wprowadzenia dzielnika równego zero. Ponieważ nie da się obliczyć ilorazu dla dzielnika równego zero, program powinien wyświetlić odpowiedni komunikat. Zapisz program w pliku pod nazwą *Iloraz*.
3. Napisz program obliczający i wyświetlający na ekranie objętość sześcianu o boku *bok* aż do wprowadzenia niedodatniej wartości zmiennej *bok*. Zapisz program w pliku pod nazwą *Objetosc_szescianu*.
4. Napisz specyfikację zadania i program zliczający, ile jest liczb podzielnych przez 3 pośród liczb całkowitych wprowadzanych z klawiatury. W przypadku wprowadzenia zera program ma zakończyć działanie. Zapisz program w pliku pod nazwą *Podzielne_przez_3*.

Dla zainteresowanych

5. Masz *n* zł na kupienie upominków dla znajomych. Upominki możesz kupować, dopóki starczy ci pieniędzy. Ceny upominków wprowadzaj z klawiatury. Nie możesz przekroczyć kwoty *n* zł ani wydać mniej. Jeśli cena ostatniego upominku przekroczy kwotę *n*, musisz kupić coś tańszego. Program kończy się, gdy wydasz dokładnie *n* zł. Napisz odpowiedni program i zapisz go w pliku pod nazwą *Upominki*.
6. Napisz program obliczający część całkowitą pierwiastka kwadratowego z podanej liczby naturalnej *n*.
Wskazówka: Istnieją różne algorytmy obliczania pierwiastka. Można na przykład obliczać kwadraty kolejnych liczb naturalnych aż do uzyskania liczby większej lub równej *n*.

Temat 61. Funkcje zwracające wartość w języku C++

1. Funkcje w języku C++
2. Ogólna definicja funkcji zwracającej wartość w języku C++
3. Funkcje zwracające wartość z parametrami w języku C++
4. Funkcje zwracające wartość bez parametrów w języku C++

Warto powtórzyć

1. W jakich środowiskach programowania tworzyliśmy programy z wykorzystaniem podprogramów?
2. W jaki sposób definiowaliśmy podprogramy w tych środowiskach? Jak je wywoływaliśmy?
3. Jak zapisujemy algorytm iteracyjny w języku C++?

1. Funkcje w języku C++

Do magazynu sprzętu RTV dowożone są kilkakrotnie w ciągu miesiąca różne artykuły. Jak napisać program zliczający, ile sztuk każdego artykułu dostarczono we wszystkich dostawach łącznie?



Jeśli musimy kilkakrotnie zsumować liczby w różnych zbiorach, możemy opracować cząstkowe rozwiązanie obliczające sumę, a potem wielokrotnie wykorzystać je w programie. Do opracowywania problemów cząstkowych, które są częściami zadania rozwiązywanego przez dany program, służą **podprogramy**.

Podprogram realizujący określone zadanie (np. sumowanie liczb) może być wielokrotnie wykorzystywany – w tym samym programie lub w innych. Stosowanie podprogramów zwiększa przejrzystość programu.

P **Podprogram**
(procedura lub funkcja)

Wyodrębniona część programu, mająca unikatową nazwę i ustalony sposób wymiany danych z innymi częściami programu.

Parametr **P**

Wartość przekazywana do funkcji.

Wartość zwracana **W**

Wartość przekazywana z funkcji do miejsca wywołania (np. programu głównego).

W języku C++ wszystkie podprogramy nazywamy **funkcjami.**

Poznaliśmy kilka wbudowanych funkcji języka C++, m.in. funkcję `main()` stanowiącą główny program i funkcję `sqrt()` zwracającą pierwiastek kwadratowy z nieujemnej liczby rzeczywistej. W języku C++ można definiować własne funkcje.

Funkcje mogą **zwracać wartość** do programu głównego lub nic nie zwracać, mogą mieć określone **parametry** lub ich nie posiadać.

Funkcje dzielimy na: zwracające wartość i niezwracające wartości.

Nazwy funkcji tworzymy na takich samych zasadach jak nazwy zmiennych (można używać tylko liter, znaków podkreślenia i cyfr; nazwa nie może zaczynać się od cyfry; przyjęte jest niestosowanie polskich liter). Funkcjom należy nadawać nazwy opisujące ich działanie. Ważne jest, aby przyjąć jednolity sposób nazewnictwa, np. w nazwach funkcji i zmiennych używa się wyłącznie małych liter, a w nazwach wieloczłonowych stosuje się znak podkreślenia, np. *liczba_dostaw*, *suma_sztuk*, *wprowadz_dane*, *wyprowadz_dane*, *najwiekszy*.

Aby zastosować funkcję, trzeba ją **zdefiniować**, a następnie **wywołać** w programie. Wywołanie funkcji powoduje wykonanie instrukcji będących treścią funkcji i zwrócenie wyniku jej działania. W momencie wywołania funkcji **parametry formalne** (zwane też **parametrami**) są zastępowane przez **parametry aktualne** (zwane też **argumentami**).

2. Ogólna definicja funkcji zwracającej wartość w języku C++

Funkcję zwracającą wartość stosujemy, jeśli celem funkcji jest obliczenie i zwrócenie pewnej wartości do programu. Funkcja musi zawierać instrukcję **`return`** z wartością zwracaną do miejsca wywołania (np. programu głównego).

Ogólna postać definicji funkcji zwracającej wartość

nagłówek funkcji

```
opis_typu nazwa_funkcji(lista_parametrów_formalnych)
{
    lista_instrukcji;
    return wartość;
}
```

treść funkcji

Miejsce definiowania funkcji	Funkcja może zostać zdefiniowana przed jej pierwszym wywołaniem lub, jeżeli zostanie zadeklarowana (poprzez umieszczenie nagłówka funkcji) przed funkcją `main()`, w dowolnym miejscu programu – ale poza funkcją `main()`.
Określanie typu funkcji i parametrów	Należy wskazać typ funkcji (typ zwracanej wartości) i określić typy oraz nazwy parametrów formalnych.
Treść funkcji	Treść (*lista_instrukcji* i instrukcja **`return`**) musi się znaleźć w bloku `{}`.

Tabela 1. Zapis funkcji zwracającej wartość w języku C++

Aby wywołać funkcję zwracającą wartość, należy podać nazwę funkcji, a w nawiasach – wartości parametrów. Wartość zwróconą przez funkcję możemy np. przypisać zmiennej w programie głównym. W przypadku definiowania i wywoływania funkcji bez parametrów nawiasy po nazwie funkcji pozostawiamy puste.

3. Funkcje zwracające wartość z parametrami w języku C++

Sposób definiowania i wywołania funkcji zwracającej wartość pokażemy na przykładzie funkcji z jednym parametrem, obliczającej sumę *n* liczb.

Przykład 1. Definiowanie i wywołanie funkcji zwracającej wartość z jednym parametrem w języku C++

Zadanie: Do pewnego magazynu sprzętu RTV dowożone są kilkakrotnie w ciągu miesiąca telewizory. Utwórz program obliczający i wyprowadzający na ekran łączną liczbę telewizorów ze wszystkich dostaw. Liczbę dostaw oraz wielkość każdej dostawy wprowadzaj z klawiatury po uruchomieniu programu.

Dane:

- liczba naturalna większa od zera *l_dostaw_tv* oznaczająca liczbę dostaw telewizorów,
- *l_dostaw_tv* liczb naturalnych większych od zera wprowadzanych kolejno i zapamiętywanych w zmiennej *a*, oznaczających odpowiednio wielkości kolejnych dostaw telewizorów.

Wyniki: łączna liczba dostarczonych telewizorów: *suma_tv*.

Opis rozwiązania: Rysunek przedstawia trzy dostawy telewizorów (czyli *l_dostaw_tv* = 3), w których dostarczono kolejno: 8, 15 i 20 telewizorów, a więc *suma_tv* = 43.



Na początku programu zdefiniujemy funkcję *suma* z jednym parametrem, obliczającą i zwracającą sumę *n* liczb wprowadzanych z klawiatury. Parametrem przekazywanym do funkcji jest liczba elementów zbioru *n*.

Funkcję *suma()* wywołamy w programie głównym z parametrem aktualnym *l_dostaw_tv*. Możemy to zrobić na dwa sposoby.

Wywołanie funkcji *suma()* – sposób 1.

Wynik działania funkcji (zwracaną wartość) możemy przypisać zmiennej *suma_tv*.

```
suma_tv = suma(l_dostaw_tv);
```

Wywołanie funkcji *suma()* – sposób 2.

Wynik działania funkcji zwracającej wartość możemy również wyprowadzić na ekran bez używania dodatkowych zmiennych (tu: *suma_tv*) i przypisywania im wartości funkcji.

```
cout << "Liczba tv z " << l_dostaw_tv << " dostaw wynosi "
<< suma(l_dostaw_tv);
```

parametr formalny *n*

wartość zwracana

[*] Sumy_dostaw.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int suma(int n)
5  {
6      int a, s = 0;
7
8      for(int i = 0; i < n; i++)
9      {
10         cout << "Podaj liczbe sztuk w " << i+1 << " dostawie: ";
11         cin >> a;
12         s += a;
13     }
14     return s;
15 }
16
17 int main()
18 {
19     int l_dostaw_tv, suma_tv;
20
21     cout << "Ile bylo dostaw tv: ";
22     cin >> l_dostaw_tv;
23     suma_tv = suma(l_dostaw_tv);
24     cout << "Liczba tv z " << l_dostaw_tv << " dostaw wynosi " << suma_tv;
25     return 0;
26 }
27
```

definicja funkcji *suma()*

część wykonawcza programu

wywołanie funkcji *suma()* z parametrem aktualnym *l_dostaw_tv*

Ćwiczenie 1. Definiujemy funkcję zwracającą wartość z jednym parametrem

1. Przepisz program podany w przykładzie 1.
2. Zapisz program w pliku pod nazwą *Sumy_dostaw*.
3. Uruchom i przetestuj program dla kilku różnych wartości zmiennych. Wyjaśnij znaczenie poszczególnych wierszy programu. Wskaż miejsce wywołania funkcji.

Ćwiczenie 2. Wywołujemy funkcję *suma()* z innym parametrem aktualnym

1. Otwórz plik *Sumy_dostaw* zapisany w ćwiczeniu 1. W programie zapisano tylko obliczenia łącznej wielkości dostaw telewizorów. Dodaj podobne instrukcje dla obliczenia łącznej wielkości dostaw głośników.
   a. Zadeklaruj zmienne: *l_dostaw_glosnikow*, oznaczającą liczbę dostaw głośników i zmienną *suma_glosnikow* oznaczającą łączną liczbę dostarczonych głośników.
   b. Funkcję *suma()* wywołaj w programie głównym z parametrem *l_dostaw_glosnikow*.
2. Zapisz plik pod tą samą nazwą.

Wskazówki: Ułatwiaj sobie pracę i kopiuj podobne fragmenty programu, w których wystarczy tylko zmienić nazwy zmiennych.

Ćwiczenie 3. Dodajemy do funkcji i programu sprawdzanie poprawności danych

1. Otwórz plik *Sumy_dostaw* zapisany w ćwiczeniu 2.
2. Zmodyfikuj funkcję i program, aby obliczenia były wykonywane dla poprawnych (czyli większych od zera) wartości wszystkich zmiennych: *a*, *l_dostaw_tv*, *l_dostaw_glosnikow*. Jeśli zostanie wprowadzona niepoprawna wartość, wyświetl na ekranie komunikat „Wprowadzono bledna liczbe” i zignoruj wprowadzoną wartość.
3. Zapisz program w pliku pod nazwą *Sumy_dostaw_popr*.
4. Uruchom i przetestuj program dla kilku różnych wartości zmiennych.

Ćwiczenie 4. Modyfikujemy program

1. Otwórz plik *Sumy_dostaw_popr* zapisany w ćwiczeniu 3.
2. Zmniejsz liczbę zmiennych, czyli do wyprowadzenia wyników funkcji użyj sposobu 2. z przykładu 1.
3. Do programu dodaj zliczanie dwóch innych artykułów przywożonych do magazynu: smartfonów i komputerów.
4. Zapisz program w pliku pod nazwą *Sumy_dostaw_zmod.*

4. Funkcje zwracające wartość bez parametrów w języku C++

Funkcje można definiować bez parametrów, ale wówczas, aby przekazać do nich dane, musimy zastosować tzw. **zmienne globalne** (widoczne w dowolnym miejscu programu) – w funkcjach i w programie głównym.

Niezależnie od zmiennych globalnych można w funkcji stosować **zmienne lokalne**. W języku C++ zmienne lokalne są widoczne od miejsca deklaracji do końca funkcji (do nawiasu klamrowego zamykającego definicję funkcji). W funkcji *suma()* (przykład 1.) zmienne *i*, *a* oraz *s* są lokalne. Zakres zmiennej *i* jest ograniczony tylko do pętli `for`.

Przykład 2. Stosowanie funkcji zwracającej wartość bez parametrów w języku C++

Zadanie: Zdefiniuj funkcję *obwod()* bez parametrów, zwracającą do programu głównego obwód kwadratu o boku *bok*. Wywołaj funkcję w programie głównym i wyprowadź wartość obwodu. Wartość zmiennej *bok* wprowadzaj z klawiatury.

Dane: liczba naturalna *bok* oznaczająca długość boku kwadratu.

Wynik: wartość obwodu kwadratu.

Opis rozwiązania: W programie zastosujemy zmienną globalną *bok*, którą musimy zadeklarować przed definicją funkcji *obwod()*.

Obwod_kwadratu.cpp

```
1  #include <iostream>
2  using namespace std;
3  int bok;
4
5  int obwod()
6  {
7      return 4*bok;
8  }
9
10 int main()
11 {
12     cout << "Podaj bok kwadratu: ";
13     cin >> bok;
14     cout << "Obwod kwadratu o boku " << bok << " wynosi " << obwod();
15     return 0;
16 }
```

Ćwiczenie 5. Definiujemy funkcję zwracającą wartość bez parametrów

1. Przepisz program podany w przykładzie 2. Zapisz program w pliku pod nazwą *Obwod_kwadratu*. Uruchom i przetestuj program dla kilku różnych wartości zmiennej. Wyjaśnij znaczenie poszczególnych wierszy programu. Wskaż miejsce wywołania funkcji.
2. Dodaj do programu funkcję *pole()* bez parametru, obliczającą pole kwadratu o boku *bok*.
3. Zapisz program w pliku pod nazwą *Obwod_i_pole_kwadratu*. Uruchom i przetestuj program dla kilku różnych wartości zmiennej.

Dla zwiększenia przejrzystości i uniwersalności programu należy unikać wykorzystywania zmiennych globalnych, a stosować funkcje z parametrami.

```
4  int obwod(int bok)
5  {
6      return 4*bok;
7  }
8  int pole(int bok)
9  {
10     return bok*bok;
11 }
```

Rys. 1. Definicje funkcji z parametrem – ćwiczenie 6.

Ćwiczenie 6. Definiujemy funkcję zwracającą wartość z parametrem

1. Otwórz plik *Obwod_i_pole_kwadratu* zapisany w ćwiczeniu 5. Zmodyfikuj program, rezygnując ze zmiennej globalnej i zmieniając definicje funkcji *suma()* i *obwod()* na funkcje z parametrem. W programie głównym wywołaj funkcje z parametrem aktualnym *x*, wprowadzanym z klawiatury jako liczba całkowita.
2. Zapisz program w pliku pod nazwą *Obwod_i_pole_kwadratu_z_par*.

Wskazówka: W obydwu funkcjach można użyć tej samej nazwy parametru. W trzecim wierszu programu (przykład 2.) zrezygnuj z deklaracji `int bok;`

Warto zapamiętać

- Podprogramy (funkcje) stosujemy, aby opracowywać oddzielnie problemy cząstkowe i w ten sposób pisać przejrzyste programy.
- Funkcje w języku C++ mogą zwracać wartości lub nie. Każdy rodzaj funkcji możemy definiować z parametrami lub bez.
- Funkcję wywołujemy w programie głównym, podając jej nazwę i parametry aktualne. W przypadku braku parametrów, po nazwie funkcji umieszczamy puste nawiasy. W przypadku funkcji zwracającej wartość, wartość zwracaną możemy np. przypisać zmiennej w programie głównym.
- Program główny komunikuje się z funkcją przez przekazywanie parametrów – parametry aktualne są wprowadzane w miejsce parametrów formalnych w momencie wywołania funkcji. Gdy wywołanie funkcji się zakończy, zwraca wartość jako swój wynik.

Pytania i polecenia

1. Dlaczego definiujemy podprogramy?
2. Na czym polega wywołanie funkcji? Czym jest parametr formalny, a czym parametr aktualny?
3. Kiedy definiujemy funkcję zwracającą wartość?
4. Czym jest wartość zwracana?
5. Omów ogólną postać definicji funkcji zwracającej wartość.
6. W jaki sposób wywołujemy funkcję zwracającą wartość?

Zadania

1. Do programu *Sumy_dostaw_zmod* zapisanego w ćwiczeniu 4. dodaj jeszcze dwa inne artykuły, które są dostarczane do magazynu i wykonaj te same obliczenia co dla telewizorów i innych artykułów. Zapisz program w pliku *Sumy_dostaw_6*.
2. Napisz program umożliwiający obliczenie objętości graniastosłupa prawidłowego czworokątnego. Zdefiniuj funkcję *objetosc()* z dwoma parametrami (*a*, *h*), obliczającą objętość graniastosłupa prawidłowego czworokątnego o krawędzi podstawy *a* i wysokości *h* i zwracającą do programu głównego wynik obliczeń. Wywołaj funkcję w programie głównym z parametrami aktualnymi *bok* i *wysokosc* wprowadzanymi z klawiatury jako liczby całkowite. Wyprowadź na ekran wynik – objętość graniastosłupa prawidłowego czworokątnego o krawędzi podstawy *bok* i wysokości *wysokosc*. Zapisz program w pliku pod nazwą *Objetosc_graniastoslupa*.
3. Do programu *Objetosc_graniastoslupa* zapisanego w zadaniu 2. dodaj sprawdzanie poprawności wprowadzanych danych. Jeśli użytkownik wprowadzi niepoprawną (ujemną lub równą zero) wartość zmiennej *bok* lub *wysokosc*, wyświetl na ekranie komunikat „Wprowadzono bledna liczbe". Zapisz program w pliku pod nazwą *Objetosc_graniastoslupa_popr*.

Dla zainteresowanych

4. Wymyśl samodzielnie problem (podobnie jak w przykładzie 1.), w którym możesz wykorzystać funkcję *suma()*. Sformułuj zadanie i jego specyfikację. Napisz program realizujący to zadanie.
5. Utwórz program zliczający, ile liczb parzystych wprowadzono, i wyświetlający wynik na ekranie. Zliczanie wykonuj, dopóki nie zostanie wprowadzone zero. Zdefiniuj funkcję logiczną (typu **bool**) *czy_parzysta()* z parametrem całkowitym *a*, sprawdzającą, czy liczba jest parzysta. Wywołaj funkcję w programie głównym. Zapisz program w pliku pod nazwą *Zlicz_parzyste*.

Wskazówka: Funkcja logiczna *czy_parzysta()* zwraca wartość logiczną PRAWDA lub FAŁSZ (w języku C++: `true / false`). W języku C++ wartość różną od zera uważa się za logiczną prawdę, a wartość równą 0 – za logiczny fałsz.
Postać funkcji może być następująca:

```
bool czy_parzysta(int a)
{
    return a % 2 == 0;
}
```

Rys. 2. Definicja funkcji – zadanie 5.

Temat 62.

Funkcje niezwracające wartości w języku C++

1. Ogólna definicja funkcji niezwracającej wartości w języku C++
2. Funkcje niezwracające wartości bez parametrów w języku C++
3. Funkcje niezwracające wartości z parametrami w języku C++

Warto powtórzyć

1. Kiedy stosujemy funkcję zwracającą wartość?
2. Czym jest parametr formalny, a czym aktualny?
3. Na czym polega wywołanie funkcji?

1. Ogólna definicja funkcji niezwracającej wartości w języku C++

Funkcje niezwracające wartości stosujemy, gdy mamy wykonać pewne czynności, np. wprowadzić dane, wyprowadzić dane lub obliczyć wiele wartości.

Ogólna postać definicji funkcji niezwracającej wartości	**void** *nazwa_funkcji*(*lista_parametrów_formalnych*) { *lista_instrukcji*; }
Miejsce definiowania funkcji	Funkcja może zostać zdefiniowana przed jej pierwszym wywołaniem lub, jeżeli zostanie zadeklarowana (poprzez umieszczenie nagłówka funkcji) przed funkcją `main()`, w dowolnym miejscu programu – ale poza funkcją `main()`.
Określanie typu funkcji i parametrów	Funkcja taka nie ma typu, dlatego w części nagłówkowej przed nazwą funkcji należy umieścić słowo kluczowe **`void`**. W przypadku większej liczby parametrów oddzielamy je przecinkami.
Treść funkcji	Treść (*lista_instrukcji*) musi się znaleźć w bloku `{}`. Funkcja nie zwraca wartości, dlatego instrukcja **`return`** nie musi wystąpić (a jeśli występuje, to nie może określać wartości).

Tabela 1. Zapis funkcji niezwracającej wartości w języku C++

Aby wywołać funkcję niezwracającą wartości, należy umieścić nazwę funkcji z parametrami aktualnymi w odpowiednim miejscu programu głównego. W przypadku braku parametrów nawiasy po nazwie funkcji pozostawiamy puste.

Wywołanie funkcji niezwracającej wartości	*nazwa_funkcji(lista_parametrów_aktualnych);* lub *nazwa_funkcji();*

2. Funkcje niezwracające wartości bez parametrów w języku C++

Przykład 1. Stosowanie funkcji niezwracającej wartości bez parametrów w języku C++

Zadanie: Napisz program umożliwiający wyprowadzenie na ekran monitora napisu „Poznajemy funkcje", a pod nim w jednym wierszu dwudziestu gwiazdek (znaków *). Zdefiniuj funkcję niezwracającą wartości o nazwie *gwiazdki()* bez parametrów. Zadaniem funkcji jest wyprowadzenie w jednym wierszu dwudziestu gwiazdek i przeniesienie kursora do nowego wiersza. Funkcję wywołaj w programie głównym.

Opis rozwiązania:

Celem funkcji jest wyświetlenie z góry określonej liczby gwiazdek. Żadna wartość nie jest przekazywana z programu głównego, więc funkcja nie musi mieć parametrów.

Jedyną zmienną w programie jest zmienna *i*, użyta w funkcji jako zmienna lokalna.

Gwiazdki.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  void gwiazdki()
5  {
6      for(int i=0; i<20; i++)
7          cout << "*";
8      cout << endl;
9  }
10
11 int main()
12 {
13     cout << "Poznajemy funkcje" << endl;
14     gwiazdki();
15     return 0;
16 }
```

definicja funkcji

wywołanie funkcji

Ćwiczenie 1. Definiujemy funkcję niezwracającą wartości bez parametrów w języku C++

1. Przepisz program podany w przykładzie 1.
2. Zapisz program w pliku pod nazwą *Gwiazdki*. Uruchom i przetestuj program. Wyjaśnij znaczenie poszczególnych wierszy programu. Wskaż miejsce wywołania funkcji.

Ćwiczenie 2. Modyfikujemy program

1. Dodaj do programu zapisanego w ćwiczeniu 1. wyświetlenie w kolejnych wierszach tekstów: „w języku C++", „na lekcji informatyki". Pod każdym napisem wyświetl gwiazdki. Ile razy wywołasz funkcję *gwiazdki()*?
2. Zapisz program w pliku pod nazwą *Gwiazdki_z_napisami*.

3. Funkcje niezwracające wartości z parametrami w języku C++

Przykład 2. Stosowanie funkcji niezwracającej wartości z parametrami w języku C++

Chcemy, aby liczba wyświetlanych gwiazdek była wprowadzana z klawiatury i przekazywana do funkcji jako parametr.

Do definicji funkcji dodamy parametr *n*, który w momencie wywołania funkcji będzie zastąpiony parametrem aktualnym *l_gwiazdek* (liczbą całkowitą większą od zera wprowadzaną z klawiatury).

```
4  void gwiazdki(int n)
5  {
6      for(int i=0; i<n; i++)
7          cout << "*";
8      cout << endl;
9  }
```

Wywołanie funkcji w programie głównym: `gwiazdki(l_gwiazdek);`

Ćwiczenie 3. Definiujemy funkcję niezwracającą wartości z parametrami w języku C++

1. Otwórz plik *Gwiazdki* zapisany w ćwiczeniu 1. Zmień definicję funkcji *gwiazdki()* na funkcję z parametrem *n* i wywołaj w programie głównym z parametrem aktualnym, będącym liczbą całkowitą większą od zera (*l_gwiazdek*). Wartość parametru wprowadzaj z klawiatury.
2. Zapisz program w pliku pod nazwą *Gwiazdki_par*.
3. Uruchom i przetestuj program dla różnych wartości zmiennej *l_gwiazdek*.

Ćwiczenie 4. Modyfikujemy program

1. Otwórz plik *Gwiazdki_z_napisami* zapisany w ćwiczeniu 2. Zmień definicję funkcji *gwiazdki* na funkcję z parametrem *n* i wywołaj w programie głównym z parametrem aktualnym będącym liczbą całkowitą (*l_gwiazdek*). Wartość parametru wprowadzaj z klawiatury.
2. Zapisz program w pliku pod nazwą *Gwiazdki_z_napisami_par*.
3. Uruchom i przetestuj program dla różnych wartości zmiennej *l_gwiazdek*.

Warto zapamiętać

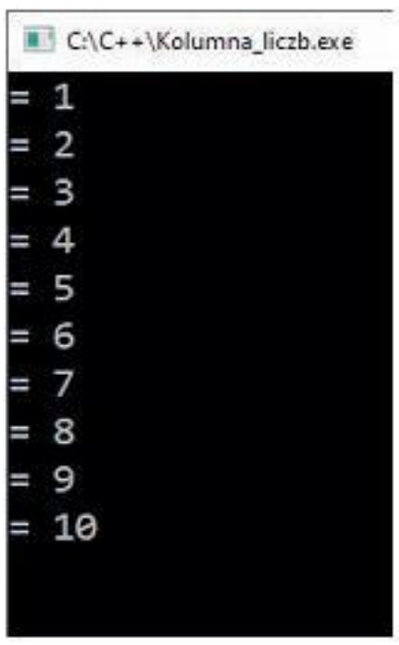
- Funkcję niezwracającą wartości stosujemy, gdy mamy wykonać pewne czynności, bez zwracania wartości. Można ją definiować z parametrami lub bez.
- W części nagłówkowej przed nazwą funkcji niezwracającej wartości należy umieścić słowo kluczowe **`void`**.
- Funkcję niezwracającą wartości wywołujemy, umieszczając nazwę funkcji z parametrami aktualnymi w odpowiednim miejscu programu głównego. W przypadku braku parametrów, po nazwie funkcji umieszczamy puste nawiasy.

Pytania i polecenia

1. Kiedy definiujemy funkcję niezwracającą wartości?
2. Omów ogólną postać definicji funkcji niezwracającej wartości.
3. W jaki sposób wywołujemy funkcję niezwracającą wartości?

Zadania

1. Zdefiniuj funkcję *kolumna()* typu **`void`** bez parametrów, wyświetlającą na ekranie w kolumnie liczby od 1 do 10 poprzedzone znakiem „=" i spacją – jak pokazano na rysunku 1. Funkcję wywołaj w programie głównym. Zapisz program w pliku pod nazwą *Kolumna_liczb*.



Rys. 1. Wynik działania programu – zadanie 1.

2. Zmodyfikuj program *Kolumna_liczb* zapisany w zadaniu 1., dodając do funkcji parametr *ile* określający liczbę wyświetlanych rzędów liczb. Funkcję wywołaj w programie głównym z parametrem aktualnym *ile_liczb*. Dodaj sprawdzanie poprawności wprowadzanych danych (muszą być większe od zera). Zapisz program w pliku pod nazwą *Kolumna_liczb_popr*.

Dla zainteresowanych

3. Wymyśl samodzielnie zadanie, w którym możesz wykorzystać funkcję niezwracającą wartości. Sformułuj zadanie i jego specyfikację. Napisz program realizujący to zadanie.

Temat 63. Deklarowanie i stosowanie tablicy w języku C++

1. Zmienne indeksowane
2. Deklarowanie tablicy w języku C++
3. Wprowadzanie elementów do tablicy i wyprowadzanie elementów na ekran

Warto powtórzyć

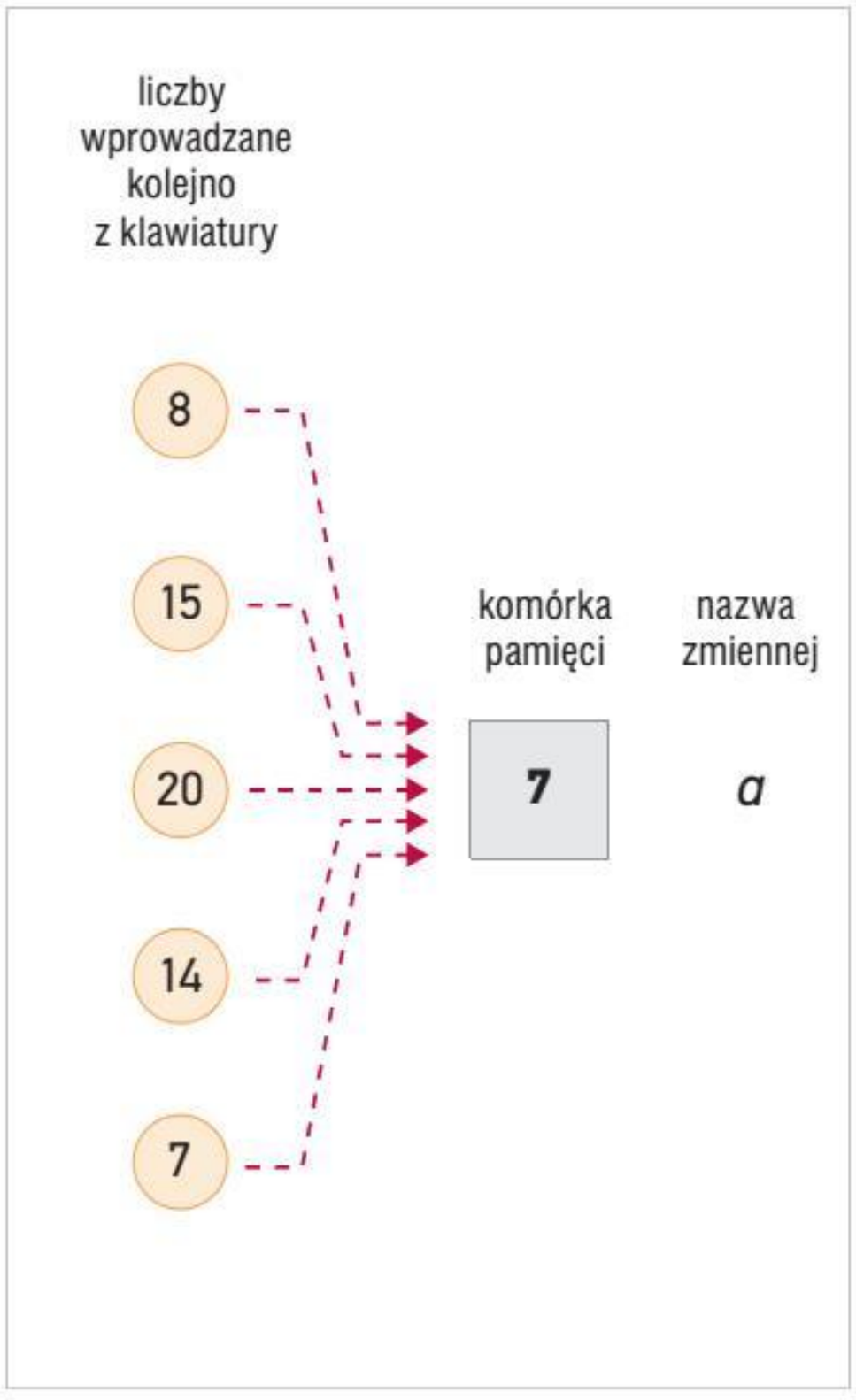
1. Do czego służą funkcje w językach programowania?
2. Kiedy stosujemy funkcję zwracającą wartość, a kiedy niezwracającą wartości?
3. W jaki sposób definiujemy funkcje w języku C++?
4. Jak wywołujemy funkcję z parametrami, a jak funkcję bez parametrów?
5. Jakie było działanie funkcji *suma()* omówionej w przykładzie 1. z tematu 61? Jaką wartość zwracała do programu głównego?

1. Zmienne indeksowane

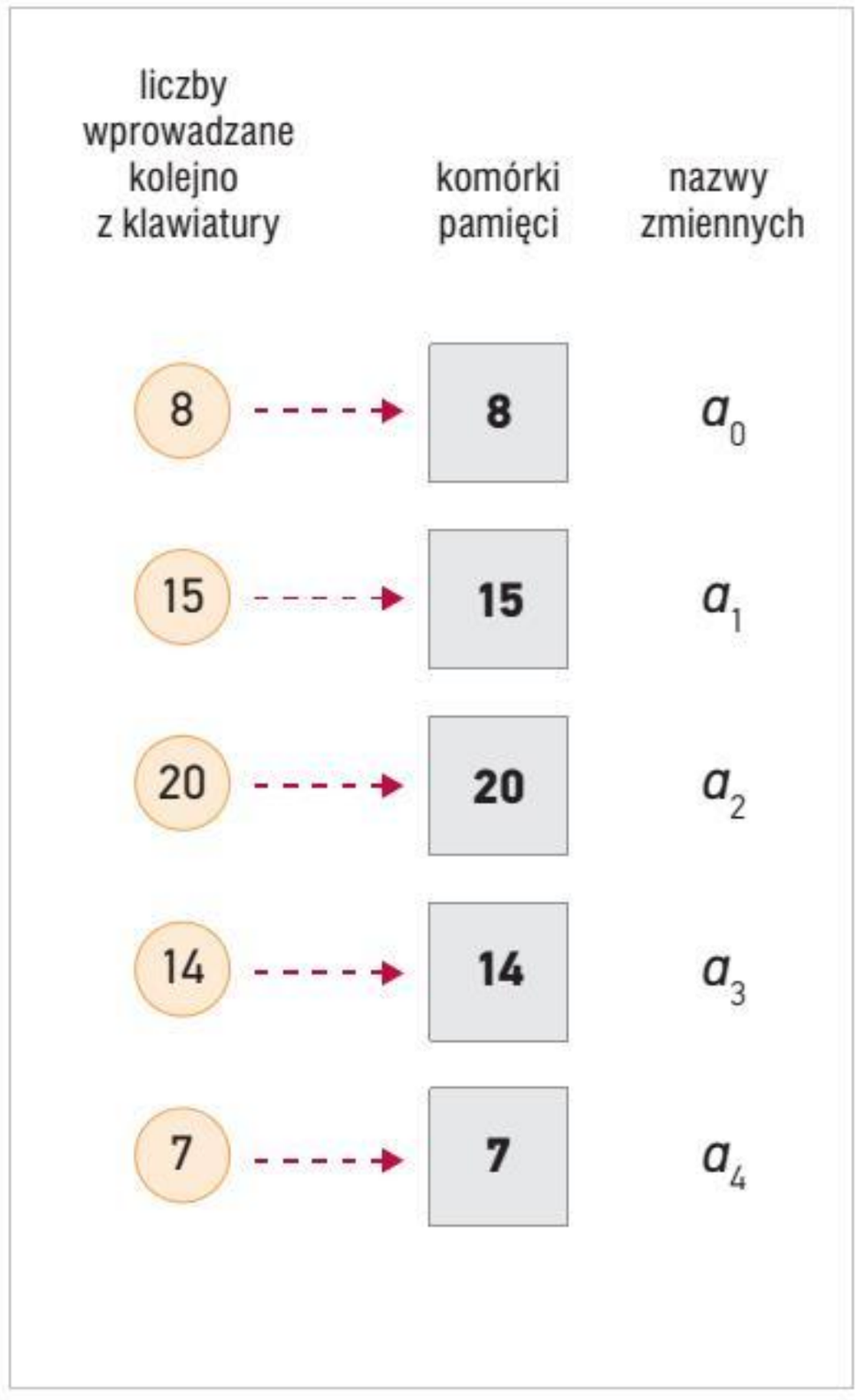
W funkcji obliczającej sumę liczb wprowadzanych z klawiatury wszystkie wprowadzane liczby komputer pamiętał kolejno w jednej zmiennej (o nazwie *a*), czyli nie zapamiętywał poszczególnych liczb. W jaki sposób zapamiętać wszystkie wprowadzone dane, aby można po ich wprowadzeniu odwoływać się do wybranych danych?

Gdy z klawiatury wprowadzimy kolejno liczby, na przykład wielkości dostaw telewizorów ze zbioru {8, 15, 20, 14, 7}, każda wprowadzana liczba zastępuje poprzednią (rys. 1a).

Aby zapamiętać wszystkie dane, musimy je różnie nazwać. Moglibyśmy użyć litery alfabetu, ale w przypadku dużej liczby danych zabrakłoby liter. Ponadto odwoływanie się do wielu zmiennych byłoby niewygodne. Dlatego stosujemy tzw. **zmienne indeksowane**, w których do nazwy zmiennej (np. *a*) dodaje się kolejny numer (**indeks**), np.: a_0, a_1, …, a_{n-1}. Wówczas dla każdej zmiennej komputer rezerwuje oddzielną komórkę pamięci, np. na poszczególne wielkości dostaw telewizorów z pięciu transportów (rys. 1b).



Rys. 1a. Zapisywanie w pamięci komputera wartości danych wprowadzanych z klawiatury z użyciem jednej zmiennej



Rys. 1b. Zapisywanie w pamięci komputera wartości danych wprowadzanych z klawiatury z użyciem pięciu zmiennych indeksowanych

Ćwiczenie 1. Wyjaśniamy, czym są zmienne indeksowane

Objaśnij, czym są zmienne indeksowane, korzystając z przygotowanych pomocy dydaktycznych, np. kartek z przykładowymi liczbami.

2. Deklarowanie tablicy w języku C++

Zmienne indeksowane w języku C++ możemy wykorzystywać, tworząc specjalną strukturę danych – **tablicę**.

Ogólna postać deklarowania tablicy	`opis_typu nazwa_tablicy [liczba_elementów_tablicy];`

Do elementów tablicy odwołujemy się, podając nazwę tablicy i indeks elementu umieszczony w nawiasach kwadratowych, np.: *a*[0], *a*[1], …, *a*[*n* - 1] – dla tablicy *n*-elementowej o nazwie *a*. W języku C++ pierwszy indeks jest zawsze równy 0.

Liczbę elementów tablicy musimy określić wcześniej:

- wpisać w deklaracji tablicy jako konkretną wartość (liczbę naturalną większą od 0)

lub

- podać za pomocą wcześniej zdefiniowanej stałej (**const**). Nie można zmienić liczby elementów raz zadeklarowanej tablicy.

! Uwaga

W języku C++ stałe będziemy zapisywać wielkimi literami. Wielkie i małe litery są rozróżniane (np. *n* i *N* oznaczają dwie różne zmienne lub stałe).

Deklaracja tablicy	Opis	Sposób odwołania
`int sztuki[100];`	oznacza zadeklarowanie tablicy o nazwie *sztuki* składającej się ze stu elementów typu całkowitego	do elementów tablicy odwołujemy się przez zmienne: *sztuki*[0], *sztuki*[1], …, *sztuki*[99]
`const int N = 20;` `int a[N];`	oznacza zadeklarowanie tablicy o nazwie *a* składającej się z dwudziestu elementów typu całkowitego (**int**)	do elementów tablicy odwołujemy się przez zmienne: *a*[0], a[1], …, *a*[*N*-1]
`const int LICZBA_EL = 80;` `float tablica[LICZBA_EL];`	oznacza zadeklarowanie tablicy o nazwie *tablica* składającej się z osiemdziesięciu elementów typu rzeczywistego (**float**)	do elementów tablicy odwołujemy się przez zmienne: *tablica*[0], *tablica*[1], …, *tablica*[*LICZBA_EL*-1]

Tabela 1. Przykłady deklarowania tablicy w języku C++

Początkowe wartości elementów tablicy można podać w momencie jej definiowania, zapisując je w nawiasach klamrowych.

Deklaracja tablicy	Opis
`int DOSTAWY[5] = {8,15,20,14,7};`	tablica *DOSTAWY*, składająca się z pięciu liczb całkowitych, zostanie wypełniona kolejnymi liczbami ze zbioru {8, 15, 20, 14, 7}
`int DOSTAWY[] = {8,15,20,14,7};`	jak wyżej, ale liczba elementów tablicy *DOSTAWY* (pięć elementów) zostanie ustalona na podstawie liczby danych początkowych
`int sztuki[100] = {0};`	tablica *sztuki*, składająca się ze stu liczb całkowitych, zostanie wypełniona liczbami 0

Tabela 2. Przykłady ustalania początkowych wartości tablicy (inicjalizowania tablicy) w języku C++

3. Wprowadzanie elementów do tablicy i wyprowadzanie elementów na ekran

Do wprowadzania danych do tablicy oraz do wyprowadzania elementów z tablicy na ekran monitora możemy zastosować funkcję niezwracającą wartości bez parametrów lub z parametrami.

Przykład 1. Deklarowanie i stosowanie tablicy w języku C++

Zadanie. Wprowadź *N* liczb całkowitych do tablicy o nazwie *a*, następnie wyprowadź w kolumnie elementy tablicy na ekran. Zdefiniuj dwie funkcje niezwracające wartości bez parametrów *wprowadz_dane()* i *wyprowadz_dane()*, które wywołaj w programie głównym.

Dane: stała wartość *N*, tablica liczb *a*[*N*].

Wyniki: wyświetlone na ekranie w kolumnie elementy tablicy *a*: *a*[0], a[1], …, *a*[*N*-1].

[*] Elementy_tablicy.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  const int N = 5;
5  int a[N];
6
7  void wprowadz_dane()
8  {
9      for(int i = 0; i < N; i++)
10     {
11         cout << "Podaj dana nr " << i << ": ";
12         cin >> a[i];
13     }
14 }
15
16 void wyprowadz_dane()
17 {
18     for(int i = 0; i < N; i++)
19         cout << "a[" << i << "] = " << a[i] << endl;
20 }
21
22 main ()
23 {
24     wprowadz_dane();
25     wyprowadz_dane();
26     return 0;
27 }
```

deklaracja stałej *N* o wartości 5

deklaracja tablicy o nazwie *a*

definicja funkcji *wprowadz_dane*

definicja funkcji *wyprowadz_dane*

wywołanie funkcji *wprowadz_dane*

wywołanie funkcji *wyprowadz_dane*

Ćwiczenie 2. Wprowadzamy dane do tablicy i wyprowadzamy je na ekran

1. Utwórz nowy plik źródłowy i przepisz program z przykładu 1. Uruchom i przetestuj program dla kilku różnych wartości zmiennych.
2. Zapisz program w pliku pod nazwą *Elementy _tablicy*.
3. Uruchom i przetestuj program dla kilku różnych wartości elementów tablicy *a*.

Ćwiczenie 3. Wyprowadzamy elementy z tablicy w odwrotnej kolejności

1. Otwórz program *Elementy _tablicy* zapisany w ćwiczeniu 2. Zmodyfikuj program, aby dane wyświetlały się w odwrotnym porządku. Dodatkowo zwiększ liczbę danych do dziesięciu.
2. Zapisz program w pliku pod nazwą *Odwrotna_kolejnosc*.

Ćwiczenie 4. Wyświetlamy na ekranie wybrany element tablicy

1. Otwórz program *Elementy _tablicy* zapisany w ćwiczeniu 2.
2. Zmodyfikuj program, aby na ekranie wyświetlać tylko element *k*-ty, gdzie *k* jest liczbą całkowitą wprowadzaną z klawiatury po uruchomieniu programu.
3. Zapisz program w pliku pod nazwą *Wybrany _element.*

Wskazówka: Należy sprawdzić, czy wprowadzona wartość *k* jest większa lub równa 0 oraz mniejsza od *N*.

Warto zapamiętać

- Aby zaprogramować algorytm, w którym w trakcie działania programu musimy pamiętać wszystkie elementy zbioru, możemy zapisywać dane w tablicy.
- W języku C++ liczba elementów (rozmiar) tablicy musi być z góry określona, np. jako zdefiniowana wcześniej stała. Musimy również określić typ elementów.
- W języku C++ do elementów tablicy, np. o nazwie *tab* i liczbie elementów *N*, odwołujemy się poprzez podanie nazwy i indeksu umieszczonego w nawiasach kwadratowych *tab*[0], *tab*[1], …, *tab*[*N* - 1].

Pytania i polecenia

1. Czym jest zmienna indeksowana?
2. W jakim celu definiujemy tablicę? Podaj przykład użycia tablicy.
3. Wyjaśnij na przykładach sposób deklarowania tablicy w języku C++.

Zadania

1. Otwórz program *Elementy _tablicy* zapisany w ćwiczeniu 2. Zmodyfikuj program, aby na ekranie wyświetlić tylko pierwszy i ostatni element. Zapisz program w pliku pod nazwą *Element_pierwszy _i_ostatni*.
2. Otwórz program *Element_pierwszy _i_ostatni* zapisany w zadaniu 1. Zmodyfikuj program, aby sprawdzać, który element jest większy: pierwszy czy ostatni. Wyświetlaj tylko element większy oraz komunikat, który to element. W przypadku elementów równych wyświetlaj komunikat „Elementy rowne”. Zapisz program w pliku pod nazwą *Element_wiekszy*.

Temat 64.

Wykonywanie operacji na elementach tablicy w języku C++

1. Odwoływanie się do elementów tablicy
2. Zamiana miejscami wartości elementów tablicy

Warto powtórzyć

1. W jaki sposób możemy określić w języku C++ liczbę elementów tablicy?
2. Dlaczego do wprowadzania danych do tablicy zdefiniowaliśmy funkcję typu `void`?
3. W jaki sposób wywołujemy w programie głównym funkcję niezwracającą wartości bez parametrów, a w jaki – z parametrami?
4. Na czym polegał problem dostaw artykułów przywożonych do magazynu RTV (temat 61.)?

1. Odwoływanie się do elementów tablicy

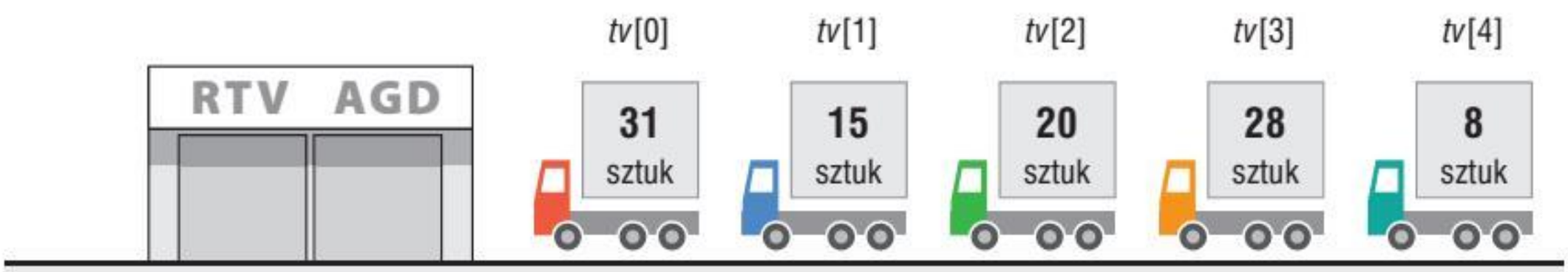
Zliczając telewizory przywożone do magazynu RTV (temat 61.), nie korzystaliśmy z oddzielnych zmiennych do pamiętania, ile sztuk telewizorów przywieziono w poszczególnych dostawach. Jakiej struktury danych powinniśmy użyć w programie, aby komputer pamiętał wprowadzone dane?



Do rozwiązania opisanego problemu zastosujemy oczywiście tablice. Jeśli wszystkie wprowadzone dane pamiętamy w tablicy (tu: liczby telewizorów z każdej dostawy), możemy wykonywać na nich różne operacje, np. odwoływać się do danej dostawy po wprowadzeniu wszystkich danych, wyświetlić wielkości wszystkich dostaw lub wybranej dostawy, wyświetlić dostawy w odwrotnej kolejności, zamienić miejscami wielkości dostaw, zsumować wybrane elementy czy uporządkować dostawy według wielkości (zastosowanie tablic w algorytmach porządkowania poznamy w dalszych tematach).

Przykład 1. Wprowadzamy wielkości dostaw telewizorów do tablicy

Zdefiniujemy funkcję *wprowadz_telewizory*() wprowadzającą wielkości kolejnych dostaw telewizorów do tablicy. Zadeklarujemy tablicę o nazwie *tv*. Liczbę elementów tablicy podamy za pomocą wcześniej zdefiniowanej stałej *T*.



Deklaracja:

```
const int T = 5;
int tv[T];
```

oznacza zadeklarowanie tablicy składającej się z pięciu liczb całkowitych o indeksach od 0 do 4.

Ponadto założymy, że dostawy będą numerowane od 1 (aby pierwszą dostawę określać jako „pierwszą", a nie „zerową" itd.).

Dostawy_tv_tablica.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  const int T = 5;
5  int tv[T];
6
7  void wprowadz_telewizory()
8  {
9      for(int i = 0; i < T; i++)
10     {
11         cout << "Podaj liczbe telewizorow w " << i + 1 << " dostawie: ";
12         cin >> tv[i];
13     }
14 }
```

Ćwiczenie 1. Wprowadzamy wielkości dostaw telewizorów do tablicy i wyświetlamy je na ekranie

1. Utwórz nowy plik źródłowy. Zdefiniuj funkcję *wprowdz_telewizory()* wprowadzającą elementy (tu: liczby telewizorów w kolejnych dostawach) do tablicy (przykład 1.). Funkcję wywołaj w programie głównym. Dlaczego w 11. wierszu do wartości zmiennej *i* dodajemy 1?
2. Zdefiniuj funkcję *wyprowdz_telewizory()* wyprowadzającą elementy (tu: liczby telewizorów w kolejnych dostawach) z tablicy *tv* na ekran monitora. Funkcję wywołaj w programie głównym.
3. Zapisz program w pliku pod nazwą *Dostawy_tv_tablica*. Uruchom i przetestuj program dla kilku różnych wartości wprowadzanych elementów tablicy *tv*.

Ćwiczenie 2. Wyświetlamy na ekranie wielkość *k*-tej dostawy telewizorów

1. Uzupełnij program zapisany w ćwiczeniu 1. o wyświetlanie na ekranie liczby telewizorów z wybranej dostawy. Numer dostawy *k* wprowadzaj z klawiatury.
2. Zapisz program w pliku pod nazwą *Wybrana_dostawa*. Uruchom i przetestuj program dla kilku różnych wartości elementów tablicy *tv* i zmiennej *k*.

Wskazówka: Pamiętaj, że elementy tablicy są numerowane od zera, natomiast numeracja dostaw zaczyna się od 1 (takie przyjęliśmy założenie).

Ćwiczenie 3. Porównujemy wielkości dwóch dostaw

1. Do programu zapisanego w ćwiczeniu 2. dodaj sprawdzanie, która dostawa telewizorów jest większa: druga czy czwarta (numery dostaw zaczynają się od 1). Zależnie od wyniku, wyprowadź na ekran komunikat: „Wieksza jest druga dostawa" lub „Wieksza jest czwarta dostawa". W przypadku dostaw o tej samej liczbie telewizorów wyprowadź komunikat „Dostawy rowne".
2. Zapisz program w pliku pod nazwą *Wieksza_dostawa*.

2. Zamiana miejscami wartości elementów tablicy

Okazało się, że w magazynie RTV pomylono półki podczas umieszczania telewizorów z danych dostaw. Jak zamienić te dostawy miejscami? Jak to zrobić w programie komputerowym, gdy wielkości dostaw wprowadzono do tablicy?

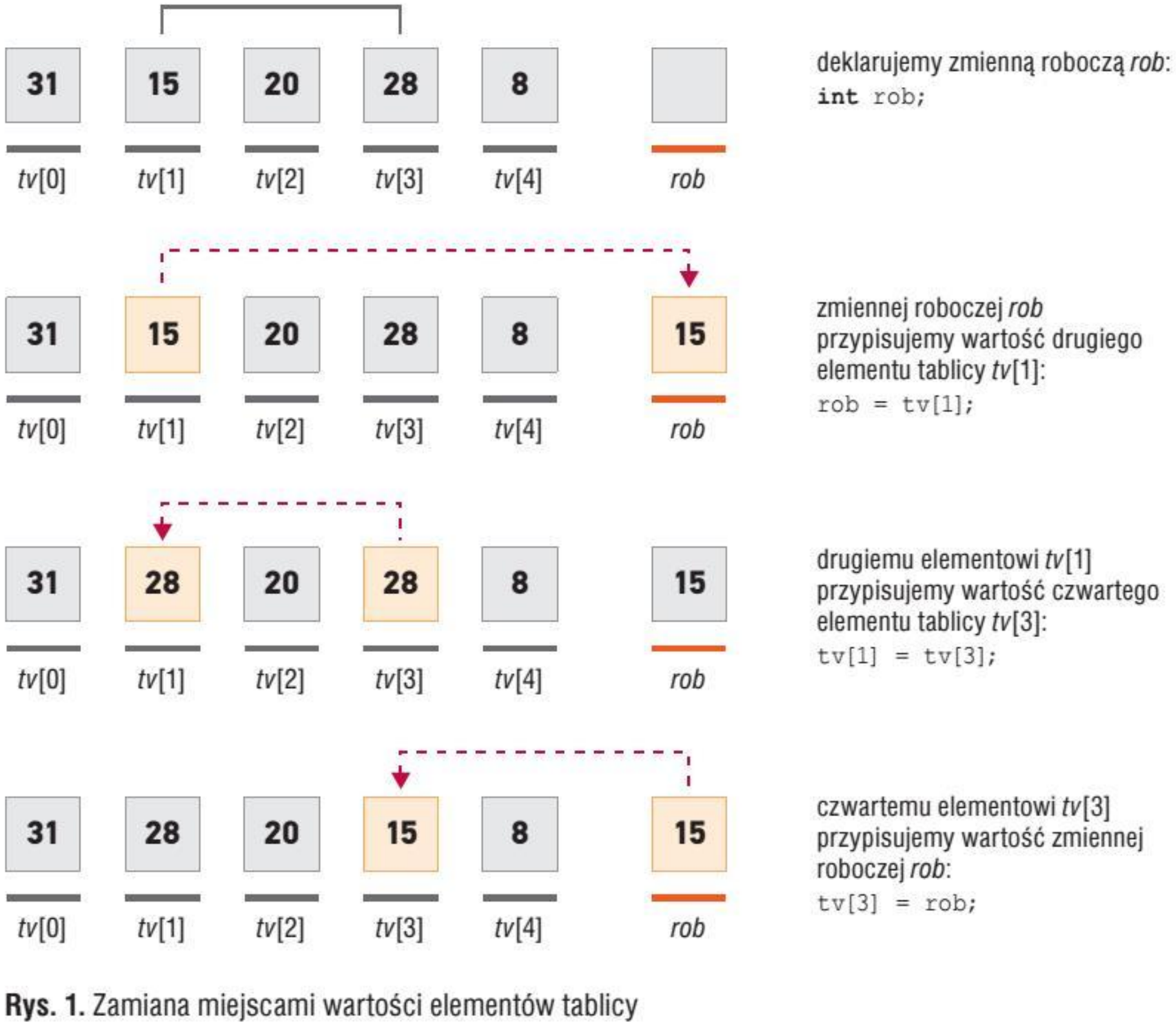
Aby przełożyć telewizory z półki A na półkę B, musimy telewizory z półki A gdzieś tymczasowo odłożyć, np. na półkę roboczą C, następnie telewizory z półki B przełożyć na półkę A, a te z półki roboczej C – na B.

Podobnie postępujemy w przypadku programu komputerowego, w którym dane (liczby poszczególnych dostaw) pamiętamy w tablicy – również używamy tzw. zmiennej roboczej do chwilowego zapamiętania „odłożonej" wartości danego elementu tablicy.

Przykład 2. Zamienianie miejscami wartości elementów tablicy

W tablicy *tv*, w której pamiętamy dostawy telewizorów, chcemy zamienić miejscami drugą dostawę z czwartą, czyli wartości elementów tablicy o indeksach 1 i 3 (*tv*[1] z *tv*[3]).

Aby zamienić miejscami elementy listy, posłużymy się zmienną roboczą *rob* (podobnie jak półką roboczą C). Przypisanie `rob = tv[1]` oznacza, że wartość zmiennej *rob* zmieni się na 15, ale jednocześnie wartość 15 pozostanie w elemencie *tv*[1]. Dopiero przypisanie `tv[1] = tv[3]` zmienia wartość *tv*[1] na 28.



Rys. 1. Zamiana miejscami wartości elementów tablicy

Wskazówki:

- Do analizy przykładu możesz posłużyć się odpowiednio przygotowanymi pomocami (np. kartkami z zapisanymi liczbami) lub zapisać dane na zwykłej tablicy i odpowiednio je wymazywać.
- Aby zamienić miejscami wartości elementów *tv*[1] z *tv*[3], należy umieścić w programie kolejno instrukcje:

```
rob = tv[1];
tv[1] = tv[3];
tv[3] = rob;
```

! Uwaga

Zamianę elementów tablicy wykorzystamy do porządkowania elementów tablicy metodą bąbelkową (temat 92.).

Ćwiczenie 4. Zamieniamy miejscami wartości elementów tablicy

1. Przeanalizuj przykład 2. Odpowiedz na pytanie: *dlaczego po zamianie miejscami elementów w zmiennej* rob *jest nadal pamiętana wartość 15*?
2. Zmodyfikuj program zapisany w ćwiczeniu 3., aby w przypadku, gdy czwarta dostawa jest większa od drugiej, zamienić je miejscami. Wyprowadź na ekran wszystkie dostawy po tej zmianie poprzedzone komunikatem „Dostawy po zamianie miejscami drugiej z czwarta, gdy czwarta jest wieksza od drugiej".
3. Zapisz program w pliku pod nazwą *Zamiana_dostaw*. Uruchom i przetestuj program dla kilku różnych wartości elementów tablicy *tv*.

Warto zapamiętać

- Zapisanie elementów w tablicy umożliwia m.in. wyświetlenie na ekranie wartości wybranego elementu tablicy, wartości elementów w odwrotnej kolejności, zamianę wartości elementów miejscami, zsumowanie wybranych elementów.
- Aby zamienić miejscami elementy tablicy, należy posłużyć się zmienną do tymczasowego przechowania jednego z elementów.

Pytania i polecenia

1. W jaki sposób odwołujemy się do wybranego elementu tablicy?
2. Wyjaśnij na przykładzie sposób zamiany miejscami wartości elementów tablicy.

Zadania

1. Zadeklaruj tablicę składającą się z pięciu elementów, będących liczbami godzin lekcyjnych w planie lekcji danej klasy od poniedziałku do piątku. Utwórz program wypisujący w kolumnie elementy tablicy, czyli liczby lekcji każdego dnia. Zapisz program w pliku pod nazwą *Lekcje_w_tygodniu*.
   Wskazówka: Zastosuj sposób ustalania początkowych wartości tablicy pokazany w pierwszym wierszu tabeli 2. z tematu 63.
2. Korzystając z tablicy zdefiniowanej w zadaniu 1., utwórz program, który dla podanego z klawiatury numeru dnia tygodnia wyświetli liczbę lekcji tego dnia. Zapisz program w pliku pod nazwą *Lekcje_w_dniu*.

Dla zainteresowanych

3. Napisz program, który wczytuje liczby całkowite do ośmioelementowej tablicy i wypisuje na ekran indeks pierwszego elementu tablicy, którego wartość wynosi zero. Jeśli nie ma takiego elementu, program powinien wyświetlić komunikat „Brak elementu zero". Zapisz program w pliku pod nazwą *Indeks_elementu_zero*.

Temat 65. Dane tekstowe w języku C++

1. Deklarowanie danych tekstowych
2. Wprowadzanie i wyświetlanie na ekranie danych tekstowych
3. Łączenie napisów
4. Używanie danych tekstowych

Warto powtórzyć

1. Co określa typ zmiennej?
2. W jaki sposób deklarujemy zmienne?
3. Jakiego typu zmienne stosowaliśmy w programach?

1. Deklarowanie danych tekstowych

Zapisując wybrany algorytm w języku programowania, m.in. dobieramy odpowiednie struktury danych. Jeśli algorytmy dotyczą działań na liczbach, stosujemy dane liczbowe, jeśli pracy na tekstach – tekstowe typy danych.

Jeśli w języku C++ danymi do zadania są pojedyncze znaki, wówczas deklarujemy zmienne typu znakowego.

Deklaracja zmiennych typu znakowego	`char zmienna_znakowa;`

Stałe znakowe umieszcza się w apostrofach (w odróżnieniu od tekstów, które umieszcza się w górnych cudzysłowach):

```
znak = 'x';
```

W języku C++ zmienna typu **char** ma również wartość liczbową, odpowiadającą kodowi ASCII lub Unicode znaku zapamiętanego w zmiennej. Na przykład, po wykonaniu następującego ciągu instrukcji:

```
char znak;
znak = 'x';
znak++;
cout << znak;
```

na ekranie zostanie wyprowadzony znak „y”.

Jeśli danymi w algorytmie są teksty lub inne ciągi znaków, to zapisując algorytm w postaci programu, należy zmienne tekstowe zadeklarować jako typ łańcuchowy (ciąg znaków w języku C++ nazywamy **łańcuchem**).

W celu łatwiejszego korzystania z łańcuchów można wykorzystać klasę standardową `std::string`. Aby ją wykorzystać, należy umieścić na początku programu polecenie: **`#include`** `<string>`.

Łańcuch to również tablica znaków o dowolnym, wcześniej ustalonym rozmiarze. W języku C++ łańcuch zakończony jest znakiem `NULL`, zapisywanym jako `'\0'`. Przy deklarowaniu zmiennej łańcuchowej należy więc dodać 1 do przewidywanego rozmiaru tekstu, by zmieścić znak `NULL`.

Deklaracja zmiennych typu tekstowego	**`string`** *`lancuch`*`;` **`char`** *`lancuch`*`[n + 1];`

Aby w języku C++ wyznaczyć długość łańcucha, możemy użyć standardowej funkcji `length()`. Na przykład `napis.length()` oznacza długość łańcucha pamiętanego w zmiennej *napis* typu `string`.
Do wyznaczenia długości tekstu zapisanego jako tablica znaków można użyć funkcji standardowej `strlen()`.

Do pojedynczego znaku łańcucha odwołujemy się tak samo jak do elementu tablicy, czyli używając indeksu: `lancuch[i]`. Znaki łańcucha są numerowane od zera. Indeks ostatniego znaku obliczamy zatem, odejmując 1 od długości napisu, np. `napis.length() - 1`.

Deklaracja łańcucha o długości do 15 znaków	**`char`** `slowo[16];`	W zmiennej *slowo* można zapamiętać napis o długości do 15 znaków.
Deklaracja łańcucha o długości do 19 znaków	**`const int`** `N = 20;` **`char`** `tekst[N];`	W zmiennej *tekst* można zapamiętać napis o długości do 19 znaków.
Deklaracja łańcucha z wykorzystaniem klasy `string`	**`#include`** `<string>` … `string napis;`	Liczba znaków pamiętanych w zmiennej typu `string` jest ograniczona tylko przez pamięć RAM, czyli teoretycznie w systemie 32-bitowym można w zmiennej *napis* zapamiętać 2^{32} znaków.

Tabela 1. Przykłady deklarowania zmiennych tekstowych w języku C++

Deklaracja łańcucha danej długości to określenie maksymalnej długości łańcucha. Oznacza to, że zawarty w nim napis może zawierać mniejszą liczbę znaków. Jeśli wprowadzony łańcuch jest krótszy niż zadeklarowany, nie wolno odwoływać się do nieistniejących znaków, np. w przypadku wykonania instrukcji:

```
slowo = "drzewo";
```

odwołanie *slowo*[8] będzie błędne, ponieważ słowo „drzewo” składa się z sześciu liter.

2. Wprowadzanie i wyświetlanie na ekranie danych tekstowych

Przykład 1. Wprowadzanie danych tekstowych w języku C++

W języku C++ dane tekstowe możemy:

- wprowadzać z klawiatury:
  ```
  cin >> napis;
  ```
- zastosować instrukcję przypisania:
  ```
  napis = "Akumulator";
  ```
- przypisać wartość zmiennej łańcuchowej w momencie jej deklarowania:
  ```
  string napis = "Akumulator";
  ```
  lub
  ```
  char napis[11] = "Akumulator";
  ```
  lub
  ```
  char napis[11] = {'A', 'k', 'u', 'm', 'u', 'l', 'a', 't', 'o', 'r', '\0'};
  ```
  lub
  ```
  char napis[] = "Akumulator";
  ```
  (wielkość tablicy zostanie dopasowana do napisu, w tym przypadku będzie to 10 + 1 znaków).

Aby wyprowadzić napis na ekran, używamy (tak jak w przypadku liczb i komunikatów) polecenia `cout <<`. Możemy też wyprowadzić na ekran litery wprowadzonego napisu w kolumnie, używając instrukcji iteracyjnej **`for`** (rys. 1.).

[*] Wyprowadzanie_napisu.cpp

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string wyraz;
8      cout << "Podaj wyraz: ";
9      cin >> wyraz;
10     cout << "Wprowadzony wyraz: " << wyraz << endl;
11     cout << "Litery wyrazu " << wyraz << " w kolumnie" << endl;
12     for(int i = 0; i < wyraz.length(); i++)
13         cout << wyraz[i] << endl;
14     return 0;
15 }
```

wyświetlanie napisu w wierszu

wyświetlanie liter napisu w kolumnie

Rys. 1. Wyprowadzenie napisu na ekran

Ćwiczenie 1. Wyprowadzamy dane tekstowe na ekran

1. Napisz program wyprowadzający w wierszu wyraz wprowadzony z klawiatury, a następnie litery tego wyrazu w kolumnie. Skorzystaj z rysunku 1.
2. Zapisz program w pliku pod nazwą *Wyprowadzanie_napisu*.
3. Zmodyfikuj program tak, aby w kolejnych wierszach wyświetlała się co druga litera wprowadzonego wyrazu (zaczynając od pierwszej litery).
4. Zapisz plik pod tą samą nazwą. Przetestuj program dla kilku różnych danych.

3. Łączenie napisów

Napisy możemy ze sobą łączyć, używając operatora +. Na przykład instrukcja
`cout << "Stosujemy dane" + " " + "tekstowe w C++";`
wyprowadzi na ekran napis: „Stosujemy dane tekstowe w C++”.

Łączenie łańcuchów nazywamy **konkatenacją**. Możemy również łączyć łańcuchy, które wcześniej przypiszemy do zmiennych (rys. 2.).

```
 7      string s1, s2;
 8      s1 = "Stosujemy dane";
 9      s2 = "tekstowe w C++";
10      cout << s1 + " " + s2  << endl;
11
```

Rys. 2. Przykład łączenia napisów

Ćwiczenie 2. Łączymy napisy

1. Napisz program łączący dwa łańcuchy, które przypiszemy do zmiennych w programie (skorzystaj z wzoru na rysunku 2., ale wpisz inne napisy).
2. Zapisz program w pliku pod nazwą *Laczenie_lancuchow*.

4. Używanie danych tekstowych

Wyraz.cpp

```
 1  #include <iostream>
 2  #include <string>
 3  using namespace std;
 4
 5  int main()
 6  {
 7      string wyraz;
 8      int dlugosc;
 9
10      cout << "Podaj wyraz: ";
11      cin >> wyraz;
12      dlugosc = wyraz.length();
13          for(int i = 0; i < dlugosc; i++)
14          if(wyraz[i] == 'a')
15              wyraz[i] = 'b';
16          cout << "Wyraz po przetworzeniu: " << wyraz << endl;
17      return 0;
18  }
```

Rys. 3. Przykładowy program z wykorzystaniem danych tekstowych

Ćwiczenie 3. Analizujemy gotowy program

1. Przeanalizuj program pokazany na rysunku 3. Sformułuj treść zadania realizowanego przez ten program i napisz specyfikację zadania (określ dane i wyniki). Wyjaśnij działanie poszczególnych instrukcji.
2. Przepisz program i zapisz w pliku pod nazwą *Wyraz*.
3. Uruchom program i przetestuj dla kilku różnych danych.

Ćwiczenie 4. Stosujemy dane tekstowe

1. Napisz program wyprowadzający na ekran pierwszą i ostatnią literę wyrazu wprowadzonego z klawiatury.
2. Zapisz program w pliku pod nazwą *Wyraz_pierwsza_ostatnia*.
3. Przetestuj program dla kilku różnych danych.

Ćwiczenie 5. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 4. Program powinien zamieniać miejscami pierwszą literę z ostatnią, gdy są różne, a następnie wyprowadzić na ekran zmieniony wyraz. W przypadku jednakowych liter powinien się wyświetlić odpowiedni komunikat.
2. Zapisz program w pliku pod nazwą *Wyraz_zamiana_pierwsza_ostatnia*.
3. Przetestuj działanie programu dla kilku różnych danych.

Warto zapamiętać

- Typy danych dobiera się do rozwiązywanego zadania, uwzględniając rodzaj informacji, które mają być przetwarzane, oraz sposób ich przetwarzania.
- Zmienne typu znakowego deklarujemy, gdy danymi do zadania są pojedyncze znaki, natomiast gdy są to teksty lub inne ciągi znaków, to zmienne tekstowe deklarujemy jako typ łańcuchowy.
- Do pojedynczego znaku łańcucha odwołujemy się tak samo, jak do elementu tablicy, czyli używając indeksu, np. *napis*[*i*].

Pytania i polecenia

1. Danymi do zadania są pojedyncze znaki. Jaki typ danych zastosujesz, aby zapisać to zadanie w języku C++?
2. Danymi do zadania są ciągi znaków. Jaki typ danych zastosujesz, aby zapisać to zadanie w języku C++?
3. Jak odwołujemy się do pojedynczego elementu łańcucha? Pokaż na przykładzie.
4. Czy można zadeklarować łańcuch jako tablicę znaków? Pokaż na przykładzie.

Zadania

1. Napisz program, który umożliwi zamianę wszystkich znaków „a” na „b” oraz wszystkich znaków „b” na „a” we wprowadzonym tekście. Skorzystaj z programu pokazanego na rysunku 3. Zapisz program w pliku pod nazwą *Zamiana*.
2. Napisz program zamieniający co drugą literę wyrazu wprowadzonego z klawiatury na literę „x” i wyprowadzający zmieniony wyraz na ekran. Zapisz program w pliku pod nazwą *Wyraz_zamiana_na_x*. Przetestuj działanie programu dla kilku różnych danych.
   Wskazówka: Zastanów się, którą instrukcję pętli najlepiej tu zastosować.

```
Podaj wyraz: qwerty
Wyraz po zamianie co drugiej litery na x: qxextx
```

Rys. 4. Przykładowy wynik działania programu – zadanie 2.

3. Zadanie dla dwóch osób: Każda z osób projektuje zadanie według własnego pomysłu (można wzorować się na zadaniach z poznanych tematów) – formułuje treść zadania, pisze specyfikację i przekazuje zadanie drugiej osobie, aby napisała program. Potem program wraca do autora zadania, który testuje go dla różnych danych.

Dla zainteresowanych

4. Napisz program, który wyświetla na ekranie indeks tego znaku łańcucha wprowadzanego z klawiatury, który jest spacją (chodzi tu o pierwsze wystąpienie spacji). Jeśli nie ma takiego elementu, program wyświetla komunikat „brak spacji”.
5. Zaproponuj własne zadanie, w którym zastosujesz dane tekstowe i poznane instrukcje (m.in. warunkowe, iteracyjne). Sformułuj zadanie i jego specyfikację. Zapisz rozwiązanie w języku C++.

Rozdział XIII

Programowanie w języku Python

Temat 66. Środowisko programistyczne języka Python, stosowanie zmiennych i wyprowadzanie wyników

Temat 67. Algorytmy z warunkami w języku Python

Temat 68. Sprawdzanie poprawności danych i zapisywanie rozwiązania problemu w języku Python

Temat 69. Instrukcje warunkowe zagnieżdżone i z klauzulą `elif` w języku Python

Temat 70. Algorytmy iteracyjne w języku Python

Temat 71. Instrukcje iteracyjne zagnieżdżone w języku Python

Temat 72. Stosowanie instrukcji iteracyjnych `for` i `while` w języku Python

Temat 73. Funkcje zwracające wartość w języku Python

Temat 74. Funkcje niezwracające wartości w języku Python

Temat 75. Definiowanie i stosowanie listy w języku Python

Temat 76. Wykonywanie operacji na elementach listy w języku Python

Temat 77. Dane tekstowe w języku Python

Temat 66.

Środowisko programistyczne języka Python, stosowanie zmiennych i wyprowadzanie wyników

1. **Środowisko programistyczne języka Python**
   1.1. Pisanie poleceń w trybie interaktywnym
   1.2. Pisanie programów w trybie skryptowym
2. **Stosowanie zmiennych**
   2.1. Nazywanie zmiennych
   2.2. Nadawanie zmiennym wartości
   2.3. Wykonywanie obliczeń
3. **Wyprowadzanie komunikatów i wyników na ekran monitora**

Warto powtórzyć

1. Jakie informacje należy umieścić w specyfikacji zadania?
2. Wyjaśnij pojęcia: *język programowania*, *program komputerowy*, *słowo kluczowe*.
3. Dlaczego o języku programowania mówimy, że jest językiem formalnym?
4. Czym jest kod źródłowy?
5. Z czego powinna wynikać kolejność instrukcji programu?
6. Na czym polega interpretacja programu?

1. Środowisko programistyczne języka Python

Aby pisać programy w języku Python, należy zainstalować darmową aplikację Python, w skład której wchodzi **zintegrowane środowisko programistyczne** IDLE.

IDLE umożliwia tworzenie programów w języku Python, zapisywanie ich i uruchamianie. Zintegrowane środowisko programistyczne IDLE zawiera **powłokę Pythona** (**Python Shell**), **edytor kodu źródłowego** i **interpreter** oraz inne narzędzia wspomagające programowanie.

W tym podręczniku będziemy korzystać z aplikacji Python 3. Po zainstalowaniu aplikacji możemy ją uruchomić i rozpocząć programowanie.

Aby uruchomić IDLE, należy z menu **Start** wybrać program **Python 3.10/IDLE (Python 3.10 64-bit)**. Otworzy się okno programu powłoka Pythona (Python Shell). Okno powłoki Pythona umożliwia pracę w dwóch trybach: **interaktywnym** i **skryptowym**.



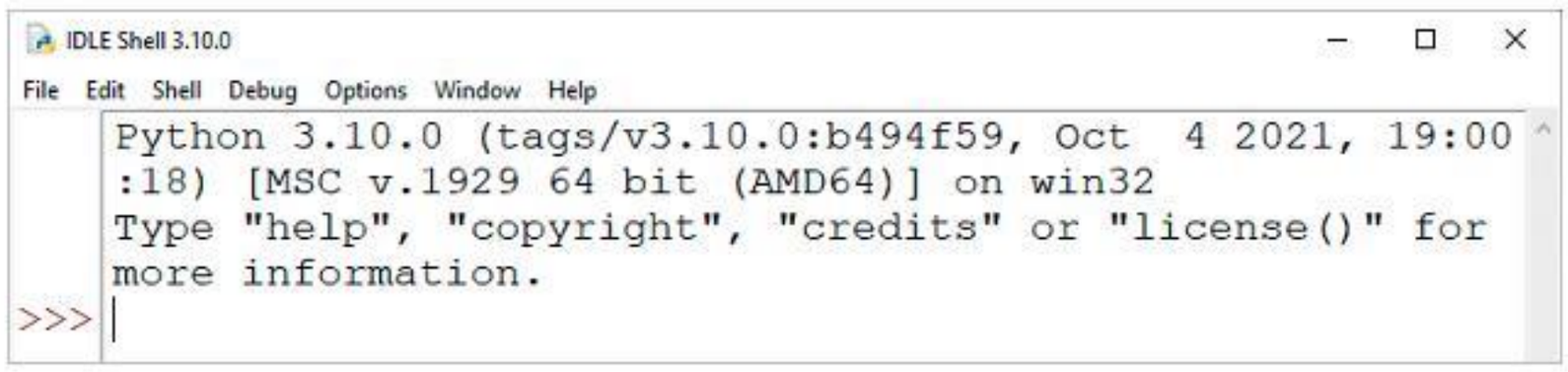
Rys. 1. Oficjalna strona internetowa służąca m.in. do pobierania aplikacji Python: *https://www.python.org/downloads/*

Z **Zintegrowane środowisko programistyczne** (IDE – z ang. *Integrated Development Environment*)

Aplikacja lub zespół aplikacji do tworzenia, modyfikowania, testowania i utrzymywania oprogramowania.

1.1. Pisanie poleceń w trybie interaktywnym

Powłoka Pythona uruchamia się w trybie interaktywnym. Po znaku zachęty >>> widzimy migający kursor. W tym miejscu możemy napisać instrukcję lub wyrażenie. Po naciśnięciu klawisza **Enter** instrukcja zostanie wykonana, a wyrażenie obliczone. Z trybem interaktywnym mogliśmy się spotkać w poznanych w szkole podstawowej językach: Logo i Scratch.



Rys. 2. Okno powłoki Pythona

Ćwiczenie 1. Sprawdzamy działanie trybu interaktywnego

1. Uruchom środowisko IDLE.
2. W oknie powłoki Pythona sprawdź działanie trybu interaktywnego. Wpisz po znaku zachęty:
   a. `2345 + 564` i naciśnij **Enter**,
   b. `print("Pracuję w trybie interaktywnym")` i naciśnij **Enter**.

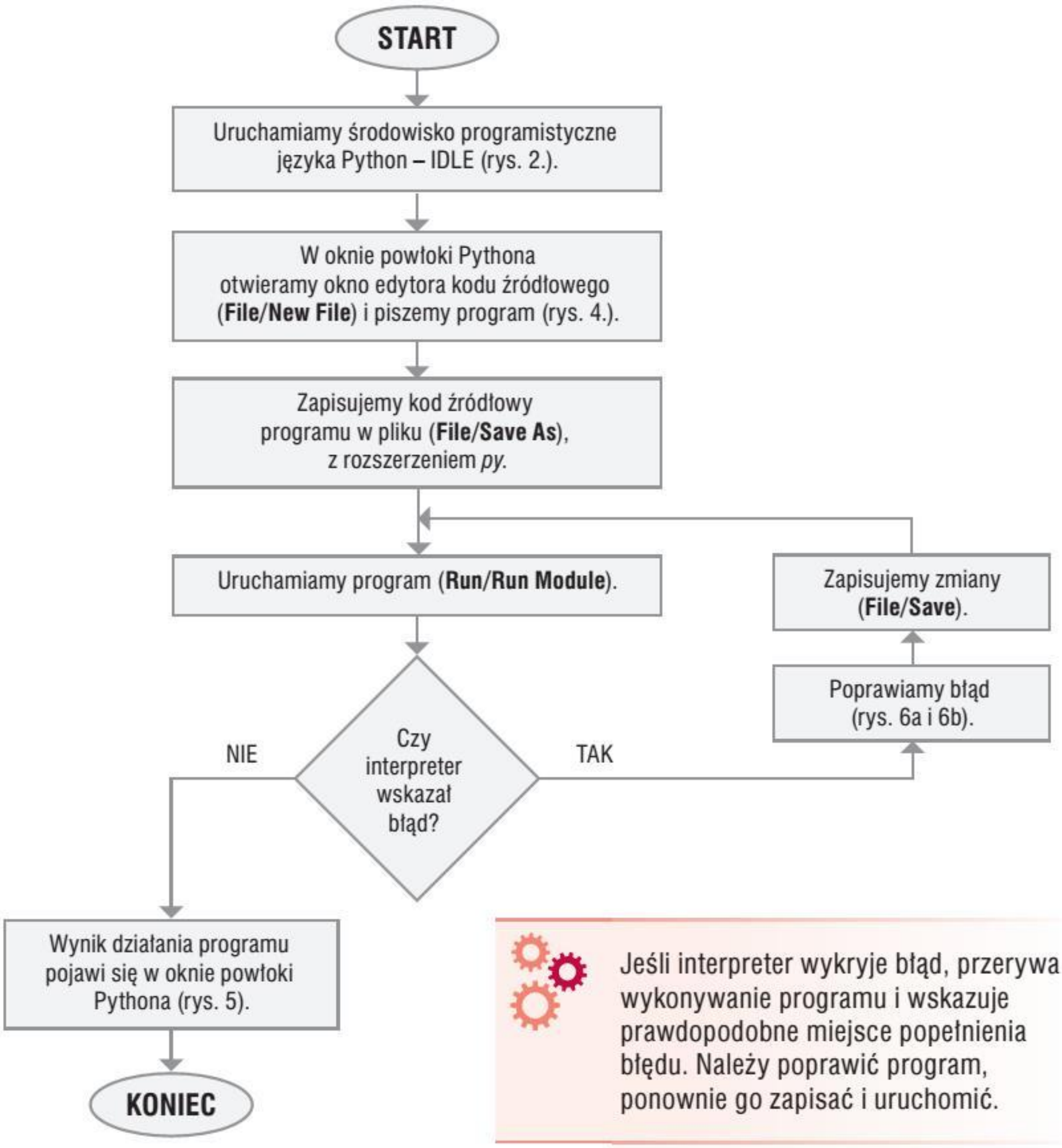
W trybie interaktywnym możemy sprawdzić działanie jednego polecenia, np. gdy chcemy od razu zobaczyć wynik obliczeń czy sprawdzić działanie konkretnej instrukcji. Natomiast do tworzenia programów w języku Python służy tryb skryptowy.

1.2. Pisanie programów w trybie skryptowym

W językach interpretowanych, takich jak Python, możemy napisać program składający się z wielu poleceń (tworząc tzw. **skrypt**), zapisać go w pliku tekstowym, a następnie uruchomić, używając interpretera. Mówimy wówczas, że pracujemy w trybie skryptowym.

Do trybu skryptowego możemy przejść z trybu interaktywnego z okna powłoki Pythona. Program piszemy w edytorze kodu źródłowego, zapisujemy w pliku na dysku, a następnie uruchamiamy (poprzez wywołanie interpretera Pythona).

W języku Python możemy wyróżnić etapy tworzenia programu pokazane na schemacie (rys. 3.).



Rys. 3. Etapy tworzenia programu w interpretowanym języku programowania Python

Kilka zasad pisania programu w języku Python

- W programowaniu w języku Python, jak w każdym języku, bardzo istotna jest prawidłowa kolejność zapisu poleceń.
- Instrukcje języka Python, podobnie jak innych języków programowania, realizują takie czynności, jak: wprowadzanie danych, wyprowadzanie wyników, wykonywanie obliczeń, sprawdzanie warunków czy powtarzanie operacji.
- Bloki instrukcji wyodrębnia się inaczej niż w innych językach. Nie stosuje się nawiasów ani słów kluczowych, tylko wcięcia, składające się z przynajmniej jednej spacji (przyjęte jest wcięcie składające się z czterech spacji).
- Komentarze rozpoczyna się znakiem #. Tekst po znaku # do końca wiersza jest ignorowany przez interpreter.



Rys. 4. Okno edytora kodu źródłowego Pythona z programem źródłowym

Uwaga

Rozmiar czcionki można zmienić, wybierając w edytorze kodu źródłowego lub oknie powłoki Pythona opcję **Options/Configure IDLE**.

Ćwiczenie 2. Poznajemy etapy tworzenia programu w języku Python

1. Utwórz nowy plik źródłowy i napisz program wyświetlający na ekranie napis „Zaczynamy" – przepisz instrukcję pokazaną na rysunku 4.
2. Zapisz program w pliku pod nazwą *Napis*.
3. Uruchom program.
4. Jeśli interpreter wykrył błędy, popraw je. Zapisz plik pod tą samą nazwą i ponownie uruchom.

W trakcie wykonywania programu interpreter rozpoznaje polecenia języka Python i zamienia je na instrukcje wewnętrznego języka procesora. Po wykonaniu programu wyniki pojawią się na wybranym urządzeniu zewnętrznym, np. na ekranie monitora (tu w oknie powłoki Pythona – rys. 5.).

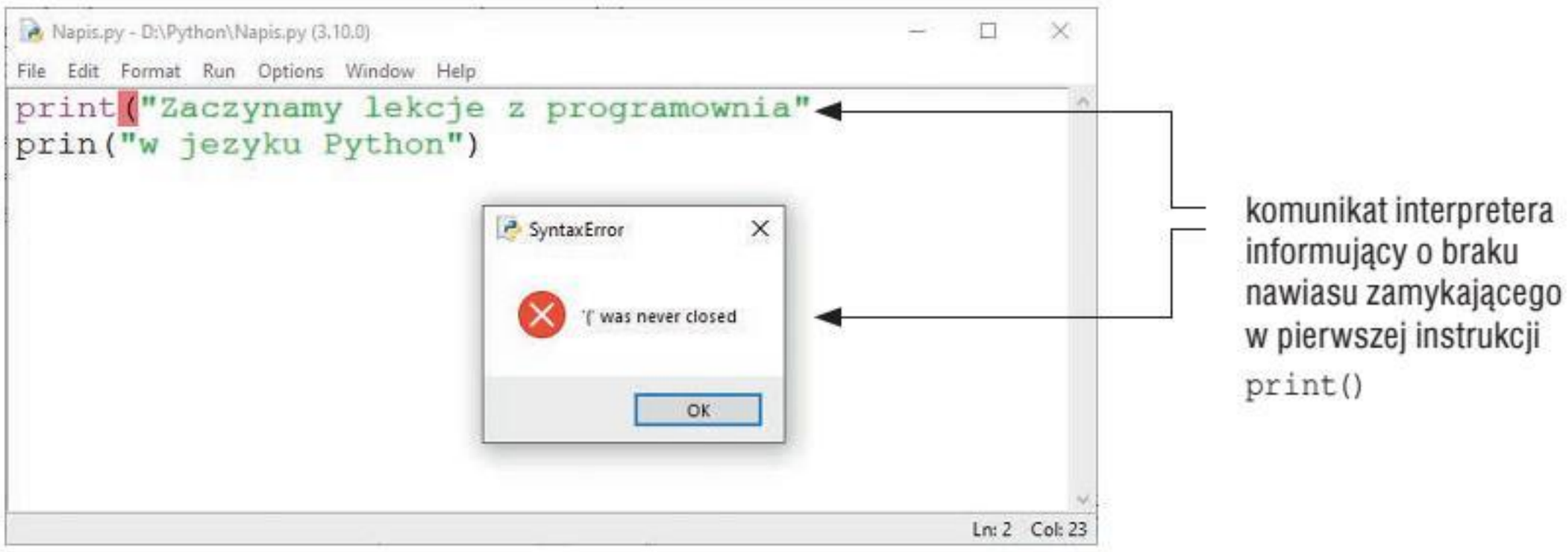


Rys. 5. Wynik działania programu – ćwiczenie 2.

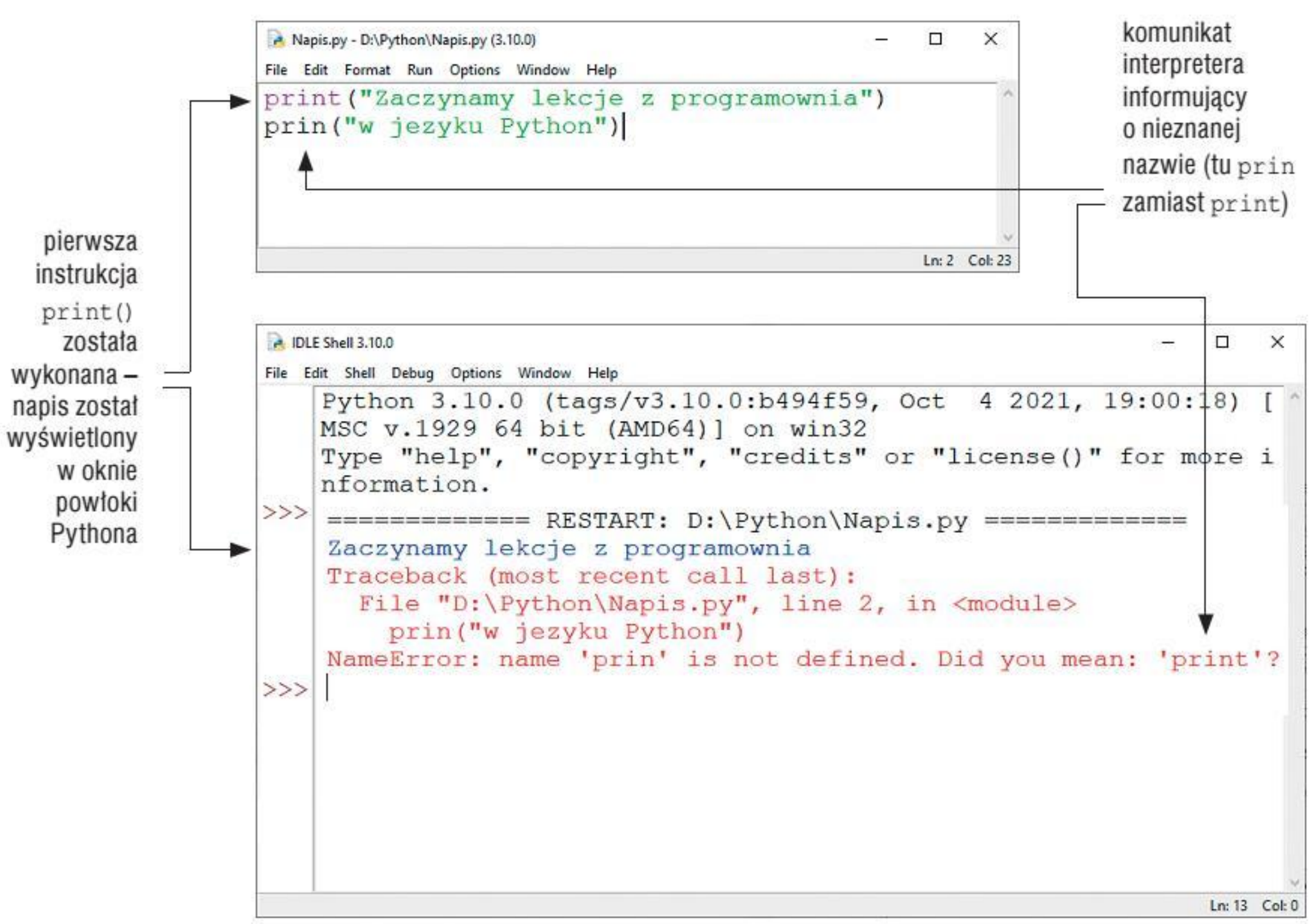
Uwaga

W nazwach plików, w których zapisujemy programy tworzone w języku Python, nie będziemy stosować polskich znaków diakrytycznych i spacji (w nazwach wieloczłonowych użyjemy znaku podkreślenia).

W odróżnieniu od kompilatora, **interpreter** próbuje wykonać program instrukcja po instrukcji. Jeżeli w programie wystąpi błąd, interpreter przerwie wykonanie programu i wyświetli komunikat o błędzie (rys. 6a i 6b). Należy poprawić wskazany błąd, ponownie zapisać program w pliku i uruchomić program.



Rys. 6a. Komunikat interpretera informujący o pierwszym błędzie



Rys. 6b. Okno powłoki Pythona z komunikatem interpretera informującym o drugim błędzie

Ćwiczenie 3. Modyfikujemy program

1. Zmodyfikuj program utworzony w ćwiczeniu 2., wyświetlając w tym samym wierszu dalszą część napisu: „lekcje z programowania", a w drugim wierszu – napis „w języku Python".
2. Jeśli interpreter wykrył błędy, popraw je. Zapisz plik pod tą samą nazwą. Ponownie uruchom program.

Wskazówki: Na rysunkach 6a i 6b pokazano przykładowe błędy wykryte przez interpreter. Na rysunku 6a interpreter podświetlił w pierwszym wierszu nawias otwierający, ponieważ brakuje nawiasu zamykającego na końcu tego wiersza.

Dobra rada

Staraj się samodzielnie poprawiać program do momentu, gdy da się go uruchomić. W nauce programowania umiejętność wyszukiwania i poprawiania błędów w programie jest bardzo ważna.

2. Stosowanie zmiennych

Chcemy obliczyć iloczyn dwóch liczb całkowitych. Jak napisać w języku Python program umożliwiający wprowadzenie dwóch liczb całkowitych z klawiatury, obliczający ich iloczyn i wyprowadzający wynik obliczeń na ekran?

Program, który utworzyliśmy w ćwiczeniach 2. i 3., wyświetlał na ekranie napis – nie używaliśmy w nim zmiennych i nie wykonywaliśmy obliczeń. Struktura kolejnych programów będzie bardziej rozbudowana, m.in. dodamy **zmienne**.

2.1. Nazywanie zmiennych

W wielu językach programowania (np. w języku C++) przed użyciem zmiennej należy ją zadeklarować, czyli określić jej nazwę i typ. W języku Python nie deklaruje się zmiennych. Nie musimy wcześniej określać typu zmiennej, czyli rodzaju przechowywanych w niej danych (np. liczby całkowite, rzeczywiste, znaki).

Zasady dotyczące nazw zmiennych w języku Python

1. Wielkie i małe litery w nazwach są traktowane odmiennie (np. *suma* i *Suma* będą oznaczać różne zmienne). Pisząc program, należy zwracać uwagę na poprawne używanie małych i wielkich liter.
2. W nazwach zmiennych powinno się używać liter, znaku podkreślenia i cyfr. Nazwa nie może zaczynać się od cyfry. Przyjęte jest stosowanie małych liter i niestosowanie polskich liter.
3. W nazwach zmiennych nie wolno stosować spacji. W przypadku nazw kilkuczłonowych zamiast spacji stosujemy znak podkreślenia.
4. Należy nadawać nazwy, które określają znaczenie danej zmiennej, np. *suma*, *liczba_elementow*.

Interpreter określa typ zmiennej na podstawie rodzaju danych, jaki jej przypiszemy. Nie ustanawia jednak typu zmiennej na stałe, ponieważ przy kolejnym przypisaniu może się on zmienić. Na przykład, jeśli na początku programu przypiszemy zmiennej *liczba* wartość całkowitą (34), to jej typ jest całkowity. Jeśli jednak w dalszej części programu przypiszemy jej wartość rzeczywistą (np. 45,8, którą w języku Python zapisujemy z kropką: 45.8), to zmienna *liczba* będzie typu rzeczywistego.

2.2. Nadawanie zmiennym wartości



Zmiennej stosowanej w programie możemy nadać konkretną wartość za pomocą **instrukcji przypisania**. W instrukcji przypisania zmiennej podanej po lewej stronie instrukcji zostanie przypisana obliczona przez komputer wartość wyrażenia znajdującego się po prawej stronie instrukcji.

Instrukcja przypisania	`zmienna = wyrażenie`

Wyrażenie to m.in.: konkretna wartość, zmienna, wyrażenie algebraiczne.

Przykład	Opis
`rok = 2019`	Zmiennej *rok* przypisujemy wartość, która jest liczbą całkowitą, czyli zmienna *rok* będzie typu całkowitego (**`int`**).
`a = 7.58`	Zmiennej *a* przypisujemy wartość, która jest liczbą rzeczywistą, czyli zmienna *a* będzie typu rzeczywistego (**`float`**).
`x = y`	Zmiennej *x* przypisujemy wartość zmiennej *y*.
`obwod = 2 * a + 2 * b`	Zmiennej *obwod* przypisujemy wartość wyrażenia $2 \cdot a + 2 \cdot b$; jeśli wynik obliczeń będzie liczbą całkowitą, interpreter określi dla zmiennej *obwod* typ całkowity; jeśli wynik będzie liczbą dziesiętną, interpreter określi typ zmiennej jako rzeczywisty.
`liczba = 12` `liczba = 35.4`	Zmiennej *liczba* przypisujemy wartość 12 (która jest liczbą całkowitą), a w drugiej instrukcji – wartość 35,4 (która jest liczbą rzeczywistą); poprzednia wartość zmiennej *liczba* zostanie usunięta, a interpreter określi dla zmiennej *liczba* typ rzeczywisty.

Tabela 1. Przykłady stosowania instrukcji przypisania do nadania wartości zmiennej



Jeśli w programie w instrukcji przypisania nadamy zmiennej nową wartość, to poprzednia wartość zostanie usunięta.



Zmiennej stosowanej w programie możemy również nadać wartość za pomocą instrukcji przypisania, wprowadzając wartość z klawiatury w trakcie działania programu. W tym celu stosujemy **instrukcję wejścia** – funkcję `input()`.

Wprowadzanie danych z klawiatury	*zmienna* = input*(tekst_zachęty)*

Argumentem funkcji `input()` (podanym w nawiasach) jest tekst (ciąg znaków, łańcuch). Tekst ten pojawi się na ekranie jako swego rodzaju „zaproszenie" do wpisania ciągu znaków z klawiatury. Jeśli użyjemy funkcji `input()` w instrukcji przypisania, wpisany przez użytkownika ciąg znaków zostanie zapamiętany w zmiennej podanej po lewej stronie.

Aby dana wprowadzona za pomocą instrukcji `input()` została zapamiętana jako liczba, musimy dodać instrukcję, która zamieni ciąg znaków na liczbę: `int()` – w przypadku liczb całkowitych, `float()` – w przypadku liczb rzeczywistych.

Przykład	Opis
`a = input("Wprowadź liczbę: ")`	Jeśli wpiszemy z klawiatury liczbę 346, w zmiennej *a* zostanie zapamiętany ciąg znaków „346", a nie liczba 346.
`a = int(input("Wprowadź liczbę: "))`	Jeśli wpiszemy z klawiatury liczbę 346, w zmiennej *a* zostanie zapamiętana liczba całkowita 346.
`srednia = float(input("Podaj średnią ocen: "))`	Jeśli wpiszemy z klawiatury liczbę 4.3, w zmiennej *srednia* zostanie zapamiętana liczba rzeczywista 4,3.
`miasto = input("Wprowadź nazwę miasta: ")`	Jeśli wpiszemy z klawiatury nazwę miasta „Wrocław", w zmiennej *miasto* zostanie zapamiętany ciąg znaków „Wrocław".

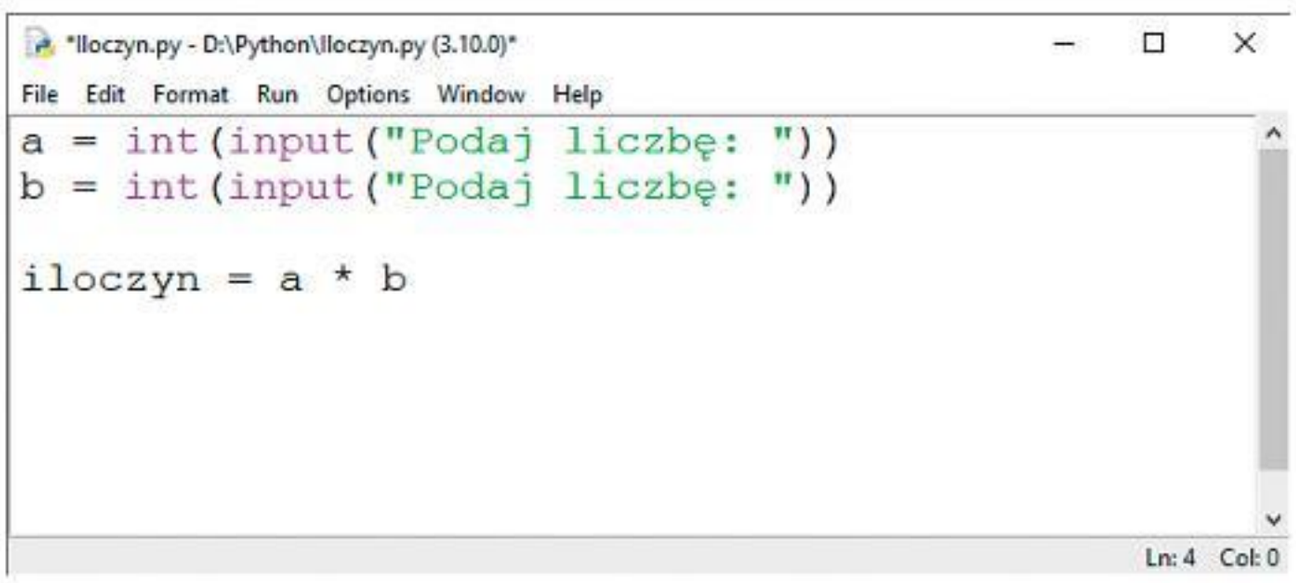
Tabela 2. Przykłady stosowania instrukcji przypisania i funkcji `input()` do wprowadzenia danych z klawiatury

2.3. Wykonywanie obliczeń

W języku Python obliczenia możemy zapisywać, korzystając z instrukcji przypisania. W obliczeniach stosujemy podstawowe **operatory arytmetyczne**: + (dodawania), – odejmowania), * (mnożenia), // (dzielenia całkowitego), / (dzielenia zmiennoprzecinkowego) oraz % (reszty z dzielenia dwóch liczb całkowitych).

Operator	Działanie	Przykład instrukcji przypisania	Wynik działania dla danych: $a = 11$ i $b = 4$
+	dodawanie	`suma = a + b`	11 + 4 = 15
-	odejmowanie	`roznica = a - b`	11 – 4 = 7
*	mnożenie	`iloczyn = a * b`	11 * 4 = 44
//	dzielenie całkowite (z zaokrągleniem części ułamkowej „w dół” – do największej liczby całkowitej mniejszej od wyniku dzielenia)	`iloraz = a // b`	11 // 4 = 2
/	dzielenie zmiennoprzecinkowe (z zachowaniem części ułamkowej)	`iloraz = a / b`	11 / 4 = 2.75
%	obliczenie reszty z dzielenia dwóch liczb całkowitych	`reszta = a % b`	11 % 4 = 3

Tabela 3. Podstawowe operatory arytmetyczne w języku Python



Rys. 7. Program obliczający iloczyn dwóch liczb wprowadzanych z klawiatury – ćwiczenie 4.

Ćwiczenie 4. Tworzymy program z wykorzystaniem zmiennych

1. Na rysunku 7. pokazano program obliczający iloczyn dwóch liczb całkowitych *a* i *b* wprowadzanych z klawiatury.
2. Utwórz nowy plik źródłowy i przepisz program. Wskaż w programie wprowadzanie danych z klawiatury i wykonywanie obliczeń.
3. Zapisz program w pliku pod nazwą *Iloczyn*. Uruchom program.

3. Wyprowadzanie komunikatów i wyników na ekran monitora

W ćwiczeniach 2. i 3. wyprowadzaliśmy komunikaty na ekran monitora, używając polecenia `print()`. Za pomocą tego polecenia można wyprowadzać również wyniki obliczeń.

Wyprowadzanie wyników i komunikatów na ekran monitora	`print(wartość)`

W języku Python `print()` jest **funkcją**, która wyświetla na ekranie wartość zapisaną w nawiasach i umieszcza kursor w nowym wierszu.

Wartość umieszczana wewnątrz nawiasów to np. zmienna, wyrażenie, a także napis. Napis umieszczany w programie, który ma zostać wyświetlony na ekranie monitora, musi być ujęty w górne cudzysłowy podwójne "" lub pojedyncze ''. Funkcja `print()` pozwala łączyć wyświetlanie napisów, wyrażeń i zmiennych. Poszczególne wartości, występujące jako parametry funkcji `print()`, należy rozdzielić przecinkami.

Przykład 1. Wyprowadzanie komunikatów i wyników

```
print("Zaczynamy lekcje z programowania")
print(p)
print(a + b)
print(23 + 89)
print("Obwód =", 2 * a + 2 * b)
print("Obwód wynosi:", obwod)
print("Dla boku o długości", a, "pole kwadratu wynosi:", a * a)
```

Umieszczenie sekwencji znaków `\n` w tekście wyprowadzanym za pomocą instrukcji `print()` wymusza przejście do nowego wiersza.

Instrukcja: `print("Pole prostokąta wynosi:", "\nbok1 * bok2 = ", pole)`

i instrukcje:

```
print("Pole prostokąta wynosi:")
print("bok1 * bok2 =", pole)
```

wyświetlą w taki sam sposób te same wyniki

```
Pole prostokąta wynosi:
bok1 * bok2 = 96
```

.

Ćwiczenie 5. Wyprowadzamy napisy i wynik na ekran

1. Do programu zapisanego w ćwiczeniu 4. dodaj wyświetlanie komunikatu i wyniku, podobnie jak pokazano na rysunku 8a.
2. Zapisz plik pod tą samą nazwą. Uruchom program.
3. Co sądzisz o programach pokazanych na rysunkach 8a i 8b? Wskaż kilka zasadniczych różnic.

Wskazówka: Aby otworzyć ponownie plik (jeśli zamkniemy okno edytora z programem), należy w oknie powłoki Pythona wybrać opcję **File/Open**.

LEPIEJ

Iloczyn.py - D:\Python\Iloczyn.py (3.10.0)

File Edit Format Run Options Window Help

```
a = int(input("Podaj liczbę: "))
b = int(input("Podaj liczbę: "))

iloczyn = a * b

print("Iloczyn wynosi:", iloczyn)
```

Ln: 4 Col: 0

Rys. 8a. Program napisany w uporządkowany sposób – ćwiczenie 5.

GORZEJ

Iloczyn.py - D:\Python\Iloczyn.py (3.10.0)

File Edit Format Run Options Window Help

```
a= int(input("Podaj liczbę: ") )
b =int(input("Podaj liczbę: ") )

iloczyn=a* b

print(  iloczyn)
```

Ln: 7 Col: 0

Rys. 8b. Program napisany niestarannie i wynik nie jest poprzedzony żadnym komentarzem – ćwiczenie 5.

Dobra rada

Mimo że interpreter pomija dodatkowe spacje, kod programu powinien być napisany czytelnie, przejrzyście i jednolicie (rys. 8a). Innej osobie łatwiej będzie zrozumieć program napisany w dobrym stylu.

Uwaga

W większości języków programowania, w tym w języku Python, część dziesiętną liczby zapisujemy po kropce, a nie po przecinku, jak na matematyce. W ten sam sposób liczby dziesiętne są również wyświetlane na ekranie.

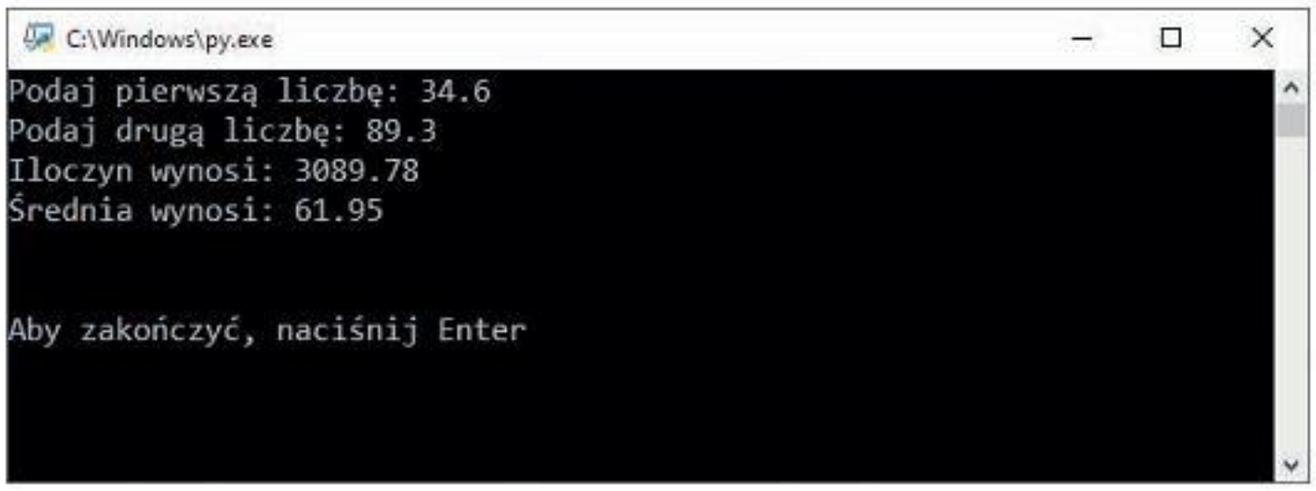
Ćwiczenie 6. Testujemy działanie programu obliczającego średnią arytmetyczną dla danych różnego typu

1. Do programu zapisanego w ćwiczeniu 5. dodaj obliczanie średniej arytmetycznej liczb *a* i *b*. Wynik wyprowadź na ekran. Dodaj odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Srednia_c*. Uruchom program i przetestuj dla par liczb (wartości zmiennych *a* i *b*): (2; 3), (14; 8); (7; 6). Dodatkowo przetestuj program dla pary liczb dziesiętnych (23,3; 12,9). Dlaczego program nie zadziałał?

3. Zmień w instrukcji przypisania typ wprowadzanych danych *a* i *b* na `float`. Zapisz program w pliku pod nazwą *Srednia_r*. Uruchom program i przetestuj dla tych samych danych. Czy teraz program zadziałał za każdym razem?
4. Przetestuj program *Srednia_r* dla liczb: (342,3; 25,7) (3763,82; 109,87).

Wskazówka: Pamiętaj, aby liczby dziesiętne wprowadzać z kropką, a nie z przecinkiem.

Program możemy również uruchomić, klikając dwukrotnie nazwę pliku z zapisanym programem w Eksploratorze plików – otworzy się okno, w którym zobaczymy wynik działania programu (rys. 9.). Aby program został wykonany, niezbędny jest interpreter Pythona.



Rys. 9. Wynik działania programu dla zmiennych typu rzeczywistego – ćwiczenie 6.

Aby okno pokazane na rysunku 9. nie zamknęło się automatycznie po wyświetleniu napisu, możemy napisać na końcu programu instrukcję oczekiwania na naciśnięcie klawisza **Enter**:

```
input("\n\nAby zakończyć, naciśnij Enter")
```

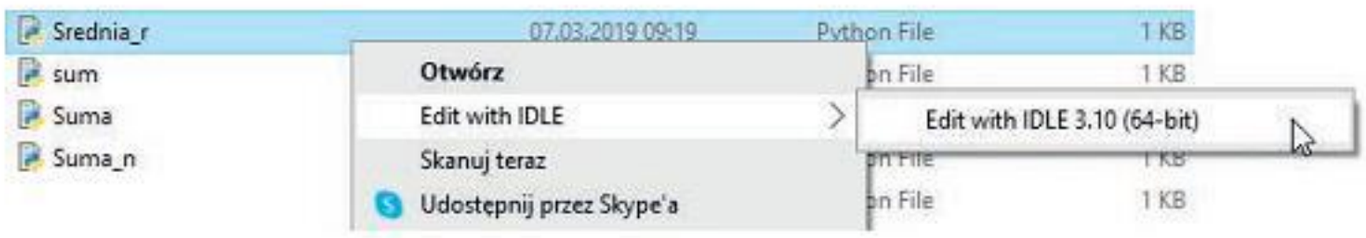
W tym przypadku wartość zwracana przez funkcję `input()` jest pomijana (nie jest przypisywana żadnej zmiennej). Umieszczenie zapisu `\n\n` oznacza wstawienie dwóch pustych wierszy przed komunikatem.



Ćwiczenie 7. Uruchamiamy program, klikając jego nazwę

1. Otwórz plik *Srednia_r* zapisany w ćwiczeniu 6.
2. Dodaj na końcu programu polecenie oczekiwania na naciśnięcie klawisza **Enter**. Zapisz plik pod tą samą nazwą.
3. Zamknij okno edytora i uruchom program, klikając dwukrotnie jego nazwę lub ikonę w oknie Eksploratora plików. Wynik działania programu pokazano na rysunku 9.

Jeśli zamknęliśmy okno powłoki Pythona, a chcemy edytować program, można z menu kontekstowego nazwy pliku w oknie Eksploratora plików wybrać polecenie **Edit with IDLE/ Edit with IDLE 3.10 (64-bit)** – rys. 10.



Rys. 10. Otwieranie okna edytora kodu źródłowego z programem zapisanym w pliku

Warto zapamiętać

- Środowisko programistyczne Pythona (IDLE) udostępnia dwa tryby pracy:
  - interaktywny, w którym możemy napisać jedno polecenie i zatwierdzić je klawiszem **Enter**; jeśli polecenie będzie poprawne, zostanie od razu wykonane,
  - skryptowy, w którym możemy napisać program, zapisać go w pliku, a następnie uruchomić; jeśli program będzie poprawny, zostanie wykonany.
- Aby utworzyć program w języku Python, piszemy kod źródłowy w edytorze kodu źródłowego, zapisujemy program w pliku i uruchamiamy za pomocą interpretera Pythona.
- W języku Python typ wartości, która jest przypisana zmiennej, określa typ tej zmiennej.
- Zmiennej możemy nadać wartość za pomocą instrukcji przypisania. Po lewej stronie znaku = podajemy nazwę zmiennej, a po prawej:
  - konkretną wartość

  lub
  - funkcję `input()`, dzięki której użytkownik wprowadzi tę wartość z klawiatury po uruchomieniu programu

  lub
  - wyrażenie, którego wartość zostanie obliczona w czasie działania programu.

Pytania i polecenia

1. Wyjaśnij, czym jest tryb interaktywny, a czym tryb skryptowy w środowisku Python.
2. Przedstaw i omów na konkretnym przykładzie etapy tworzenia programu komputerowego w języku Python.
3. Przedstaw na przykładzie sposób wprowadzania danych z klawiatury.
4. Co w języku Python określa typ zmiennej?
5. W jaki sposób możemy nadać wartość zmiennej?
6. Do czego służy funkcja `input()`?
7. Omów na przykładzie sposób wyświetlania napisów i wyników na ekranie monitora.
8. Wyjaśnij zapisy:
   a. `bok1 = int(input("podaj długość boku: "))`
   b. `liczba1 = float(input("podaj liczbę: "))`
   c. `iloraz = liczba1 / liczba2`
   d. `print("Iloraz wynosi:", iloraz)`
9. Wyjaśnij, jaka dana zostanie zapamiętana w zmiennej *x* po wykonaniu instrukcji przypisania, gdy z klawiatury wprowadzimy liczbę 258:
   a. `x = int(input("podaj długość"))`
   b. `x = input("podaj długość")`
10. Wskaż nieprawidłowo zapisane instrukcje. Wyjaśnij, na czym polegają błędy:
    a. `liczba1 = input(int("Podaj liczbę: "))`
    b. `y = 29871`
    c. `print("Suma wynosi: ", a + b)`
    d. `iloraz = liczba1 : liczba2`

Zadania

Uwagi:

- W edytorze języka Python można kopiować (**Edit/Copy**) i wklejać (**Edit/Paste**) fragmenty kodu programu – w tym samym dokumencie i pomiędzy dokumentami (tak jak w edytorze tekstu). Ta możliwość jest przydatna, gdy ponownie wykorzystujemy te same lub podobne fragmenty kodu.
- Staraj się pisać przejrzyste programy oraz dodawać w odpowiednich miejscach komentarze.
- Każdy program należy uruchomić i przetestować dla różnych danych, nawet jeśli w zadaniu nie ma takiego polecenia.

1. Napisz specyfikację zadania i program obliczający pole powierzchni prostokąta dla długości boków *a* i *b*, wprowadzanych z klawiatury. Po uruchomieniu programu na ekranie powinny pojawiać się w kolejnych wierszach komunikaty: „Podaj długość boku *a*:", „Podaj długość boku *b*:". Po podaniu danych, w trzecim wierszu powinny wyświetlić się: napis „Pole prostokąta wynosi" oraz wartość pola. Zapisz program w pliku pod nazwą *Pole*.
2. Napisz specyfikację zadania i program realizujący algorytm obliczania kwadratu i sześcianu wprowadzanej z klawiatury liczby rzeczywistej. Wynik i odpowiedni komunikat wyprowadzaj na ekran. Zapisz program w pliku pod nazwą *Potegi*.
3. Napisz specyfikację zadania i program obliczający drogę *S* przebytą w czasie *t* przez pojazd poruszający się ze średnią prędkością *v*. Wynik i odpowiedni komunikat wyprowadzaj na ekran. Zapisz program w pliku pod nazwą *Droga*.
4. Otwórz plik przekazany przez nauczyciela. W pliku zapisano błędny program pokazany na rysunku 11. Po wprowadzeniu z klawiatury liczby program powinien wyświetlić na ekranie liczbę o 2 większą od wprowadzonej. Uruchom program. Jakie błędy pojawiły się w zapisie programu? Zmodyfikuj program, aby działał poprawnie. Zapisz ponownie plik pod tą samą nazwą, uruchom i sprawdź jego działanie.

```
liczba1 = int(input("podaj liczbę: "))
print(liczba2)
liczba2 = liczba1 + 2
```

Rys. 11. Błędny program – zadanie 4.

5. Napisz specyfikację i program do zadania: *Jaki procent liczby* a *stanowi liczba* b*?* Zapisz program w pliku pod nazwą *Procenty*.

Dla zainteresowanych

6. Dane są suma i iloczyn dwóch liczb całkowitych. Napisz specyfikację zadania i program znajdujący te liczby. Zapisz program w pliku pod nazwą *Liczby*.

Temat 67.

Algorytmy z warunkami w języku Python

1. **Instrukcja warunkowa w języku Python**
2. **Stosowanie warunków logicznych**
   - 2.1. Proste warunki logiczne
   - 2.2. Złożone warunki logiczne

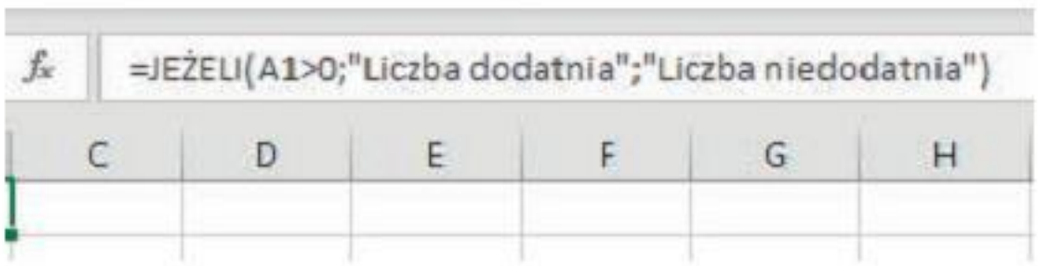
Warto powtórzyć

1. Na czym polega sytuacja warunkowa? Podaj dwa przykłady.
2. Jakie są zasady dotyczące nazw zmiennych w języku Python?
3. W jaki sposób możemy uruchomić program napisany w języku Python?

1. Instrukcja warunkowa w języku Python

Chcemy napisać program sprawdzający, czy wprowadzona z klawiatury liczba jest dodatnia czy niedodatnia (ujemna albo równa zero). Jak zrealizować ten algorytm w arkuszu kalkulacyjnym? W jaki sposób napisać program realizujący sytuację warunkową w języku Python?

Arkusz kalkulacyjny Excel



Scratch



Python

```
if a > 0:
    print("Liczba dodatnia")
else:
    print("Liczba niedodatnia")
```

Rys. 1. Zastosowanie funkcji JEŻELI w arkuszu kalkulacyjnym i instrukcji warunkowej w dydaktycznym środowisku programowania Scratch oraz w języku programowania Python do realizacji tego samego algorytmu z warunkiem

Aby w wybranym języku programowania (w tym w języku Python) napisać program realizujący algorytm z warunkami, korzystamy **z instrukcji warunkowej**, której postać i działanie są zbliżone do funkcji JEŻELI arkusza kalkulacyjnego: JEŻELI(*test_logiczny*;*wartość_jeżeli_prawda*;*wartość_jeżeli_fałsz*).

Ćwiczenie 1. Analizujemy sposób realizacji sytuacji warunkowej w różnych narzędziach

Porównaj sposób zapisu sytuacji warunkowej w różnych narzędziach. Przeanalizuj rysunek 1., zwracając szczególną uwagę na postać instrukcji warunkowej. Wskaż podobieństwa i różnice.

Instrukcja warunkowa	**if** warunek: *lista_instrukcji1* **else:** *lista_instrukcji2* *kolejna_instrukcja*
Uproszczona postać instrukcji warunkowej	**if** warunek: *lista_instrukcji1* *kolejna_instrukcja*

Działanie instrukcji warunkowej jest w większości języków programowania podobne. Sprawdzany jest warunek logiczny (po słowie kluczowym **if**) – jeśli jest prawdziwy, wykonywana jest instrukcja (instrukcje) umieszczona (umieszczone) po warunku (*lista_instrukcji1*); jeśli fałszywy – wykonywana jest instrukcja (instrukcje) po słowie **else** (*lista_instrukcji2*), a jeśli brak tej części instrukcji – wykonywana jest od razu *kolejna_instrukcja* zapisana po instrukcji warunkowej.

Jako *lista_instrukcji1* i *lista_instrukcji2* mogą wystąpić również kolejne instrukcje warunkowe lub **blok instrukcji** (**blok kodu**). W języku Python ważne są wcięcia w programie – instrukcje *lista_instrukcji1* i *lista_instrukcji2* należy przesunąć w prawo przynajmniej o jedną spację (przyjęte jest wcięcie składające się z czterech spacji). Jeśli *lista_instrukcji1* i *lista_instrukcji2* obejmują więcej instrukcji, wszystkie należy tak samo przesunąć. Jest to sposób wyróżnienia **bloku kodu**.

Operator	Określenie	Przykład warunku	Interpretacja wyrażenia
==	równy	a == b	*a* równe *b*
!=	różny	a != b	*a* różne od *b*
<	mniejszy	a < b	*a* mniejsze od *b*
>	większy	a > b	*a* większe od *b*
>=	większy lub równy	a >= b	*a* większe lub równe *b*
<=	mniejszy lub równy	a <= b	*a* mniejsze lub równe *b*

Tabela 1. Podstawowe operatory porównania w języku Python

2. Stosowanie warunków logicznych

Uwaga

Do porównywania wartości używamy operatora porównania ==. Nie stosujemy w tym celu operatora przypisania =.

W instrukcji warunkowej warunek po słowie kluczowym **if** może być:

- prostym warunkiem logicznym, np.:
  `a == 9`

lub

- złożonym warunkiem logicznym, np.:
  `x > -5 and x <= 0`.

2.1. Proste warunki logiczne

Przykład 1. Zapisywanie algorytmu z warunkiem logicznym prostym w języku Python

Zadanie: Oblicz wartość bezwzględną dowolnej liczby całkowitej.
Dane: dowolna liczba całkowita: *x*.
Wynik: wartość bezwzględna liczby *x* równa *w*.

Wartosc_bezwzgl.py - D:/Python/Wartosc_bezwzgl.py (3.10.0)

File Edit Format Run Options Window Help

```
x = int(input("Podaj liczbę x: "))

if x >= 0:
    w = x
else:
    w = -x
print("Wartość bezwzględna |x| =", w)


input("\n\nAby zakończyć, naciśnij Enter")
```

Ln: 14 Col: 0

Ćwiczenie 2. Realizujemy algorytm obliczania wartości bezwzględnej liczby w języku Python

1. Przepisz program podany w przykładzie 1. Sprawdź, czy program jest zgodny ze specyfikacją zadania. Wyjaśnij, która instrukcja zostanie wykonana, gdy warunek będzie prawdziwy, a która – gdy warunek będzie fałszywy.
2. Zapisz program w pliku pod nazwą *Wart_bezwzgl*. Skompiluj program i wykonaj dla kilku różnych wartości *x*.

Ćwiczenie 3. Realizujemy algorytm z warunkiem logicznym prostym w języku Python

1. Napisz specyfikację zadania i program realizujący algorytm sprawdzania, czy wprowadzona z klawiatury liczba jest dodatnia czy niedodatnia. Wykorzystaj zapis instrukcji warunkowej z rysunku 1.
2. Zapisz program w pliku pod nazwą *Liczby* i wykonaj program dla kilku różnych wartości zmiennych.

2.2. Złożone warunki logiczne

Wszystkie klasy twojej szkoły rywalizują w tzw. lidze klas. Zdobycie największej liczby punktów oznacza wygraną. Liczba punktów zależy m.in. od frekwencji i średniej ocen w ostatnim półroczu. W jaki sposób napisać program, w którym będziemy mogli obliczać punkty zależnie od spełnienia obydwu warunków?

Jest to przykład sytuacji warunkowej z warunkiem logicznym złożonym. Jeśli frekwencja i średnia ocen spełniają podane warunki, to klasa uzyskuje punkty. Mamy tu do czynienia z logiczną koniunkcją. W wybranym języku programowania zastosujemy zatem instrukcję warunkową z warunkiem złożonym. W instrukcjach warunkowych warunki złożone zapisujemy, stosując odpowiednie operatory logiczne (tabela 2.).

Operator	Określenie	Przykład warunku logicznego	Interpretacja wyrażenia
`or`	alternatywa logiczna (lub)	`a < -5 or a > 5`	*a* mniejsze od –5 lub *a* większe od 5
`and`	koniunkcja logiczna (i)	`a > 0 and a < 10`	*a* większe od zera i *a* mniejsze od 10

Tabela 2. Operatory logiczne w języku Python

Przykład 2. Zapisywanie algorytmu z warunkiem logicznym złożonym w języku Python

Zadanie: Pewna klasa uczestniczy w lidze klas. Napisz program obliczający, ile wynosi liczba punktów (*pkt*) uzyskanych przez daną klasę zależnie od frekwencji (*f*) i średniej ocen (*so*) w ostatnim półroczu. Klasa otrzymuje dodatkowe 20 punktów, jeśli frekwencja jest powyżej 94% ($f > 94\%$) i średnia ocen nie jest mniejsza niż 4,0 ($so \geq 4{,}0$). Zdobytą wcześniej liczbę punktów (w innych kategoriach), średnią ocen i frekwencję wprowadzaj z klawiatury, a aktualną liczbę punktów wyświetlaj na ekranie.

Dane: liczba całkowita $pkt > 0$, oznaczająca punkty zdobyte przez klasę w innych kategoriach, liczba rzeczywista $f > 0$, oznaczająca procent frekwencji, liczba rzeczywista *so*, oznaczająca średnią ocen.

Wynik: liczba całkowita *pkt*, oznaczająca aktualną liczbę punktów danej klasy.

Opis rozwiązania: W rozwiązaniu zastosujemy instrukcję warunkową (w wersji uproszczonej) z warunkiem złożonym zawierającym operator koniunkcji **`and`**. Zastosujemy wersję uproszczoną, ponieważ niezależnie od tego, czy warunek zostanie spełniony czy nie, na ekranie ma się wyświetlać aktualna liczba punktów.

Liga.py - D:/Python/Liga.py (3.10.0)

File Edit Format Run Options Window Help

```
pkt = int(input("Podaj liczbę punktów zdobytych przez klasę: "))
f = float(input("Podaj frekwencję klasy: "))
so = float(input("Podaj średnią ocen klasy: "))

if f > 94 and so >= 4.0:
    pkt = pkt + 20
print("Aktualna liczba punktów wynosi: " , pkt)


input("\n\nAby zakończyć, naciśnij Enter")
```

Ln: 12 Col: 0

Uwaga

Na razie w programach nie sprawdzamy poprawności danych wprowadzanych przez użytkownika. Zakładamy, że użytkownik podaje poprawne dane. Problem poprawności danych omówimy w temacie 69.

Ćwiczenie 4. Realizujemy algorytm z warunkiem logicznym złożonym w języku Python

1. Przepisz program podany w przykładzie 2. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Dlaczego zastosowano instrukcję warunkową w wersji uproszczonej (bez części z `else`)? Uzasadnij odpowiedź.
2. Wykonaj program dla kilku różnych wartości zmiennych.

Ćwiczenie 5. Realizujemy algorytm z warunkiem logicznym złożonym w języku Python

1. Klasa uczestniczy w lidze klas. Jeśli klasa w ostatnim semestrze uzyskała frekwencję powyżej 93% lub średnią ocen powyżej 4,2, to w nagrodę pojedzie na wycieczkę. Napisz program sprawdzający, czy klasa zakwalifikuje się na wycieczkę. Frekwencję i średnią ocen wprowadzaj z klawiatury, a komunikat „nagroda” lub „brak nagrody” wyświetlaj na ekranie.
2. Zapisz program w pliku pod nazwą *Wycieczka*.

Warto zapamiętać

- W algorytmie z warunkami występują sytuacje warunkowe – wynik lub dalsze działanie algorytmu zależą od spełnienia lub niespełnienia warunku.
- W językach programowania do zapisania algorytmów z warunkami stosujemy instrukcje warunkowe. Działanie tych instrukcji jest podobne w większości języków programowania.
- Warunek w instrukcji warunkowej może być prosty lub złożony. W warunkach złożonych w języku Python stosujemy operatory alternatywy (`or`) i koniunkcji logicznej (`and`).

Pytania i polecenia

1. Wyjaśnij, na czym polega sytuacja warunkowa. Podaj przykład sytuacji warunkowej.
2. W jaki sposób zapisuje się w językach programowania algorytmy zawierające sytuacje warunkowe?
3. Omów działanie instrukcji warunkowej w wybranym języku programowania. Porównaj działanie tej instrukcji do działania funkcji JEŻELI w arkuszu kalkulacyjnym.
4. Podaj dwa przykłady złożonych warunków logicznych.

Zadania

1. Kiedy zostanie wykonana instrukcja `print(x - y)` w programach z podpunktów a i b? Rozważ przypadki: $x > y$, $x = y$, $x < y$. Czym różni się efekt działania instrukcji **`if`** w punktach a i b?

   a.

```
x = int(input("Podaj pierwszą liczbę: "))
y = int(input("Podaj drugą liczbę: "))

if x > y:
    x = x - y
print(x - y)
```

   b.

```
x = int(input("Podaj pierwszą liczbę: "))
y = int(input("Podaj drugą liczbę: "))

if x > y:
    x = x - y
else:
    print(x - y)
```

2. Napisz specyfikację zadania i program realizujący algorytm sprawdzania, czy wprowadzona z klawiatury liczba jest parzysta czy nieparzysta. Zapisz program w pliku pod nazwą *Parzyste_nieparzyste*.
   Wskazówka: Oblicz resztę z dzielenia wprowadzonej liczby przez 2.
3. Napisz specyfikację zadania i program realizujący algorytm sprawdzania, czy liczba jest podzielna przez 3. Zapisz program w pliku pod nazwą *Podzielna_przez_3*.
   Wskazówka: Oblicz resztę z dzielenia wprowadzonej liczby przez 3.
4. Klasa ma być podzielona na dwie grupy, w zależności od wyniku testu kompetencji językowych z wybranego języka obcego lub oceny na świadectwie ukończenia szkoły podstawowej. Jeśli dany uczeń uzyskał z testu wynik powyżej 90% punktów lub jeśli na świadectwie ukończenia szkoły podstawowej miał z danego języka ocenę nie niższą niż 5, to kwalifikuje się do grupy zaawansowanej. W przeciwnym wypadku kwalifikuje się do grupy podstawowej. Napisz program sprawdzający, do której grupy zakwalifikuje się dany uczeń. Liczbę punktów uzyskanych z testu i ocenę ze świadectwa wprowadzaj z klawiatury, a komunikat: „Grupa zaawansowana" lub „Grupa podstawowa" wyświetlaj na ekranie. Zapisz program w pliku pod nazwą *Grupy*.

Dla zainteresowanych

5. Napisz specyfikację zadania i program sprawdzający, czy wprowadzone trzy liczby naturalne są bokami trójkąta prostokątnego. Zależnie od rezultatu program ma wyprowadzać komunikaty „Tak" lub „Nie". Zapisz program w pliku pod nazwą *Trojkat_prostokat*.

Temat 68.

Sprawdzanie poprawności danych i zapisywanie rozwiązania problemu w języku Python

1. Sprawdzanie poprawności danych
2. Zapisywanie rozwiązania przykładowego problemu

Warto powtórzyć

1. W jaki sposób możemy nadać wartości zmiennym w języku Python?
2. Jak działa instrukcja warunkowa w języku Python?
3. Jak wyróżnić blok instrukcji (blok kodu) w języku Python?

1. Sprawdzanie poprawności danych

Chcemy napisać program obliczający pierwiastek kwadratowy danej liczby. Pierwiastki kwadratowe w zbiorze liczb rzeczywistych można jednak obliczyć tylko dla liczb nieujemnych. Jak sprawdzić w programie, czy wprowadzona przez użytkownika liczba jest nieujemna?

Podczas pisania programów ważne jest sprawdzanie poprawności wprowadzanych danych. Dzięki temu jesteśmy zabezpieczeni przed niespodziewanym przerwaniem pracy programu oraz zapewniamy zgodność danych wejściowych ze specyfikacją zadania.

Poprawny program powinien spełniać kilka kryteriów:

- rozwiązywać problem, dla którego został utworzony;
- rozwiązywać problem dla wszystkich danych określonych w specyfikacji i odpowiednio reagować na wprowadzanie niepoprawnych danych;
- rozwiązywać problem w sposób jak najbardziej efektywny, optymalnie wykorzystując zasoby komputera.

Z punktu widzenia wygody obsługi programu wskazane byłoby, aby użytkownik był natychmiast powiadamiany o wprowadzeniu niepoprawnych danych i mógł je od razu poprawić.

Przykład 1. Sprawdzanie poprawności danych

Zadanie: Napisz program obliczający pierwiastek kwadratowy *p* z liczby rzeczywistej nieujemnej *x*, wprowadzanej z klawiatury. Program ma sprawdzać, czy wprowadzona przez użytkownika liczba jest poprawna (nieujemna) i wyprowadzać na ekran wynik obliczenia lub komunikat o błędnych danych.

Dane: liczba rzeczywista *x*.

Wyniki: pierwiastek kwadratowy *p* z liczby *x* lub komunikat „Wprowadzono liczbę mniejszą od zera".

Opis rozwiązania: Aby sprawdzić, czy wprowadzona przez użytkownika liczba spełnia warunki określone w zadaniu, zastosujemy instrukcję warunkową. Pierwiastki kwadratowe w zbiorze liczb rzeczywistych można obliczyć tylko dla liczb nieujemnych. W programie sprawdzamy, czy wprowadzona liczba *x* jest ujemna. Jeśli tak, wyprowadzamy komunikat o błędnych danych, w przeciwnym wypadku wykonujemy obliczenia i wyprowadzamy wynik na ekran.

Wykorzystamy standardową funkcję matematyczną `sqrt()`, służącą do obliczania pierwiastka kwadratowego.

Pierwiastek.py - D:/Dysk C/Python/Pierwiastek.py (3.10.0)

File Edit Format Run Options Window Help

```
from math import sqrt                    #sqrt

x = float(input("Podaj liczbę x: "))

if x < 0:
    print("Wprowadzono liczbę mniejszą od zera")
else:
    p = sqrt(x)
    print( "Pierwiastek z podanej liczby x wynosi: ", p)

input("\n\nAby zakończyć, naciśnij Enter")
```

importowanie z **modułu** `math` (zawierajacego funkcje matematyczne) funkcji `sqrt()` zwracającej pierwiastek kwadratowy z nieujemnej liczby rzeczywistej

wywołanie funkcji `sqrt()`

Ćwiczenie 1. Analizujemy gotowy program

1. Przepisz program podany w przykładzie 1. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Dlaczego zastosowano instrukcję warunkową w pełnej wersji (z **`else`**)?
2. Wykonaj program dla kilku różnych wartości zmiennych.
3. Sprawdź, czy program jest zgodny ze specyfikacją zadania. Wyjaśnij, która instrukcja zostanie wykonana, gdy warunek będzie prawdziwy, a która – gdy warunek będzie fałszywy.

Uwaga

W przykładach i ćwiczeniach dotyczących obliczeń nie uwzględniamy przypadków wprowadzenia danych innych niż liczba. Takie przypadki omówimy na informatyce realizowanej w zakresie rozszerzonym.

Ćwiczenie 2. Sprawdzamy poprawność wprowadzanych danych liczbowych

1. Napisz specyfikację zadania i program obliczający pole trójkąta. W programie uwzględnij sprawdzanie poprawności wprowadzanych danych (podstawy i wysokości trójkąta).
2. Zapisz program w pliku pod nazwą *Pole_trojkata*.

2. Zapisywanie rozwiązania przykładowego problemu

Chcesz przygotować tradycyjną sałatkę jarzynową na imprezę urodzinową. W Internecie są przepisy, ale w proporcjach dla dwóch, pięciu lub dziesięciu osób. Nie możesz znaleźć przepisu dla trzynastu osób – uczestników spotkania. Jak napisać program w języku Python, który policzy, ile potrzeba danego składnika sałatki w zależności od liczby gości, i wyświetli wynik na ekranie?

Przykład 2. Sformułowanie zadania, specyfikacja i projekt rozwiązania

Zadanie: Napisz program obliczający, ile danego składnika sałatki potrzeba dla *o2* osób, gdy znasz ilość składnika dla *o1* osób.

Dane: liczba całkowita $o1 > 0$, oznaczająca liczbę osób uwzględnioną w przepisie, liczba rzeczywista $s1 > 0$, określająca ilość składnika dla *o1* osób, liczba całkowita $o2 > 0$, oznaczająca liczbę uczestników spotkania.

Wynik: liczba rzeczywista $s2 > 0$, oznaczająca ilość składnika dla *o2* osób.

Opis rozwiązania: W rozwiązaniu skorzystamy z zależności:

jeśli

$s1 \longrightarrow$ dla *o1* osób
$s2 \longrightarrow$ dla *o2* osób

to

$$s2 = \frac{s1 \cdot o2}{o1}$$

Aby zapisać te obliczenia w języku Python, zastosujemy instrukcję przypisania:

```
s2 = (s1 * o2) / o1
```



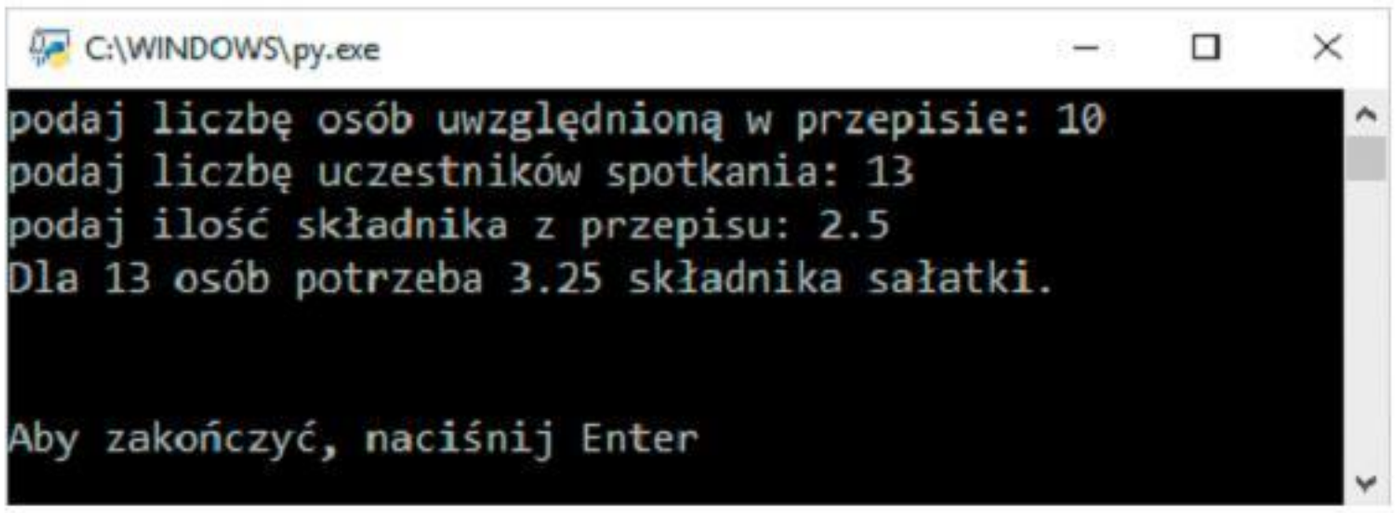
Uwaga: Z matematycznego punktu widzenia nawiasy są zbędne – umieściliśmy je w programie dla zwiększenia czytelności zapisu.

Ćwiczenie 3. Realizujemy algorytm obliczania ilości składnika sałatki w języku Python

1. Utwórz program realizujący zadanie sformułowane w przykładzie 2. Dodaj komunikaty dotyczące wprowadzania zmiennych oraz komunikat przed wyprowadzeniem wyniku. Komunikaty i wynik wyprowadź na ekran monitora, tak jak pokazano na rysunku 1.
2. Zapisz program w pliku pod nazwą *Przepis*. Uruchom program i przetestuj dla następujących trójek liczb (wartości zmiennych *o1*, *o2*, *s1*): (10; 13; 2,5), (3; 12; 0,45), (15; 4; 3,20).
3. Przetestuj program dla trzech innych trójek liczb.

Wskazówka: Aby wyświetlić wyniki, jak pokazano na rysunku 1., należy wpisać instrukcję:

```
print("Dla ", o2, "osób potrzeba", s2, "składnika sałatki.")
```



Rys. 1. Wynik działania programu – ćwiczenie 3.

Zauważmy, że program utworzony w ćwiczeniu 3. oblicza ilość tylko jednego składnika sałatki. Jeśli chcemy obliczyć ilość kolejnego składnika, musimy ponownie uruchomić program. W temacie 70. zmodyfikujemy ten program, aby obliczał i wyświetlał na ekranie kolejno ilości wszystkich składników.

Ćwiczenie 4. Dodajemy do programu sprawdzanie poprawności danych

1. Otwórz plik *Przepis* zapisany w ćwiczeniu 3. Dodaj do programu sprawdzanie poprawności wprowadzanych danych, zgodnie ze specyfikacją.
2. Zapisz plik pod tą samą nazwą.

Warto zapamiętać

- Program powinien efektywnie rozwiązywać problem dla wszystkich danych określonych w specyfikacji.
- Należy dbać o zgodność danych wejściowych ze specyfikacją i sprawdzać w programie, czy użytkownik wprowadził właściwe dane.
- Do sprawdzenia poprawności danych możemy zastosować w programie instrukcję warunkową.

Pytania i polecenia

1. Dlaczego ważne jest sprawdzanie poprawności danych w programie? Uzasadnij odpowiedź.
2. Jakie kryteria powinien spełniać poprawny program?
3. Podaj przykład sytuacji, w której należy sprawdzić poprawność danych.
4. Jakiej instrukcji możemy użyć do sprawdzenia poprawności danych w programie napisanym w języku Python?

Zadania

1. Napisz specyfikację zadania i program obliczający wysokość trójkąta, gdy podane są jego pole i podstawa. Zadbaj o poprawność wprowadzanych danych, zgodnie z zapisaną wcześniej specyfikacją. Zapisz program w pliku pod nazwą *Wysokosc_trojkata*.
2. Napisz specyfikację zadania i program realizujący algorytm obliczania objętości walca o wysokości *h* i promieniu podstawy *r*, gdzie *h* i *r* są dodatnimi liczbami rzeczywistymi. Zadbaj o poprawność wprowadzanych danych, zgodnie z zapisaną wcześniej specyfikacją. Dla liczb niedodatnich powinien być wyprowadzany napis „błędne dane". Zapisz program w pliku pod nazwą *Objetosc_walca*.
3. Napisz specyfikację zadania i program obliczający, ile litrów wody spadło na plac o powierzchni *P* metrów kwadratowych, jeśli pokryła go warstwa wody o grubości *d* milimetrów. Zadbaj o poprawność wprowadzanych danych, zgodnie z zapisaną wcześniej specyfikacją. Zapisz program w pliku pod nazwą *Deszcz*.
4. Napisz specyfikację zadania i opisz rozwiązanie: *Dany jest średni wiek trzech osób: S. Najmłodsza osoba ma* x *lat, najstarsza:* y. *Ile lat ma trzecia osoba?* S, x, y *to liczby naturalne różne od zera*. Napisz program na podstawie zapisanej specyfikacji. Przetestuj program dla następujących wartości zmiennych (*S*, *x*, *y*): (15, 10, 20), (27, 20, 40), (24, 14, 33). Dodaj kilka swoich propozycji danych. Zadbaj o poprawność wprowadzanych danych, zgodnie ze specyfikacją. Jakie powinny być wartości zmiennych: *S* oraz *x* i *y*, aby otrzymany wynik był sensowny? Uwzględnij te warunki w opisie algorytmu. Zapisz program w pliku pod nazwą *Wiek_3_osoby*.
5. Opisz wymyślony samodzielnie problem podobnie jak w punkcie 2. tego tematu. Ułóż zadanie, zapisz specyfikację i przygotuj rozwiązanie podobnie jak w przykładzie 2. Na koniec napisz program realizujący to zadanie. Zapisz program w pliku pod nazwą *Problem*.

Dla zainteresowanych

6. Napisz specyfikację i program do zadania: *Dane są: suma długości przekątnych rombu i długość jednego boku. Oblicz długości przekątnych rombu*. Zadbaj o poprawność wprowadzanych danych, zgodnie ze specyfikacją. Zapisz program w pliku pod nazwą *Romb*.
7. Napisz program obliczający pierwiastki równania kwadratowego $ax^2 + bx + c = 0$. Program powinien poprawnie reagować na błędne dane ($a = 0$) oraz rozróżniać sytuacje: $\Delta = 0$ i $\Delta > 0$. Zapisz program w pliku pod nazwą *Rownanie_kwadratowe*.

Temat 69.

Instrukcje warunkowe zagnieżdżone i z klauzulą `elif` w języku Python

1. Algorytmy z warunkami zagnieżdżonymi
2. Stosowanie instrukcji wyboru

Warto powtórzyć

1. Jak działa instrukcja iteracyjna?
2. Jak wyróżnić blok instrukcji (blok kodu) w języku Python?
3. Dla jakiej wartości warunku logicznego zostaną wykonane instrukcje umieszczone po słowie w instrukcji warunkowej **`else`**?

1. Algorytmy z warunkami zagnieżdżonymi

Chcemy napisać program realizujący algorytm wyboru najmniejszej z trzech różnych liczb całkowitych wprowadzanych z klawiatury. W jaki sposób po sprawdzeniu, czy pierwsza liczba jest mniejsza od drugiej, sprawdzać, czy pierwsza jest też mniejsza od trzeciej oraz druga od trzeciej?

Instrukcje warunkowe, podobnie jak funkcję JEŻELI w arkuszu kalkulacyjnym, można **zagnieżdżać**, czyli po warunku oraz po słowie **`else`** może wystąpić kolejna instrukcja warunkowa.

Ćwiczenie 1. Piszemy specyfikację problemu

Sformułuj zadanie i napisz specyfikację zadania do sytuacji problemowej podanej na początku tematu. Przyjmij następujące nazwy zmiennych: *a*, *b*, *c* (dla sprawdzanych liczb), *najmniejsza* (dla zapamiętania najmniejszej liczby).

Napiszemy program realizujący algorytm wyboru najmniejszej z trzech różnych liczb całkowitych. W rozwiązaniu zastosujemy zagnieżdżone instrukcje warunkowe.

*T69_c2_Najmniejsza_z_trzech_R.py - D:/Dysk C/Python/T69_c2_Najmniejsza_z_trzech_R.py (3....

File Edit Format Run Options Window Help

```
a = int(input("Podaj pierwszą liczbę: "))
b = int(input("Podaj drugą liczbę: "))
c = int(input("Podaj trzecią liczbę: "))
if a < b:
    if a < c:
        najmniejsza = a
    else:
        najmniejsza = c
else:
    if b < c:
        najmniejsza = b
    else:
        najmniejsza = c
print("Najmniejsza liczba wynosi:", najmniejsza)

input("\n\nAby zakończyć, naciśnij Enter")
```

Ln: 17 Col: 0

Rys. 1. Zapisywanie algorytmu z warunkami zagnieżdżonymi w języku Python

Ćwiczenie 2. Stosujemy zagnieżdżone instrukcje warunkowe

1. Przepisz program pokazany na rysunku 1. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Sprawdź, czy program jest zgodny ze specyfikacją zadania napisaną w ćwiczeniu 1. Jeśli nie, popraw program. Zwróć uwagę na zapis zagnieżdżonych instrukcji warunkowych. Dlaczego w tym programie je zastosowano?
2. Zapisz program w pliku pod nazwą *Najmniejsza_z_trzech*. Uruchom program i przetestuj dla następujących trójek liczb (wartości zmiennych *a*, *b*, *c*): (3; -56; 0), (76; 123; -2), (-70; 40; -40), (56; 68; 88), (100; 0; 2).
3. Wyjaśnij, co dzieje się w poszczególnych wierszach programu. W zeszycie przedmiotowym dokończ opis: *Sprawdzamy, czy pierwsza liczba jest mniejsza od drugiej. Jeśli tak, to…*
4. Odpowiedz na pytanie: *Kiedy zostanie wykonane przypisanie* `najmniejsza = b`*?*

Ćwiczenie 3. Zapisujemy problem sprzedawcy komputerów w języku programowania

1. Przypomnij sobie problem sprzedawcy komputerów i, korzystając ze specyfikacji zadania i projektu rozwiązania z tematu 52. (przykłady 3-5), napisz program realizujący przedstawione rozwiązanie. Zadbaj o sprawdzanie poprawności wprowadzanych danych (zgodnie ze specyfikacją).
2. Zapisz program w pliku pod nazwą *Cena_komputera*, uruchom i przetestuj dla różnych wartości zmiennych (*w*, *cenak*, *cenap*).

2. Stosowanie instrukcji wyboru

Chcemy napisać program „Prosty kalkulator" umożliwiający wybór rodzaju obliczenia z wykorzystaniem menu – aby po wprowadzeniu z klawiatury np. liczby 1 została obliczona suma dwóch liczb, a po wprowadzeniu liczby 2 – różnica itd. Jak to zrobić w języku Python?

Jeśli chcemy utworzyć menu ekranowe, w którym użytkownik ma wybrać jedną z czterech opcji, możemy zastosować kilkakrotnie instrukcję warunkową **`if`**. W programie pokazanym na rysunku 2. wybór rodzaju obliczeń zależy od spełnienia warunku (od wprowadzonej wartości zmiennej *opcja*).

```
opcja = int(input("Podaj opcję menu: 1 - suma, 2 - różnica,"
        "3 - iloczyn, 4 - iloraz: "))
if opcja == 1:
    print("suma =", x + y)
if opcja == 2:
    print("różnica =", x - y)
if opcja == 3:
    print("iloczyn =", x * y)
if opcja == 4:
    print("iloraz = ", x / y)
if opcja != 1 and opcja != 2 and opcja != 3 and opcja != 4:
    print("Nie ma takiej opcji")
```

Rys. 2. Stosowanie instrukcji warunkowej **`if`** do wykonania menu programu

Do tworzenia menu ekranowego, w którym użytkownik musi wybrać jedną z opcji, w języku Python można, zamiast kilku instrukcji **`if`**, użyć instrukcji **`if`** z klauzulą **`elif`**.

```
opcja = int(input("Podaj opcję menu: 1 - suma, 2 - różnica"
        "3 - iloczyn, 4 - iloraz: "))
if opcja == 1:
    print("suma =", x + y)
elif opcja == 2:
    print("różnica =", x - y)
elif opcja == 3:
    print("iloczyn =", x * y)
elif opcja == 4:
    print("iloraz = ", x / y)
else:
    print("Nie ma takiej opcji")
```

Rys. 3. Stosowanie instrukcji wyboru **`if`** z klauzulą **`elif`** do wykonania menu programu

Instrukcja `if` z klauzulą `elif`	**if** *warunek1*: *instrukcja1* **elif** *warunek2*: *instrukcja2* **elif** *warunek3*: *instrukcja3* … **else**: *instrukcjan* *kolejna_instrukcja*

Najpierw jest sprawdzany *warunek1*. Jeśli *warunek1* jest prawdziwy, wykonywana jest *instrukcja1* i następuje wyjście z instrukcji. Jeśli *warunek1* nie jest prawdziwy, sprawdzany jest *warunek2* itd. W instrukcji **`if`** można umieścić dowolną liczbę klauzul **`elif`**. Jeśli żaden z tych warunków nie jest spełniony, zostanie wykonana *instrukcjan* znajdująca się po opcjonalnym słowie **`else`**, a następnie program przejdzie do *kolejnej_instrukcji*. Jeśli nie ma tego słowa, program przejdzie do *kolejnej_instrukcji*.

Ćwiczenie 4. Stosujemy instrukcję `if` z klauzulą `else` do utworzenia menu programu

1. Otwórz plik przekazany przez nauczyciela. W pliku zapisano rozwiązanie problemu „Prosty kalkulator". Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Sprawdź działanie programu dla wszystkich opcji (również podając inne liczby niż 1, 2, 3 lub 4).
2. Zmodyfikuj program, stosując instrukcję `if` z klauzulą `elif` (wzór na rysunku 2.). Zapisz program w pliku pod nazwą *Prosty _kalkulator_elif*.
3. Porównaj obydwa rozwiązania (z kilkoma instrukcjami `if` oraz z instrukcją `if` z klauzulą `elif`). Wskaż różnice.

Warto zapamiętać

- W programach komputerowych można stosować zagnieżdżone instrukcje warunkowe.
- W przypadku gdy wykonywanie operacji jest uzależnione od różnych wartości zmiennej, w języku Python można zastosować instrukcję `if` z klauzulą `elif`.
- W instrukcji `if` z klauzulą `elif` wykonywana jest jedna z instrukcji, dla której umieszczony przed nią warunek jest prawdziwy. Jeśli żaden z tych warunków nie jest spełniony, zostanie wykonana instrukcja znajdująca się po `else`.
- Instrukcja wyboru jest przydatna, np. do tworzenia menu programu, w którym musimy wybrać jedną z opcji.

Pytania i polecenia

1. Podaj przykład sytuacji problemowej, w której występują zagnieżdżające się sytuacje warunkowe.
2. Omów sposób zapisywania zagnieżdżonej instrukcji warunkowej.
3. Podaj przykład problemu, dla którego możemy zastosować instrukcję `if` z klauzulą `elif`.
4. Na czym polega działanie instrukcji `if` z klauzulą `elif`?

Zadania

1. Napisz specyfikację zadania i program realizujący algorytm sprawdzania, która z dwóch liczb rzeczywistych jest większa. Uwzględniaj przypadek, gdy liczby są równe. Zapisz program w pliku pod nazwą *Wieksza_z_dwoch*.
2. Napisz specyfikację zadania: *Wprowadź dowolną liczbę rzeczywistą. Gdy liczba jest dodatnia, obliczaj pole koła o promieniu równym tej liczbie. Dla liczby ujemnej podawaj jej wartość bezwzględną, a dla równej zero – wyprowadź napis „równa zero"*. Napisz program na podstawie zapisanej specyfikacji. Zapisz program w pliku pod nazwą *Zadanie2*.

3. Napisz specyfikację zadania i program obliczający prędkość samochodu *v*, gdy dane są droga *s* i czas *t*, podane odpowiednio w km i h. Jeśli prędkość samochodu przekracza 90 km/h, wyświetlaj komunikat „za szybko". Jeśli prędkość będzie równa 90 km/h, wyświetlaj komunikat „w sam raz", a jeśli mniejsza, wyświetlaj komunikat „za wolno". Program napisz na podstawie przygotowanej wcześniej specyfikacji. Zapisz program w pliku pod nazwą *Predkosc*.
4. Napisz program, który dla wprowadzonej liczby 1 oblicza pole kwadratu, dla 2 – pole prostokąta, dla 3 – pole trójkąta, dla 4 – pole trapezu. Dla innej liczby – komunikat „Nie ma takiej opcji". Zapisz program w pliku pod nazwą *Pola_figur*.
5. Napisz program obliczający wartość funkcji przedstawionej na rysunku 4. Zapisz program w pliku pod nazwą *Wartosc_funkcji*.

$$f(x) = \begin{cases} -1 & dla\ x < -1 \\ x & dla\ -1 \leq x \leq 1 \\ 1 & dla\ x > 1 \end{cases}$$

Rys. 4. Wzór funkcji – zadanie 5.

Dla zainteresowanych

6. Napisz specyfikację zadania i program sprawdzający, czy z odcinków o długościach wprowadzonych przez użytkownika z klawiatury można zbudować trójkąt (przypomnij sobie warunek istnienia trójkąta z lekcji matematyki). Program ma sprawdzać dziesięć trójek liczb i zależnie od rezultatu wyprowadzać komunikaty: „można zbudować trójkąt" lub „nie można zbudować trójkąta". Dodaj do programu sprawdzanie poprawności wprowadzanych danych. Zapisz program w pliku pod nazwą *Trojkat*.
7. Otwórz plik *Najmniejsza_z_trzech* zapisany w ćwiczeniu 2. Zmodyfikuj program, aby był uwzględniony przypadek, gdy liczby są równe. Zapisz plik pod tą samą nazwą.
8. Wymyśl samodzielnie problem, w którym można wykorzystać instrukcję `if` z klauzulą `elif` do utworzenia menu. Napisz program w języku Python realizujący ten problem.

Temat 70. Algorytmy iteracyjne w języku Python

1. Instrukcja iteracyjna `for` w języku Python
2. Zapisywanie rozwiązania problemu iteracyjnego w języku Python

Warto powtórzyć

1. Czym jest iteracja? Podaj przykład problemu, w którym występuje iteracja.
2. Jak działa instrukcja iteracyjna w wybranym dydaktycznym środowisku programowania (Baltie, Scratch)?
3. Czym jest blok instrukcji (blok kodu) i jak go wyróżniamy?
4. Jak działa iteracyjny algorytm sumowania *n* liczb?

1. Instrukcja iteracyjna `for` w języku Python

Chcemy napisać program obliczający sumę *n* liczb. W algorytmie mamy do czynienia z powtarzaniem poleceń wprowadzania liczby i dodawania. W jaki sposób napisać program realizujący iterację w języku Python?

W temacie 53. omówiliśmy iterację na przykładzie algorytmu sumowania *n* liczb. Zapisaliśmy specyfikację zadania i listę kroków tego algorytmu. W tym temacie zapiszemy go w postaci programu w języku Python.

Aby w języku programowania napisać program realizujący algorytm iteracyjny, stosujemy **instrukcje iteracyjne** (zwane też **instrukcjami pętli**).

W języku Python są dostępne dwie instrukcje iteracyjne: **`for`** i **`while`**. W tym temacie pokażemy stosowanie instrukcji **`for`** (stosowanie instrukcji **`while`** omówimy w temacie 72.).

Instrukcję **`for`** możemy stosować, gdy liczba powtórzeń (iteracji) jest z góry określona lub wprowadzana z klawiatury po uruchomieniu programu.

Instrukcja iteracyjna `for`	`for` *zmienna* `in` *lista_wartości*: *lista_instrukcji*

Jako *lista_instrukcji* może wystąpić pojedyncza instrukcja (w tym instrukcja pętli) lub więcej instrukcji (blok instrukcji). *Lista_instrukcji* musi być przesunięta w prawo przynajmniej o jedną spację (przyjęte jest wcięcie składające się z czterech spacji).



W języku Python liczbę iteracji w instrukcji **`for`** określa długość *listy_wartości* po słowie **`in`**. *Lista_instrukcji* zostanie wykonana dla wszystkich wartości z *listy_wartości*. *Listę_wartości* możemy zapisać w różny sposób.



Przykład 1. Stosowanie instrukcji **`for`** w języku Python

Podajemy przykłady określania *listy_wartości* w instrukcji **`for`**:

```
for i in [0, 1, 2, 3, 4, 5]:
    print(i)
```

Instrukcja `print(i)` zostanie wykonana sześć razy dla listy wartości [0, 1, 2, 3, 4, 5]; zmienna *i* będzie przyjmować kolejne wartości z tej listy, czyli: 0, 1, 2, 3, 4, 5.

```
for i in [2, 4, 6, 8, 10]:
    print(i)
```

Instrukcja `print(i)` zostanie wykonana pięć razy dla listy [2, 4, 6, 8, 10]; zmienna *i* będzie przyjmować kolejne wartości z tej listy, czyli: 2, 4, 6, 8, 10.

Nie zawsze chcemy wpisywać wszystkie wartości listy, zwłaszcza dla dużej liczby iteracji. Do utworzenia *listy_wartości* można użyć funkcji `range()`, która tworzy sekwencję wartości całkowitych. Argumentami funkcji `range()` mogą być konkretne wartości lub zmienne, np. `range(10)`, `range(n)`. Wartość zmiennej *n* możemy wprowadzać z klawiatury.



Przykład 2. Stosowanie funkcji `range()` do określania *listy_wartości* w instrukcji **`for`**

- Dla jednego argumentu: `range(`*koniec*`)`

```
for i in range(7):
    a = int(input("Podaj liczbę: "))
```

Instrukcja `a = int(input("Podaj liczbę: "))` zostanie wykonana siedem razy. Funkcja `range()` wygeneruje kolejne liczby całkowite z przedziału ⟨0, *koniec*), czyli zmienna *i* będzie przyjmować kolejno wartości: 0, 1, 2, 3, 4, 5, 6.

```
for i in range(10):
    print("Witaj")
```

Instrukcja `print("Witaj")` zostanie wykonana dziesięć razy. Funkcja `range()` wygeneruje kolejne liczby całkowite z przedziału ⟨0, *koniec*), czyli zmienna *i* będzie przyjmować kolejno wartości: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

- Dla dwóch argumentów: `range(`*początek, koniec*`)`

```
for i in range(18, 95):
    print(i)
```

Instrukcja `print(i)` zostanie wykonana siedemdziesiąt siedem (95 – 18 = 77) razy. Funkcja `range()` wygeneruje kolejne liczby całkowite z przedziału ⟨*początek*, *koniec*), czyli zmienna *i* będzie przyjmować kolejno wartości: 18, 19, 20, …, 94.

- Dla trzech argumentów: `range(początek, koniec, krok)`

```
for i in range(2, 102, 2):
    print(i)
```

Instrukcja `print(i)` zostanie wykonana pięćdziesiąt razy. Funkcja `range()` wygeneruje kolejne liczby całkowite z przedziału ⟨*początek*, *koniec*), zmieniające się o *krok*, czyli zmienna *i* będzie przyjmować kolejno wartości: 2, 4, 6, …, 100. Trzeci argument (*krok*) określa tym samym, o jaką wartość zmienia się zmienna *i*.

```
for i in range(10, 0, -1):
    print(i)
```

Instrukcja `print(i)` zostanie wykonana dziesięć razy. *Krok* wynosi -1, czyli zmienna *i* będzie przyjmować kolejno wartości: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.

Ćwiczenie 1. Stosujemy instrukcję `for` w języku Python

Korzystając z przykładów 1. i 2., napisz pięć programów:

a. Program wprowadzający 10 liczb rzeczywistych z klawiatury i zapamiętujący je kolejno w zmiennej *liczba*.
b. Program wyświetlający na ekranie w kolejnych wierszach *n* napisów „Lubię informatykę" (wartość zmiennej *n* wprowadzaj z klawiatury).
c. Program wyświetlający w kolumnie liczby całkowite od 1 do *k* (wartość zmiennej *k* wprowadzaj z klawiatury).
d. Program wyświetlający w kolumnie liczby całkowite nieparzyste od 1 do 33.
e. Program wyświetlający w kolumnie liczby całkowite od 10 do -10.

Każdy program zapisz w pliku i uruchom.

2. Zapisywanie rozwiązania problemu iteracyjnego w języku Python

Przykład 3. Zapisywanie algorytmu iteracyjnego sumowania liczb w języku Python

Zadanie: Oblicz sumę *n* liczb całkowitych.

Dane: liczba naturalna *n* większa od zera, oznaczająca, ile liczb ma być sumowanych, *n* dowolnych liczb całkowitych.

Wynik: wartość sumy liczb: *suma*.

Suma_n.py - D:/Python/Suma_n.py (3.10.0)

File Edit Format Run Options Window Help

```
n = int(input("Ile liczb: "))

suma = 0
for i in range(n):
    a = int(input("Wprowadź liczbę: "))
    suma += a
print("Suma wynosi:", suma)


input("\n\nAby zakończyć, naciśnij Enter")
```

W języku Python instrukcję przypisania, w której danej zmiennej przypisujemy jej poprzednią wartość powiększoną o wartość innej zmiennej, np. `suma = suma + a`, można zapisać krócej: `suma += a`. W podobny sposób można zapisać instrukcje przypisania z użyciem pozostałych operatorów (tabela 1.).

Przykład zastosowania	Odpowiednik	Znaczenie
`s += a`	`s = s + a`	przypisanie zmiennej *s* poprzedniej wartości tej zmiennej, powiększonej o *a*
`s -= a`	`s = s - a`	przypisanie zmiennej *s* poprzedniej wartości tej zmiennej, pomniejszonej o *a*
`s *= a`	`s = s * a`	przypisanie zmiennej *s* poprzedniej wartości tej zmiennej, pomnożonej *a* razy
`s /= a` `s //= a`	`s = s / a` `s = s // a`	przypisanie zmiennej *s* poprzedniej wartości tej zmiennej, podzielonej przez *a*

Tabela 1. Operatory przypisania w języku Python

Ćwiczenie 2. Zapisujemy algorytm sumowania *n* liczb w języku Python

1. Przypomnij listę kroków algorytmu sumowania *n* liczb z tematu 53.
2. Otwórz plik przekazany przez nauczyciela. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Sprawdź, czy program jest zgodny ze specyfikacją zadania i listą kroków. Uruchom i przetestuj program dla kilku różnych wartości zmiennych.

Ćwiczenie 3. Dodajemy do programu sprawdzanie poprawności danych

1. Dodaj do programu z ćwiczenia 2. sprawdzanie poprawności wprowadzanej wartości zmiennej *n*, zgodnie ze specyfikacją zadania (podaną w przykładzie 3.). Sumowanie wprowadzanych liczb powinno być wykonywane dla poprawnej wartości *n*. Jeśli użytkownik wprowadzi błędne dane, powinien wyświetlić się komunikat „Niepoprawne dane".
2. Zapisz program w pliku pod nazwą *Suma_n* i uruchom program dla różnych wartości zmiennej *n*.

Ćwiczenie 4. Stosujemy instrukcję iteracyjną `for`

1. Napisz specyfikację zadania i program obliczający sumę *n* kolejnych liczb naturalnych (począwszy od 1). Sprawdzaj poprawność wprowadzanej wartości zmiennej *n*.
2. Zapisz program w pliku pod nazwą *Naturalne_suma_n*, uruchom i sprawdź jego działanie dla różnych wartości zmiennej *n*.

Wskazówka: Możesz zmodyfikować kod programu z ćwiczenia 3.

Warto zapamiętać

- W językach programowania dostępne są instrukcje iteracyjne (instrukcje pętli), z których korzystamy, aby zapisać algorytm iteracyjny w postaci programu.
- Jeśli instrukcje wewnątrz pętli mają zostać wykonane zadaną liczbę razy, stosujemy pętlę **`for`**.
- W języku Python liczbę iteracji określa długość *listy_wartości* po słowie **`in`** w instrukcji **`for`**.

Pytania i polecenia

1. Wyjaśnij na przykładach stosowanie instrukcji iteracyjnej **`for`** w języku Python.
2. W jaki sposób określa się liczbę iteracji w instrukcji **`for`** w języku Python?
3. Omów zastosowanie iteracji na przykładzie sumowania ośmiu liczb.
4. Na podstawie następujących fragmentów programów napisz, jakie są wyniki ich działania:

   a.

```
for i in range(10):
    print("$", end = "")
```

   b.

```
for i in range(10):
    print("$")
```

5. Ile razy będzie wykonana instrukcja `print(i)`? Dla jakich wartości zmiennej *i* zostanie wykonana instrukcja `print(i)`?

```
for i in range(9, 51):
    print(i)
```

Zadania

1. Zmodyfikuj program *Przepis* zapisany w ćwiczeniu 4. z tematu 68., aby obliczenia wykonywane były dla każdego z *n* składników sałatki, gdzie *n* jest wprowadzane z klawiatury. Zapisz plik pod tą samą nazwą.
2. Napisz program zliczający, ile jest liczb parzystych wśród dziesięciu liczb wprowadzanych z klawiatury i na koniec wyprowadzający ich liczbę. Zapisz program w pliku pod nazwą *Parzyste*.
3. Napisz specyfikację zadania, utwórz listę kroków oraz napisz program obliczania sumy liczb podzielnych przez 7 dla *n* liczb wprowadzanych z klawiatury. Zapisz program w pliku pod nazwą *Podzielne_7_suma*.

4. Napisz program realizujący algorytm obliczania iloczynu *n* dowolnych liczb całkowitych. Wynik mnożenia zapamiętuj w zmiennej *iloczyn* i wyprowadź na ekran. Zapisz program w pliku pod nazwą *Iloczyn_n*.
   Wskazówka: Zmiennej *iloczyn* na początku programu przypisz wartość 1.
5. Napisz specyfikację zadania i program obliczający osobno sumę liczb dodatnich i sumę liczb ujemnych dla *n* liczb całkowitych wprowadzanych z klawiatury. Zapisz program w pliku pod nazwą *Dodatnie_i_ujemne*.
6. W firmie X miesięczna płaca podstawowa jest zwiększana m.in. o kwotę za przepracowane nadgodziny. Jeśli liczba nadgodzin przekroczy 30, to stawka za kolejne nadgodziny jest zwiększana o 50%. Wprowadzaj liczbę nadgodzin przepracowanych przez jednego pracownika oraz stawkę za jedną nadgodzinę. Oblicz i wyprowadź płacę za przepracowane nadgodziny dla *n* pracowników. Program napisz na podstawie specyfikacji podanej we wskazówce. Zapisz program w pliku pod nazwą *Place*.
   Wskazówka:
   Dane: liczba całkowita *n* – liczba pracowników, liczba całkowita nieujemna *lng* – liczba nadgodzin, liczba rzeczywista dodatnia *s* – stawka za jedną nadgodzinę.
   Wynik: wartość płacy za nadgodziny: *placa*.

Dla zainteresowanych

7. Napisz program obliczający sumę: 1 + 11 + 111 + 1111 + … + 1…1 (*n* składników), gdzie *n* wprowadzane jest z klawiatury. Zapisz program w pliku pod nazwą *Jedynki*.

Temat 71.

Instrukcje iteracyjne zagnieżdżone w języku Python

1. Zagnieżdżone instrukcje iteracyjne `for`
2. Stosowanie zagnieżdżonej instrukcji iteracyjnej `for`

Warto powtórzyć

1. Czym jest iteracja?
2. Jak działa instrukcja iteracyjna **`for`** w języku Python?
3. Do czego można zastosować funkcję `range()`?

1. Zagnieżdżone instrukcje iteracyjne `for`

Chcemy napisać program, który umożliwi wyświetlenie na ekranie monitora „prostokąta" utworzonego ze znaków „x" o określonych wymiarach oraz wypełnionego tymi znakami. W jaki sposób zapisać ten program w języku Python?

W problemie mamy do czynienia z powtarzaniem poleceń wyświetlania określonej z góry liczby znaków „x" w określonej z góry liczbie wierszy. Występuje tu pętla w pętli. Do zapisania tego problemu w języku Python możemy zastosować **zagnieżdżone** instrukcje iteracyjne **`for`**.

Instrukcje iteracyjne **`for`** mogą być zagnieżdżone, czyli instrukcją powtarzaną w pętli może być kolejna instrukcja pętli. Liczba kroków tej iteracji jest określona przez iloczyn: $n \cdot m$, gdzie n i m to liczby powtórzeń odpowiednio w każdej pętli.

```
xxxxxxxxxx
xxxxxxxxxx
xxxxxxxxxx
xxxxxxxxxx
```

Rys. 1. Układ znaków „x" w wyświetlonym na ekranie prostokącie dla $n = 4$ i $m = 10$

Przykład 1. Pętla w pętli w języku Python

Zadanie: Zapisz w języku Python algorytm, który umożliwi wyświetlenie na ekranie monitora prostokąta utworzonego ze znaków „x” o bokach *n* i *m*, gdzie *m* – liczba znaków „x” w wierszu, *n* – liczba wierszy. Wnętrze prostokąta ma być wypełnione znakami „x”. Liczba znaków w wierszu określa długość jednego boku prostokąta, a liczba wierszy określa długość drugiego boku.

Prostokat.py - D:/Dysk C/Python/Prostokat.py (3.10.0)

File Edit Format Run Options Window Help

```
m = int(input("Podaj liczbę znaków w wierszu: "))
n = int(input("Podaj liczbę wierszy: "))

for i in range(n):
    for j in range(m):
        print("x", end = "")
    print()

input("\n\nAby zakończyć, naciśnij Enter")
```

Uwaga: Domyślnie po wykonaniu funkcji `print()` kursor przechodzi do następnego wiersza, czyli ostatnim znakiem wypisywanym przez funkcję `print()` jest znak nowego wiersza. Jeśli tego nie chcemy, możemy użyć parametru `end` i określić, co ma być ostatnim znakiem, np. pusty ciąg znaków `(end = "")`.

Ćwiczenie 1. Stosujemy zagnieżdżone instrukcje iteracyjne `for`

1. Napisz specyfikację zadania z przykładu 1.
2. Przepisz program podany w przykładzie 1. Sprawdź, czy program jest zgodny z napisaną specyfikacją zadania. Zapisz program w pliku pod nazwą *Prostokat*. Uruchom i przetestuj program dla kilku różnych wartości zmiennych *m* i *n*.
3. Objaśnij, co się dzieje w poszczególnych wierszach programu.

Wskazówka: Aby objaśnić działanie pętli zagnieżdżonych, możesz prześledzić zmianę wartości zmiennych sterujących pętli zewnętrznej i wewnętrznej:

- $i = 0$ uruchamia się pętla zewnętrzna:
  - $j = 0$ uruchamia się pętla wewnętrzna,
  - dopóki $j < m$ program wyświetla znaki „x” w jednym wierszu `(end = "")`,
  - gdy j = m, czyli program wyświetlił *m* znaków „x” w jednym wierszu, następuje wyjście z pętli wewnętrznej, kursor przechodzi do nowego wiersza.
- $i = 1$ uruchamia się pętla zewnętrzna:
  - $j = 0$ uruchamia się pętla wewnętrzna itd.

Można tak rozpisywać działanie programu, aż dojdzie się do sytuacji, w której pętla zewnętrzna zostanie zakończona (czyli spełniony będzie warunek $i = n$), po czym program się zakończy.

2. Stosowanie zagnieżdżonej instrukcji iteracyjnej `for`

Ćwiczenie 2. Wyświetlamy na ekranie choinkę z gwiazdek

1. Napisz program, który umożliwi wyprowadzenie na ekran monitora „choinki" (rys. 2.) składającej się z gwiazdek „*". Liczba gwiazdek tworzących podstawę i wysokość choinki jest taka sama – wprowadzamy ją z klawiatury.
2. Zapisz program w pliku pod nazwą *Choinka*.

Wskazówka: Pamiętaj, że w funkcji `range()` parametr *krok* może być liczbą ujemną.

```
*
**
***
****
*****
******
*******
********
*********
```

Rys. 2. Wynik działania programu dla $n = 9$ – ćwiczenie 2.

```
*
**
***
****
*****
******
*******
******
*****
****
***
**
*
```

Rys. 3. Wynik działania programu dla $n = 6$ – ćwiczenie 3.



Ćwiczenie 3. Wyświetlamy na ekranie figurę z gwiazdek

1. Zmodyfikuj program zapisany w ćwiczeniu 2., aby wyświetlił figurę podobną do pokazanej na rysunku 3.
2. Zapisz program w pliku pod nazwą *Figura*.

Warto zapamiętać

- W programach komputerowych można stosować zagnieżdżone instrukcje iteracyjne.
- W przypadku dwóch zagnieżdżonych instrukcji **`for`** liczba kroków iteracji to iloczyn liczby iteracji zewnętrznej pętli i liczby iteracji wewnętrznej pętli.

Pytania i polecenia

1. Przedstaw na przykładzie, na czym polega zagnieżdżanie się instrukcji pętli.
2. Jak określa się liczbę kroków iteracji w zagnieżdżonych instrukcjach pętli?

Zadania

1. Napisz program, który umożliwi wyprowadzenie na ekran monitora prostokąta o bokach *n* i *m*, składającego się z cyfr 1 i 0, występujących w co drugim wierszu, gdzie *m* – liczba cyfr 1 lub 0 w wierszu, *n* – liczba wierszy. Zapisz program w pliku pod nazwą *Zero_jeden*.

```
1111111111
0000000000
1111111111
0000000000
1111111111
```

Rys. 4. Wynik działania programu dla $m = 10$ i $n = 5$ – zadanie 1.

2. Napisz program tworzący na ekranie obraz składający się z gwiazdek, innych znaków lub cyfr – według własnego pomysłu.

Dla zainteresowanych

3. Napisz program realizujący algorytm, który umożliwi wyprowadzenie na ekran monitora szachownicy składającej się z zer i jedynek. Zapisz program w pliku pod nazwą *Szachownica*.
4. Napisz program wypisujący na ekranie tabliczkę mnożenia liczb naturalnych od 1 do 10. Zapisz program w pliku pod nazwą *Tabliczka*.

Temat 72. Stosowanie instrukcji iteracyjnych `for` i `while` w języku Python

1. Stosowanie instrukcji iteracyjnej `for` w języku Python
2. Stosowanie instrukcji `while` w języku Python

Warto powtórzyć

1. Jakie instrukcje wykorzystujemy do zapisania powtarzających się poleceń w językach programowania?
2. Kiedy możemy zastosować instrukcję **`for`**? Podaj przykład problemu iteracyjnego, do rozwiązania którego można zastosować tę instrukcję.
3. W jaki sposób oznaczamy blok instrukcji w języku Python?

1. Stosowanie instrukcji iteracyjnej `for` w języku Python

Zbierasz pieniądze na zakup nowego komputera. Wpłacasz co miesiąc dowolną kwotę na swoje konto. Chcesz obliczyć sumę pieniędzy wpłaconych w ciągu roku. Jak zapisać rozwiązanie tego zadania w wybranym języku programowania?



W przedstawionym problemie powtarzają się dwie czynności: wprowadzania wpłacanej kwoty i powiększania sumy pieniędzy o tę kwotę. Mamy do czynienia z algorytmem iteracyjnym, w którym liczba powtórzeń (iteracji) jest z góry określona (pieniądze wpłacamy przez rok co miesiąc, czyli dwanaście razy).

Ćwiczenie 1. Obliczamy sumę dwunastu liczb wprowadzanych z klawiatury

1. Napisz specyfikację zadania i program, który obliczy sumę dwunastu comiesięcznych wpłat na zakup komputera i wyświetli wynik na ekranie.
2. Zapisz program w pliku pod nazwą *Suma_12*.

2. Stosowanie instrukcji `while` w języku Python

Chcesz dalej oszczędzać na zakup nowego komputera, ponieważ potrzebna jest większa kwota. Zamierzasz wpłacać pieniądze, dopóki suma wpłat nie przekroczy potrzebnej kwoty. Jak przedstawić ten problem w wybranym języku programowania?

Nie wiemy z góry, ilu wpłat musimy dokonać, aby osiągnąć lub przekroczyć kwotę potrzebną na zakup komputera. Liczba iteracji zależy w tym przypadku od spełnienia warunku przekroczenia założonej kwoty.

Do zapisywania algorytmów iteracyjnych, w których liczba iteracji nie jest z góry określona, możemy zastosować instrukcję iteracyjną **`while`**.

Instrukcja iteracyjna **`while`**	**`while`** *`warunek`*: *`lista_instrukcji`* *`kolejna_instrukcja`*

Działanie pętli **`while`**: najpierw sprawdzany jest *warunek*; jeśli jest spełniony, to wykonywana jest *lista_instrukcji*. Wewnątrz bloku *lista_instrukcji* powinna być zawsze umieszczona instrukcja, która zmienia wartość *warunku* – w przeciwnym wypadku pętla nigdy się nie zakończy. W szczególnej sytuacji, gdy *warunek* od razu nie jest spełniony, *lista_instrukcji* w ogóle nie zostanie wykonana. *Lista_instrukcji* może zawierać jedną lub wiele instrukcji.

W realizacji algorytmów iteracyjnych ważne jest prawidłowe określenie sposobu zakończenia działań. Nieprawidłowe określenie tych warunków może spowodować, że obliczenia nigdy się nie zakończą, czyli nastąpi **zapętlenie programu**.

Przykład 1. Stosowanie instrukcji `while` w języku Python

Zadanie: Napisz program obliczający sumę wpłat aż do osiągnięcia lub przekroczenia założonej kwoty.

Dane: liczba rzeczywista dodatnia *kwota*, oznaczająca kwotę potrzebną na zakup komputera, ciąg dowolnych liczb rzeczywistych dodatnich *wplata*, oznaczających kolejne comiesięczne wpłaty.

Wynik: liczba rzeczywista dodatnia *suma_wplat*, oznaczająca wartość sumy wszystkich wpłat.

Zbiorka.py - D:/Python/Zbiorka.py (3.10.0)

File Edit Format Run Options Window Help

```
kwota = float(input("Podaj potrzebną kwotę: "))
suma_wplat = 0

while suma_wplat < kwota:
    wplata = float(input("Wprowadź wpłatę: "))
    suma_wplat = suma_wplat + wplata

print("Suma wpłat wynosi:", suma_wplat, "zł")

input("\n\n\Aby zakończyć, naciśnij Enter")
```

Ln: 13 Col: 0

Ćwiczenie 2. Analizujemy, testujemy i modyfikujemy gotowy program

1. Przepisz program podany w przykładzie 1. Wyjaśnij, co się dzieje w poszczególnych wierszach programu. Sprawdź, czy program jest zgodny ze specyfikacją zadania. Zapisz program w pliku pod nazwą *Zbiorka*. Uruchom i przetestuj program dla kilku różnych wartości zmiennych. Objaśnij działanie instrukcji **`while`**, m.in. odpowiedz na pytanie: *Kiedy zostanie wykonana instrukcja* `print`*?*
2. Zmodyfikuj program, aby na ekranie wyświetlała się również bieżąca suma, uzyskana po każdej wpłacie, a na koniec – różnica pomiędzy kwotą wpłaconą a założoną (nadpłata). Dodaj odpowiednie komunikaty. Zapisz plik pod tą samą nazwą.

Ćwiczenie 3. Sprawdzamy poprawność danych

1. Dodaj do programu zapisanego w ćwiczeniu 2. sprawdzanie poprawności wprowadzanych wartości zmiennych *kwota* i *wplata*, zgodnie ze specyfikacją zadania (podaną w przykładzie 1.). Wprowadzanie i sumowanie liczb powinny być wykonywane dla poprawnych wartości zmiennych *kwota* i *wplata*:
   a. jeśli użytkownik wprowadzi niepoprawną wartość zmiennej *kwota*, wyświetl na ekranie komunikat „Wprowadzono błędną liczbę" i zakończ program,
   b. jeśli użytkownik wprowadzi niepoprawną wartość zmiennej *wplata*, wyświetl na ekranie komunikat „Wprowadzono błędną liczbę" i zignoruj wprowadzoną wartość.
2. Zapisz plik pod tą samą nazwą. Uruchom program i sprawdź jego działanie dla różnych wartości zmiennych.

Ćwiczenie 4. Stosujemy instrukcję iteracyjną **`while`**

1. Napisz specyfikację zadania i program, który będzie obliczał sumę liczb całkowitych wprowadzanych z klawiatury aż do wprowadzenia zera. Wprowadzenie zera kończy zliczanie. Wynik sumowania wyprowadź na ekran.
2. Zapisz program w pliku pod nazwą *Suma_while*.

Wskazówka: Na początku programu przypisz zmiennej *liczba* wartość różną od zera, np. 1, a zmiennej *suma* – 0.

Warto zapamiętać

- W językach programowania występują instrukcje iteracyjne, które umożliwiają zapis algorytmów iteracyjnych, np.: w języku Python instrukcje **`for`** i **`while`**. Zwykle gdy znamy liczbę iteracji, stosujemy instrukcję **`for`**; jeśli nie wiemy, ile będzie powtórzeń – instrukcję **`while`**.
- W instrukcji **`while`** najpierw jest sprawdzany warunek logiczny, a potem (gdy warunek jest prawdziwy) są wykonywane instrukcje znajdujące się wewnątrz instrukcji **`while`**. Gdy warunek od razu nie jest spełniony, instrukcje wewnątrz instrukcji **`while`** nie zostaną wykonane ani razu.
- Zapętlenie programu nastąpi wtedy, gdy nie określimy prawidłowo warunku zakończenia algorytmu.

Pytania i polecenia

1. Jakie instrukcje w językach programowania umożliwiają zapis algorytmów iteracyjnych?
2. Wyjaśnij na przykładzie wybranego języka programowania, jak działa instrukcja **`while`**.
3. Czy zostaną wykonane instrukcje wewnątrz instrukcji **`while`**, jeśli warunek jest prawdziwy? Uzasadnij odpowiedź.
4. Czym różni się działanie instrukcji **`for`** od **`while`**?

Zadania

1. Znajdź w Internecie wartość średniej temperatury w Polsce w poszczególnych miesiącach – w dzień i w nocy. Napisz program obliczający na podstawie tych danych średnie roczne temperatury w dzień i w nocy. Wyniki wyświetl na ekranie. Zapisz program w pliku pod nazwą *Temperatury*.
2. Napisz program obliczający i wyświetlający na ekranie iloraz dwóch liczb całkowitych (*dzielna*, *dzielnik*) wprowadzanych z klawiatury aż do wprowadzenia dzielnika równego zero. Ponieważ nie da się obliczyć ilorazu dla dzielnika równego zero, program powinien wyświetlić odpowiedni komunikat. Zapisz program w pliku pod nazwą *Iloraz*.
3. Napisz program obliczający i wyświetlający na ekranie objętość sześcianu o boku *bok* aż do wprowadzenia niedodatniej wartości zmiennej *bok*. Zapisz program w pliku pod nazwą *Objetosc_szescianu*.
4. Napisz specyfikację zadania i program zliczający, ile jest liczb podzielnych przez 3 pośród liczb całkowitych wprowadzanych z klawiatury. W przypadku wprowadzenia zera program ma zakończyć działanie. Zapisz program w pliku pod nazwą *Podzielne_przez_3*.

Dla zainteresowanych

5. Masz *n* zł na kupienie upominków dla znajomych. Upominki możesz kupować, dopóki starczy ci pieniędzy. Ceny upominków wprowadzaj z klawiatury. Nie możesz przekroczyć kwoty *n* zł ani wydać mniej. Jeśli cena ostatniego upominku przekroczy kwotę *n*, musisz kupić coś tańszego. Program kończy się, gdy wydasz dokładnie *n* zł. Napisz odpowiedni program i zapisz go w pliku pod nazwą *Upominki*.
6. Napisz program obliczający część całkowitą pierwiastka kwadratowego z podanej liczby naturalnej *n*.
   Wskazówka: Istnieją różne algorytmy obliczania pierwiastka. Można na przykład obliczać kwadraty kolejnych liczb naturalnych aż do uzyskania liczby większej lub równej *n*.

Temat 73.

Funkcje zwracające wartość w języku Python

1. **Funkcje w języku Python**
2. **Ogólna definicja funkcji zwracającej wartość w języku Python**
3. **Funkcje zwracające wartość z parametrami w języku Python**
4. **Funkcje zwracające wartość bez parametrów w języku Python**

Warto powtórzyć

1. W jakich środowiskach programowania tworzyliśmy programy z wykorzystaniem podprogramów?
2. W jaki sposób definiowaliśmy podprogramy w tych środowiskach? Jak je wywoływaliśmy?
3. Jak zapisujemy algorytm iteracyjny w języku Python?

1. Funkcje w języku Python

Do magazynu sprzętu RTV dowożone są kilkakrotnie w ciągu miesiąca różne artykuły. Jak napisać program zliczający, ile sztuk każdego artykułu dostarczono we wszystkich dostawach łącznie?



Jeśli musimy kilkakrotnie zsumować liczby w różnych zbiorach, możemy opracować cząstkowe rozwiązanie obliczające sumę, a potem wielokrotnie wykorzystać je w programie. Do opracowywania problemów cząstkowych, które są częściami zadania rozwiązywanego przez dany program, służą **podprogramy**.

Podprogram realizujący określone zadanie (np. sumowanie liczb) może być wielokrotnie wykorzystywany – w tym samym programie lub w innych. Stosowanie podprogramów zwiększa przejrzystość programu.

P **Podprogram**
(procedura lub funkcja)

Wyodrębniona część programu, mająca unikatową nazwę i ustalony sposób wymiany danych z innymi częściami programu.

Parametr P

Wartość przekazywana do funkcji.

Wartość zwracana W

Wartość przekazywana z funkcji do miejsca wywołania (np. programu głównego).

W języku Python wszystkie podprogramy nazywamy **funkcjami.**

Poznaliśmy kilka wbudowanych funkcji języka Python, m.in.: `print()`, `input()`. W języku Python można definiować własne funkcje.

Funkcje mogą **zwracać wartość** do programu głównego lub nic nie zwracać, mogą mieć określone **parametry** lub ich nie posiadać.

Funkcje dzielimy na: zwracające wartość i niezwracające wartości.

Nazwy funkcji tworzymy na takich samych zasadach jak nazwy zmiennych (można używać tylko liter, znaków podkreślenia i cyfr; nazwa nie może zaczynać się od cyfry; przyjęte jest niestosowanie polskich liter). Funkcjom należy nadawać nazwy opisujące ich działanie. Ważne jest, aby przyjąć jednolity sposób nazewnictwa, np. w nazwach funkcji i zmiennych używa się wyłącznie małych liter, a w nazwach wieloczłonowych stosuje się znak podkreślenia, np. *liczba_dostaw*, *suma_sztuk*, *wprowadz_dane*, *wyprowadz_dane*, *najwiekszy*.

Aby zastosować funkcję, trzeba ją **zdefiniować**, a następnie **wywołać** w programie. Wywołanie funkcji powoduje wykonanie instrukcji będących treścią funkcji i zwrócenie wyniku jej działania. W momencie wywołania funkcji **parametry formalne** (zwane też **parametrami**) są zastępowane przez **parametry aktualne** (zwane też **argumentami**).

2. Ogólna definicja funkcji zwracającej wartość w języku Python

Funkcję zwracającą wartość stosujemy, jeśli celem funkcji jest obliczenie i zwrócenie pewnej wartości do programu. Funkcja musi zawierać instrukcję **`return`** z wartością zwracaną do miejsca wywołania (np. programu głównego).

Ogólna postać definicji funkcji zwracającej wartość

```
def nazwa_funkcji(lista_parametrów_formalnych):
    lista_instrukcji
    return wartość
```

nagłówek funkcji

treść funkcji

Miejsce definiowania funkcji	Funkcję definiujemy w dowolnym miejscu programu (zazwyczaj na początku), ale przed pierwszym wywołaniem.
Określanie typu funkcji i parametrów	Nie określamy typu funkcji ani typów parametrów formalnych funkcji – interpreter rozpozna je automatycznie (podobnie jak w przypadku zmiennych).
Treść funkcji	Treść funkcji (*lista_instrukcji* i instrukcja `return`) musi być przesunięta w prawo przynajmniej o jedną spację względem nagłówka funkcji (przyjęte jest wcięcie składające się z czterech spacji).

Tabela 1. Zapis funkcji zwracającej wartość w języku Python

Aby wywołać funkcję zwracającą wartość, należy podać nazwę funkcji, a w nawiasach – wartości parametrów. Wartość zwróconą przez funkcję możemy np. przypisać zmiennej w programie głównym. W przypadku definiowania i wywoływania funkcji bez parametrów nawiasy po nazwie funkcji pozostawiamy puste.

3. Funkcje zwracające wartość z parametrami w języku Python

Sposób definiowania i wywołania funkcji zwracającej wartość pokażemy na przykładzie funkcji z jednym parametrem, obliczającej sumę *n* liczb.

Przykład 1. Definiowanie i wywołanie funkcji zwracającej wartość z jednym parametrem w języku Python

Zadanie: Do pewnego magazynu sprzętu RTV dowożone są kilkakrotnie w ciągu miesiąca telewizory. Utwórz program obliczający i wyprowadzający na ekran łączną liczbę telewizorów ze wszystkich dostaw. Liczbę dostaw oraz wielkość każdej dostawy wprowadzaj z klawiatury po uruchomieniu programu.

Dane:

- liczba naturalna większa od zera *l_dostaw_tv* oznaczająca liczbę dostaw telewizorów,
- *l_dostaw_tv* liczb naturalnych większych od zera wprowadzanych kolejno i zapamiętywanych w zmiennej *a*, oznaczających odpowiednio wielkości kolejnych dostaw telewizorów.

Wyniki: łączna liczba dostarczonych telewizorów: *suma_tv*.

Opis rozwiązania: Rysunek przedstawia trzy dostawy telewizorów (czyli *l_dostaw_tv* = 3), w których dostarczono kolejno: 8, 15 i 20 telewizorów, a więc *suma_tv* = 43.

Na początku programu zdefiniujemy funkcję *suma* z jednym parametrem, obliczającą i zwracającą sumę *n* liczb wprowadzanych z klawiatury. Parametrem przekazywanym do funkcji jest liczba elementów zbioru *n*.

Funkcję *suma()* wywołamy w programie głównym z parametrem aktualnym *l_dostaw_tv*. Możemy to zrobić na dwa sposoby.



Wywołanie funkcji *suma()* – sposób 1.
Wynik działania funkcji (zwracaną wartość) możemy przypisać zmiennej *suma_tv*.

```
suma_tv = suma(l_dostaw_tv).
```

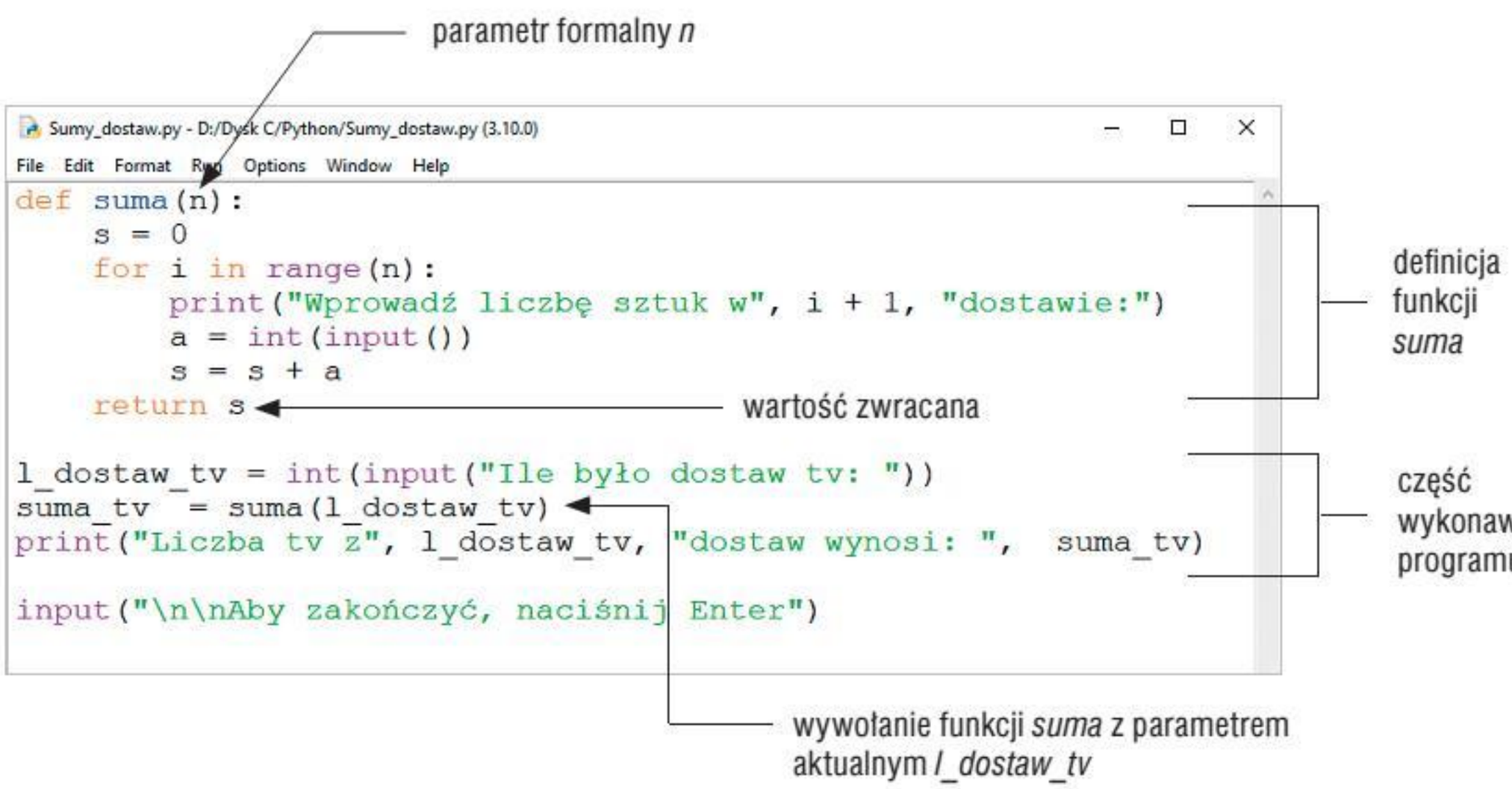
Wywołanie funkcji *suma()* – sposób 2.
Wynik działania funkcji zwracającej wartość możemy również wyprowadzić na ekran bez używania dodatkowych zmiennych (tu: *suma_tv*) i przypisywania im wartości funkcji.

```
print("Liczba tv z", l_dostaw_tv, "dostaw wynosi", suma(l_dostaw_tv))
```

Ćwiczenie 1. Definiujemy funkcję zwracającą wartość z jednym parametrem

1. Przepisz program podany w przykładzie 1. Zapisz program w pliku pod nazwą *Sumy _dostaw*. Uruchom i przetestuj program dla kilku różnych wartości zmiennych. Wyjaśnij znaczenie poszczególnych wierszy programu. Wskaż miejsce wywołania funkcji.
2. W programie zapisano tylko obliczenia łącznej wielkości dostaw telewizorów. Dodaj podobne instrukcje dla obliczenia łącznej wielkości dostaw głośników.
   a. Przyjmij następujące nazwy dla zmiennych: *l_dostaw_glosnikow*, oznaczającą liczbę dostaw głośników i *suma_glosnikow*, oznaczającą łączną liczbę dostarczonych głośników.
   b. Funkcję *suma()* wywołaj w programie głównym z parametrem *l_dostaw_glosnikow*.
3. Zapisz plik pod tą samą nazwą.

Wskazówka: Ułatwiaj sobie pracę i kopiuj podobne fragmenty programu, w których wystarczy tylko zmienić nazwy zmiennych.



Ćwiczenie 2. Dodajemy do funkcji i programu sprawdzanie poprawności danych

1. Otwórz plik *Sumy_dostaw* zapisany w ćwiczeniu 1.
2. Zmodyfikuj funkcję i program, aby obliczenia były wykonywane dla poprawnych (czyli większych od zera) wartości wszystkich zmiennych: *a*, *l_dostaw_tv*, *l_dostaw_glosnikow*. Jeśli zostanie wprowadzona niepoprawna wartość, wyświetl na ekranie komunikat „Wprowadzono błędną liczbę" i zignoruj wprowadzoną wartość.
3. Zapisz program w pliku pod nazwą *Sumy_dostaw_popr*.
4. Uruchom i przetestuj program dla kilku różnych wartości zmiennych.

Ćwiczenie 3. Modyfikujemy program

1. Otwórz plik *Sumy_dostaw_popr* zapisany w ćwiczeniu 2.
2. Zmniejsz liczbę zmiennych, czyli do wyprowadzenia wyników funkcji użyj sposobu 2. z przykładu 1.
3. Do programu dodaj zliczanie dwóch innych artykułów przywożonych do magazynu: smartfonów i komputerów.
4. Zapisz program w pliku pod nazwą *Sumy_dostaw_zmod*.

4. Funkcje zwracające wartość bez parametrów w języku Python

Funkcje można definiować bez parametrów, ale wówczas, aby przekazać do nich dane, musimy zastosować tzw. **zmienne globalne** (widoczne w dowolnym miejscu programu) – w funkcjach i w programie głównym.

Niezależnie od zmiennych globalnych można w funkcji stosować **zmienne lokalne**. W języku Python zmienne lokalne są widoczne w całej funkcji. W funkcji *suma* (przykład 1.) zmienne *i*, *a* oraz *s* są lokalne.

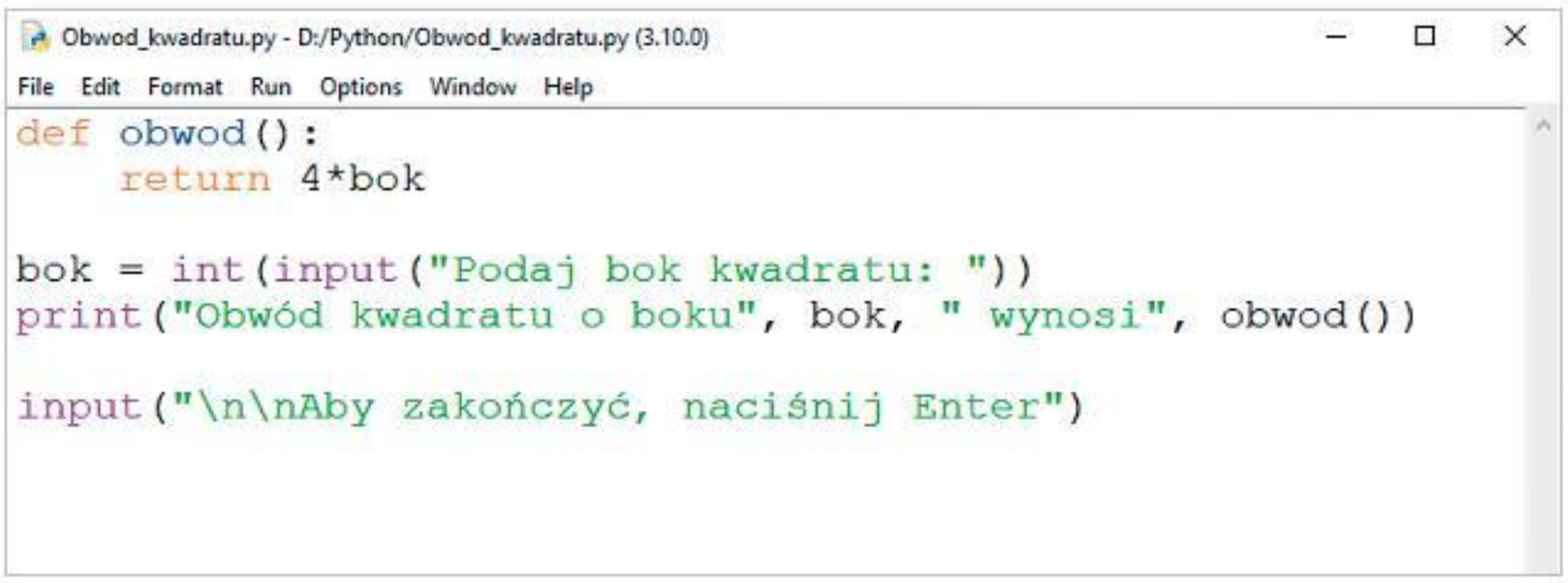
Przykład 2. Stosowanie funkcji zwracającej wartość bez parametrów w języku Python

Zadanie: Zdefiniuj funkcję *obwod()* bez parametrów, zwracającą do programu głównego obwód kwadratu o boku *bok*. Wywołaj funkcję w programie głównym i wyprowadź wartość obwodu. Wartość zmiennej *bok* wprowadzaj z klawiatury.

Dane: liczba naturalna *bok* oznaczająca długość boku kwadratu.

Wynik: wartość obwodu kwadratu.

Opis rozwiązania: W programie zastosujemy zmienną globalną *bok*. Typ zmiennej *bok* określa instrukcja `bok = int(input("Podaj bok kwadratu: "))`.



Ćwiczenie 4. Definiujemy funkcję zwracającą wartość bez parametrów

1. Przepisz program podany w przykładzie 2. Zapisz program w pliku pod nazwą *Obwod_kwadratu*. Uruchom i przetestuj program dla kilku różnych wartości zmiennej. Wyjaśnij znaczenie poszczególnych wierszy programu. Wskaż miejsce wywołania funkcji.
2. Dodaj do programu funkcję *pole()* bez parametru, obliczającą pole kwadratu o boku *bok*.
3. Zapisz program w pliku pod nazwą *Obwod_i_pole_kwadratu*. Uruchom i przetestuj program dla kilku różnych wartości zmiennej.

Ćwiczenie 5. Definiujemy funkcję zwracającą wartość z parametrem

1. Otwórz plik *Obwod_i_pole_kwadratu* zapisany w ćwiczeniu 4. Zmodyfikuj program, rezygnując ze zmiennej globalnej i zmieniając definicje funkcji *suma()* i *obwod()* na funkcje z parametrem. W programie głównym wywołaj funkcje z parametrem aktualnym *x*, wprowadzanym z klawiatury jako liczba całkowita.
2. Zapisz program w pliku pod nazwą *Obwod_i_pole_kwadratu_z_par*.

Wskazówka: W obydwu funkcjach można użyć tej samej nazwy parametru.

```
def obwod(bok):
    return 4*bok

def pole(bok):
    return bok*bok
```

Rys. 1. Definicje funkcji z parametrem – ćwiczenie 5.

Warto zapamiętać

- Podprogramy (funkcje) stosujemy, aby opracowywać oddzielnie problemy cząstkowe i w ten sposób pisać przejrzyste programy.
- Funkcje w języku Python mogą zwracać wartości lub nie. Każdy rodzaj funkcji możemy definiować z parametrami lub bez.
- Funkcję wywołujemy w programie głównym, podając jej nazwę i parametry aktualne. W przypadku braku parametrów, po nazwie funkcji umieszczamy puste nawiasy. W przypadku funkcji zwracającej wartość, wartość zwracaną możemy np. przypisać zmiennej w programie głównym.
- Program główny komunikuje się z funkcją przez przekazywanie parametrów – parametry aktualne są wprowadzane w miejsce parametrów formalnych w momencie wywołania funkcji. Gdy wywołanie funkcji się zakończy, zwraca wartość jako swój wynik.

Pytania i polecenia

1. Dlaczego definiujemy podprogramy?
2. Na czym polega wywołanie funkcji? Czym jest parametr formalny, a czym parametr aktualny?
3. Kiedy definiujemy funkcję zwracającą wartość?
4. Czym jest wartość zwracana?
5. Omów ogólną postać definicji funkcji zwracającej wartość.
6. W jaki sposób wywołujemy funkcję zwracającą wartość?

Zadania

1. Do programu *Sumy_dostaw_zmod* zapisanego w ćwiczeniu 3. dodaj jeszcze dwa inne artykuły, które są dostarczane do magazynu i wykonaj te same obliczenia co dla telewizorów i innych artykułów. Zapisz program w pliku *Sumy_dostaw_6*.
2. Napisz program umożliwiający obliczenie objętości graniastosłupa prawidłowego czworokątnego. Zdefiniuj funkcję *objetosc()* z dwoma parametrami (*a*, *h*), obliczającą objętość graniastosłupa prawidłowego czworokątnego o krawędzi podstawy *a* i wysokości *h* i zwracającą do programu głównego wynik obliczeń. Wywołaj funkcję w programie głównym z parametrami aktualnymi *bok* i *wysokosc* wprowadzanymi z klawiatury jako liczby całkowite. Wyprowadź na ekran wynik – objętość graniastosłupa prawidłowego czworokątnego o krawędzi podstawy *bok* i wysokości *wysokosc*. Zapisz program w pliku pod nazwą *Objetosc_graniastoslupa*.
3. Do programu *Objetosc_graniastoslupa* zapisanego w zadaniu 2. dodaj sprawdzanie poprawności wprowadzanych danych. Jeśli użytkownik wprowadzi niepoprawną (ujemną lub równą zero) wartość zmiennej *bok* lub *wysokosc*, wyświetl na ekranie komunikat „Wprowadzono błędną liczbę”. Zapisz program w pliku pod nazwą *Objetosc_graniastoslupa_popr*.

Dla zainteresowanych

4. Wymyśl samodzielnie problem (podobnie jak w przykładzie 1.), w którym możesz wykorzystać funkcję *suma()*. Sformułuj zadanie i jego specyfikację. Napisz program realizujący to zadanie.
5. Utwórz program zliczający, ile liczb parzystych wprowadzono i wyświetlający wynik na ekranie. Zliczanie wykonuj, dopóki nie zostanie wprowadzone zero. Zdefiniuj funkcję *czy _parzysta()* z parametrem całkowitym *a*, sprawdzającą, czy liczba jest parzysta. Wywołaj funkcję w programie głównym. Zapisz program w pliku pod nazwą *Zlicz_parzyste*.
   Wskazówka: Funkcja *czy _parzysta()* zwraca wartość logiczną PRAWDA lub FAŁSZ. Wartość różną od zera uważa się za logiczną prawdę, a wartość równą 0 – za logiczny fałsz.
   Postać funkcji może być następująca:

```
def czy_parzysta(a):
    return a % 2 == 0
```

Rys. 2. Definicja funkcji – zadanie 5.

Temat 74. Funkcje niezwracające wartości w języku Python

1. Ogólna definicja funkcji niezwracającej wartości w języku Python
2. Funkcje niezwracające wartości bez parametrów w języku Python
3. Funkcje niezwracające wartości z parametrami w języku Python

Warto powtórzyć

1. Kiedy stosujemy funkcję zwracającą wartość?
2. Czym jest parametr formalny, a czym aktualny?
3. Na czym polega wywołanie funkcji?

1. Ogólna definicja funkcji niezwracającej wartości w języku Python

Funkcje niezwracające wartości stosujemy, gdy mamy wykonać pewne czynności, np. wprowadzić dane, wyprowadzić dane lub obliczyć wiele wartości.

Ogólna postać definicji funkcji niezwracającej wartości	`def` *`nazwa_funkcji(lista_parametrów_formalnych)`*: `    lista_instrukcji`
Miejsce definiowania funkcji	Funkcję definiujemy w dowolnym miejscu programu (zazwyczaj na początku), ale przed pierwszym wywołaniem.
Określanie typu funkcji i parametrów	Nie określamy typu parametru formalnego funkcji, ponieważ, podobnie jak w przypadku zmiennych, interpreter automatycznie rozpoznaje typ parametru.
Treść funkcji	Treść funkcji (*lista_instrukcji*) musi być przesunięta w prawo przynajmniej o jedną spację względem nagłówka funkcji (przyjęte jest wcięcie składające się z czterech spacji). Funkcja nie zwraca wartości, dlatego instrukcja `return` nie musi wystąpić (a jeśli występuje, to nie może określać wartości).

Tabela 1. Zapis funkcji niezwracającej wartości w języku Python

Aby wywołać funkcję niezwracającą wartości, należy umieścić nazwę funkcji z **parametrami aktualnymi** w odpowiednim miejscu programu głównego. W przypadku braku parametrów nawiasy po nazwie funkcji pozostawiamy puste.

Wywołanie funkcji niezwracającej wartości	*nazwa_funkcji(lista_parametrów_aktualnych)* lub *nazwa_funkcji()*

2. Funkcje niezwracające wartości bez parametrów w języku Python

Przykład 1. Stosowanie funkcji niezwracającej wartości bez parametrów w języku Python

Zadanie: Napisz program umożliwiający wyprowadzenie na ekran monitora napisu „Poznajemy funkcje", a pod nim w jednym wierszu dwudziestu gwiazdek (znaków *). Zdefiniuj funkcję niezwracającą wartości o nazwie *gwiazdki()* bez parametrów. Zadaniem funkcji jest wyprowadzenie w jednym wierszu dwudziestu gwiazdek i przeniesienie kursora do nowego wiersza. Funkcję wywołaj w programie głównym.

Opis rozwiązania: Celem funkcji jest wyświetlenie z góry określonej liczby gwiazdek. Żadna wartość nie jest przekazywana z programu głównego, więc funkcja nie musi mieć parametrów.

W programie nie używamy żadnych zmiennych.

Gwiazdki.py - C:\Python\Gwiazdki.py (3.7.2)

File Edit Format Run Options Window Help

```
def gwiazdki():
    print("*"*20)

print("Poznajemy funkcje")
gwiazdki()

input("\n\nAby zakończyć, naciśnij Enter")
```

definicja funkcji

wywołanie funkcji

Ćwiczenie 1. Definiujemy funkcję niezwracającą wartości bez parametrów w języku Python

1. Przepisz program podany w przykładzie 1.
2. Zapisz program w pliku pod nazwą *Gwiazdki*. Uruchom i przetestuj program. Wyjaśnij znaczenie poszczególnych wierszy programu. Wskaż miejsce wywołania funkcji.

Ćwiczenie 2. Modyfikujemy program

1. Dodaj do programu zapisanego w ćwiczeniu 1. wyświetlenie w kolejnych wierszach tekstów: „w języku Python", „na lekcji informatyki". Pod każdym napisem wyświetl gwiazdki. Ile razy wywołasz funkcję *gwiazdki()*?
2. Zapisz program w pliku pod nazwą *Gwiazdki_z_napisami*.

3. Funkcje niezwracające wartości z parametrami w języku Python

Przykład 2. Stosowanie funkcji niezwracającej wartości z parametrami w języku Python

Chcemy, aby liczba wyświetlanych gwiazdek była wprowadzana z klawiatury i przekazywana do funkcji jako parametr.

Do definicji funkcji dodamy parametr *n*, który w momencie wywołania funkcji będzie zastąpiony parametrem aktualnym *l_gwiazdek* (liczbą całkowitą większą od zera wprowadzaną z klawiatury).

```
def gwiazdki(n):
    print("*" * n)
```

Wywołanie funkcji w programie głównym: `gwiazdki(l_gwiazdek)`

Ćwiczenie 3. Definiujemy funkcję niezwracającą wartości z parametrami w języku Python

1. Otwórz plik *Gwiazdki* zapisany w ćwiczeniu 1. Zmień definicję funkcji *gwiazdki()* na funkcję z parametrem *n* i wywołaj w programie głównym z parametrem aktualnym, będącym liczbą całkowitą większą od zera (*l_gwiazdek*). Wartość parametru wprowadzaj z klawiatury.
2. Zapisz program w pliku pod nazwą *Gwiazdki_par*.
3. Uruchom i przetestuj program dla różnych wartości zmiennej *l_gwiazdek*.

Ćwiczenie 4. Modyfikujemy program

1. Otwórz plik *Gwiazdki_z_napisami* zapisany w ćwiczeniu 2. Zmień definicję funkcji *gwiazdki* na funkcję z parametrem *n* i wywołaj w programie głównym z parametrem aktualnym będącym liczbą całkowitą (*l_gwiazdek*). Wartość parametru wprowadzaj z klawiatury.
2. Zapisz program w pliku pod nazwą *Gwiazdki_z_napisami_par*.
3. Uruchom i przetestuj program dla różnych wartości zmiennej *l_gwiazdek*.

Warto zapamiętać

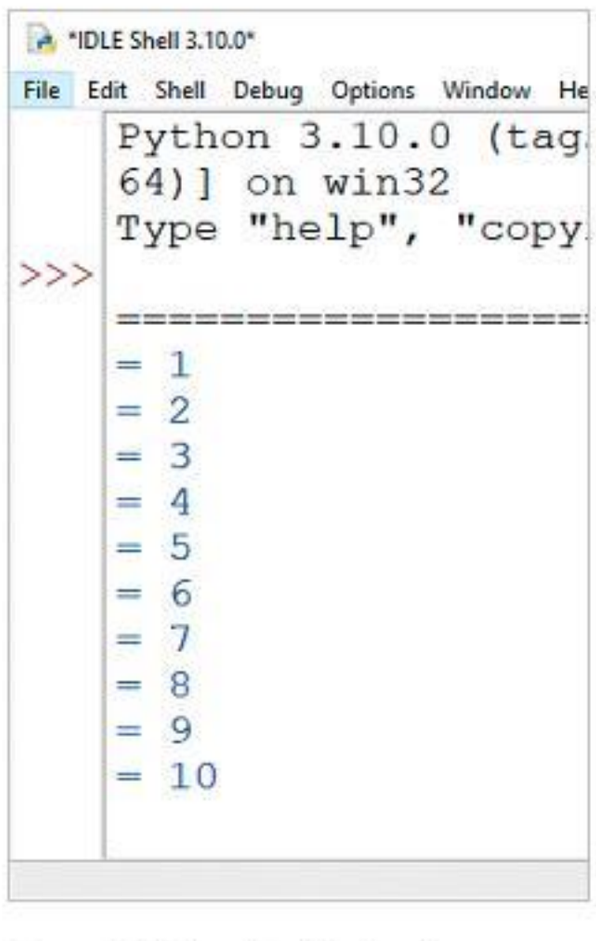
- Funkcję niezwracającą wartości stosujemy, gdy mamy wykonać pewne czynności, bez zwracania wartości. Można ją definiować z parametrami lub bez.
- Funkcję niezwracającą wartości wywołujemy, umieszczając nazwę funkcji z parametrami aktualnymi w odpowiednim miejscu programu głównego. W przypadku braku parametrów, po nazwie funkcji umieszczamy puste nawiasy.

Pytania i polecenia

1. Kiedy definiujemy funkcję niezwracającą wartości?
2. Omów ogólną postać definicji funkcji niezwracającej wartości.
3. W jaki sposób wywołujemy funkcję niezwracającą wartości?

Zadania

1. Zdefiniuj funkcję *kolumna()* niezwracającą wartości bez parametrów, wyświetlającą na ekranie w kolumnie liczby od 1 do 10 poprzedzone znakiem „=" i spacją – jak pokazano na rysunku 1. Funkcję wywołaj w programie głównym. Zapisz program w pliku pod nazwą *Kolumna_liczb*.



Rys. 1. Wynik działania programu – zadanie 1.

2. Zmodyfikuj program *Kolumna_liczb* zapisany w zadaniu 1., dodając do funkcji parametr *ile* określający liczbę wyświetlanych rzędów liczb. Funkcję wywołaj w programie głównym z parametrem aktualnym *ile_liczb*. Dodaj sprawdzanie poprawności wprowadzanych danych (muszą być większe od zera). Zapisz program w pliku pod nazwą *Kolumna_liczb_popr*.

Dla zainteresowanych

3. Wymyśl samodzielnie zadanie, w którym możesz wykorzystać funkcję niezwracającą wartości. Sformułuj zadanie i jego specyfikację. Napisz program realizujący to zadanie.

Temat 75. Definiowanie i stosowanie listy w języku Python

1. Zmienne indeksowane
2. Definiowanie listy w języku Python
3. Wprowadzanie elementów do listy i wyprowadzanie elementów na ekran

Warto powtórzyć

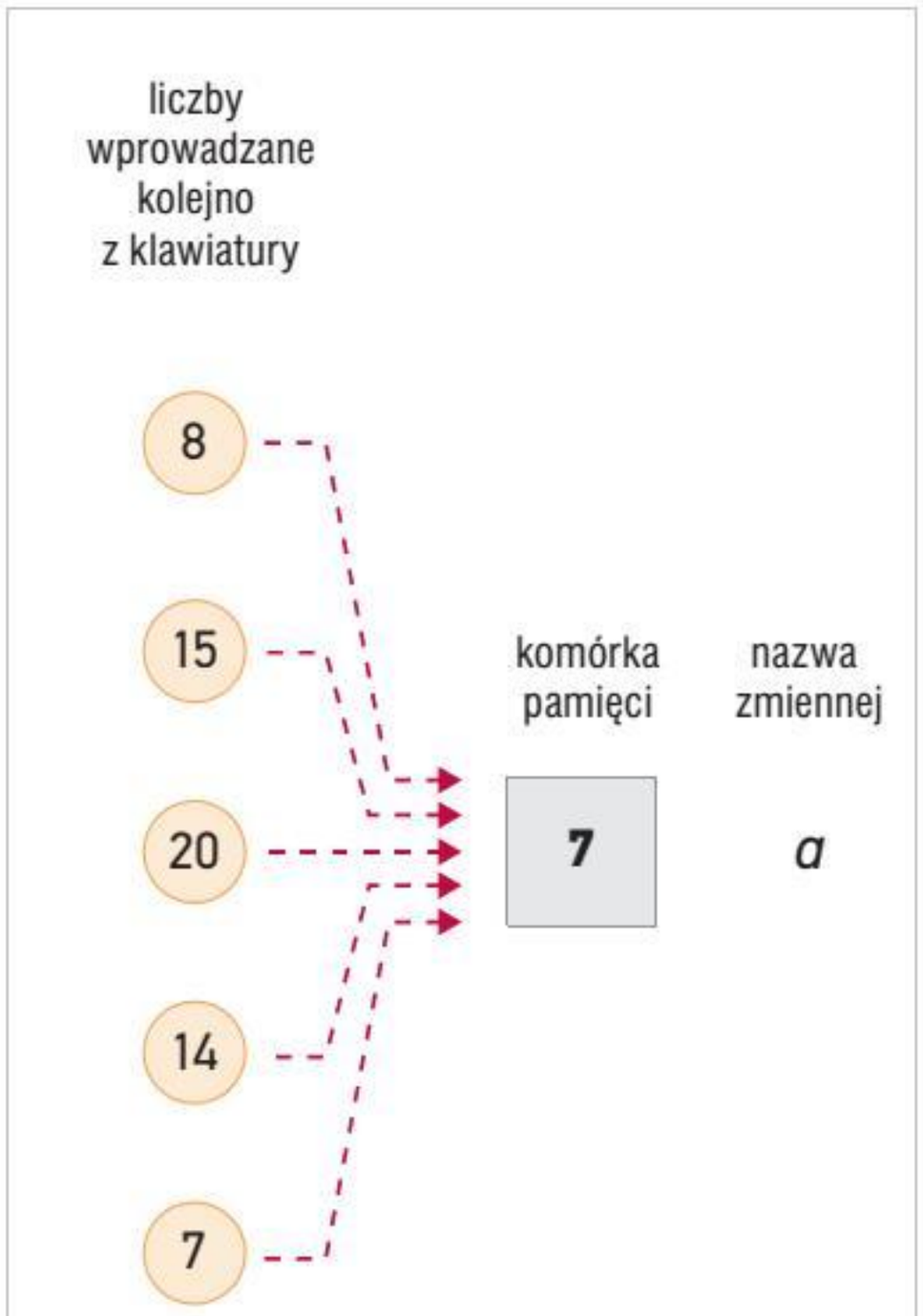
1. Do czego służą funkcje w językach programowania?
2. Kiedy stosujemy funkcję zwracającą wartość, a kiedy niezwracającą wartości?
3. W jaki sposób definiujemy funkcje w języku Python?
4. Jak wywołujemy funkcję z parametrami, a jak funkcję bez parametrów?
5. Jakie było działanie funkcji *suma()* omówionej przykładzie 1. z tematu 73? Jaką wartość zwracała do programu głównego?

1. Zmienne indeksowane

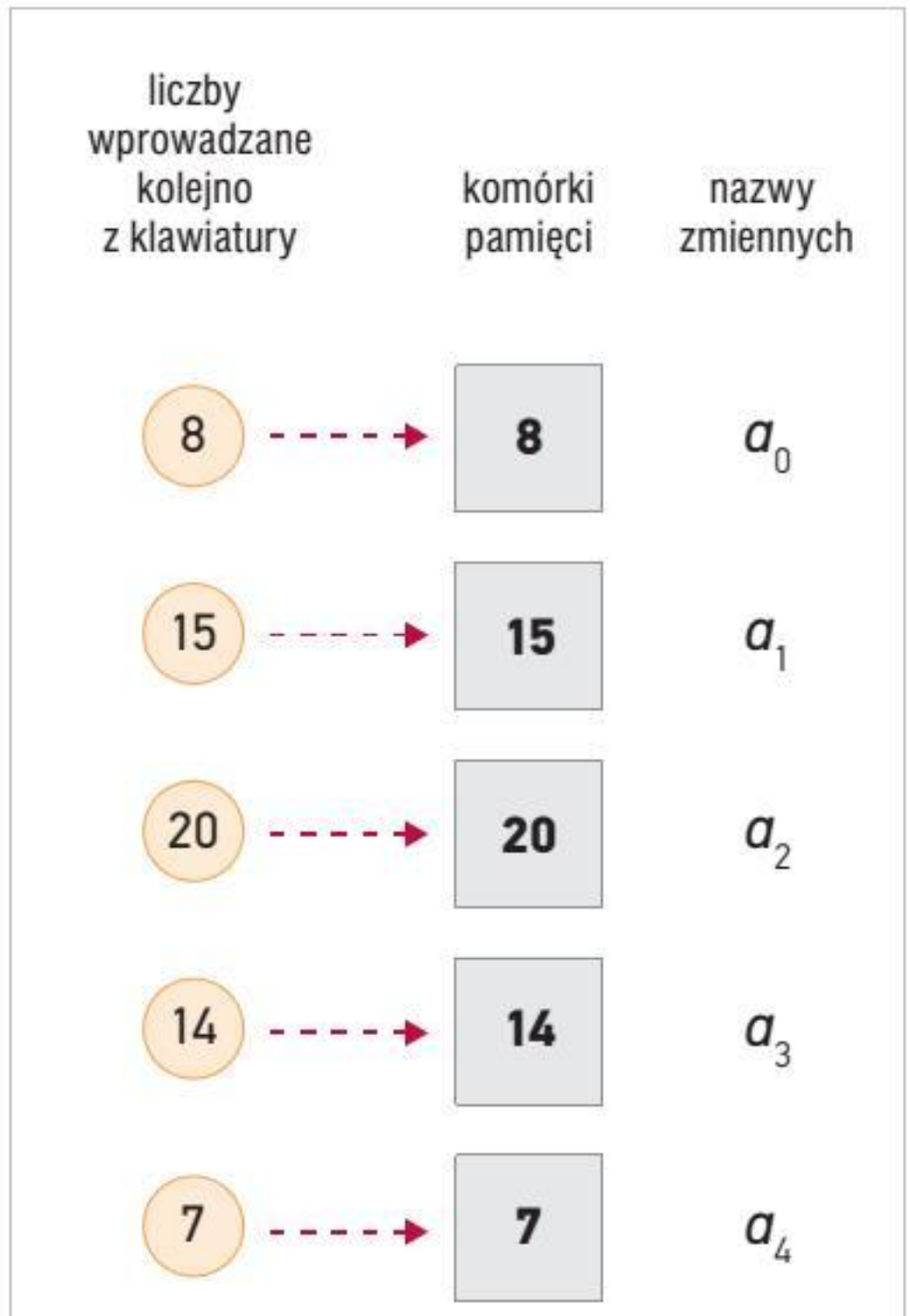
W funkcji obliczającej sumę liczb wprowadzanych z klawiatury wszystkie wprowadzane liczby komputer pamiętał kolejno w jednej zmiennej (o nazwie *a*), czyli nie zapamiętywał poszczególnych liczb. W jaki sposób zapamiętać wszystkie wprowadzone dane, aby można po ich wprowadzeniu odwoływać się do wybranych danych?

Gdy z klawiatury wprowadzimy kolejno liczby, na przykład wielkości dostaw telewizorów ze zbioru {8, 15, 20, 14, 7}, każda wprowadzana liczba zastępuje poprzednią (rys. 1a).

Aby zapamiętać wszystkie dane, musimy je różnie nazwać. Moglibyśmy użyć liter alfabetu, ale w przypadku dużej liczby danych zabrakłoby liter. Ponadto odwoływanie się do wielu zmiennych byłoby niewygodne. Dlatego stosujemy tzw. **zmienne indeksowane**, w których do nazwy zmiennej (np. *a*) dodaje się kolejny numer (**indeks**), np.: a_0, a_1, …, a_{n-1}. Wówczas dla każdej zmiennej komputer rezerwuje oddzielną komórkę pamięci, np. na poszczególne wielkości dostaw telewizorów z pięciu transportów (rys. 1b).



Rys. 1a. Zapisywanie w pamięci komputera wartości danych wprowadzanych z klawiatury z użyciem jednej zmiennej



Rys. 1b. Zapisywanie w pamięci komputera wartości danych wprowadzanych z klawiatury z użyciem pięciu zmiennych indeksowanych

Ćwiczenie 1. Wyjaśniamy, czym są zmienne indeksowane

Objaśnij, czym są zmienne indeksowane, korzystając z przygotowanych pomocy dydaktycznych, np. kartek z przykładowymi liczbami.

2. Definiowanie listy w języku Python

Zmienne indeksowane w języku Python możemy wykorzystywać, definiując specjalną strukturę danych – **listę**.

Ogólna postać definiowania listy	`nazwa_listy = [element1, element2, …, element_n]`

Do elementów listy odwołujemy się, podając nazwę listy i indeks elementu umieszczony w nawiasach kwadratowych, np. *a*[0], *a*[1], …, *a*[*n* - 1] – dla listy *n*-elementowej o nazwie *a*.

W języku Python pierwszy indeks jest zawsze równy 0.

Lista może zawierać elementy różnego typu, dlatego w języku Python nie określamy typu elementów listy.

Zarówno typ zmiennych przechowywanych w liście, jak i liczba elementów listy mogą się zmienić w trakcie działania programu.

Wartości elementów listy możemy:

- podać od razu w definicji listy,
- przypisać od razu taką samą wartość wszystkim elementom listy.

Liczbę elementów listy możemy przypisać zmiennej.

! Uwaga

Przykłady definiowania list w języku Python ograniczyliśmy do kilku przydatnych do dalszych ćwiczeń.

Definicja listy	Opis	Sposób odwołania
`DOSTAWY = [8,15,20,14,7]`	oznacza zdefiniowanie listy o nazwie *DOSTAWY*, składającej się z pięciu elementów	do elementów listy odwołujemy się przez zmienne: *DOSTAWY*[0], *DOSTAWY*[1], …, *DOSTAWY*[4]
`lista_elementow = [0] * 20`	oznacza zdefiniowanie listy o nazwie *lista_elementow* składającej się z dwudziestu elementów, z których każdy jest równy zero (wyzerowanie wszystkich elementów listy)	do elementów listy odwołujemy się przez zmienne: *lista_elementow* [0], *lista_elementow*[1], …, *lista_elementow*[19]
`N = 50` `a = [0] * N`	oznacza zdefiniowanie listy o nazwie *a*, składającej się z pięćdziesięciu elementów o wartościach zero	do elementów listy odwołujemy się przez zmienne: *a*[0], *a*[1], …, *a*[N-1]

Tabela 1. Przykłady definiowania listy w języku Python

3. Wprowadzanie elementów do listy i wyprowadzanie elementów na ekran

Do wprowadzania danych do listy oraz do wyprowadzania elementów z listy na ekran monitora możemy zastosować funkcję niezwracającą wartości bez parametrów lub z parametrami.

Przykład 1. Definiowanie i stosowanie listy w języku Python

Zadanie: Wprowadź *N* liczb całkowitych do listy o nazwie *a*, następnie wyprowadź w kolumnie elementy listy na ekran. Zdefiniuj dwie funkcje niezwracające wartości bez parametrów *wprowadz_dane()* i *wyprowadz_dane()*, które wywołaj w programie głównym.

Dane: liczba naturalna *N* określająca liczbę elementów listy, lista liczb *a*[*N*].

Wyniki: wyświetlone w kolumnie elementy listy *a*: *a*[0], *a*[1], …, *a*[*N*-1].

Elementy_listy.py - C:\Python\Elementy_listy.py (3.7.2)

File Edit Format Run Options Window Help

```
N = 5
a = [0] * N

def wprowadz_dane():
    for i in range(N):
        a[i] = int(input("Podaj liczbe: "))

def wyprowadz_dane():
    for i in range(N):
        print("a[", i, "] = ", a[i])

wprowadz_dane()
wyprowadz_dane()

input("\n\nAby zakończyć, naciśnij Enter")
```

określenie liczby elementów listy

zdefiniowanie listy o nazwie *a* i wyzerowanie jej elementów

definicja funkcji *wprowadz_dane()*

definicja funkcji *wyprowadz_dane()*

wywołanie funkcji *wprowadz_dane()*

wywołanie funkcji *wyprowadz_dane()*

Uwaga: Funkcja *wprowadz_dane* zmodyfikowała wartości istniejących zmiennych (które na początku były równe zero).

Ćwiczenie 2. Wprowadzamy dane do listy i wyprowadzamy je na ekran

1. Utwórz nowy plik źródłowy i przepisz program z przykładu 1. Uruchom i przetestuj program dla kilku różnych wartości zmiennych.
2. Zapisz program w pliku pod nazwą *Elementy _listy*.
3. Uruchom i przetestuj program dla kilku różnych wartości elementów listy *a*.

Uwaga

W języku Python wszystkie zmienne, których nazwa jest zapisana wielkimi literami (i ewentualnie z podkreślnikami), będziemy traktować w programie jak stałe.

Ćwiczenie 3. Wyprowadzamy elementy z listy w odwrotnej kolejności

1. Otwórz program *Elementy _listy* zapisany w ćwiczeniu 2. Zmodyfikuj program, aby dane wyświetlały się w odwrotnym porządku. Dodatkowo zwiększ liczbę danych do dziesięciu.
2. Zapisz program w pliku pod nazwą *Odwrotna_kolejnosc*.

Ćwiczenie 4. Wyświetlamy na ekranie wybrany element listy

1. Otwórz program *Elementy _listy* zapisany w ćwiczeniu 2.
2. Zmodyfikuj program, aby na ekranie wyświetlać tylko element *k*-ty, gdzie *k* jest liczbą całkowitą wprowadzaną z klawiatury po uruchomieniu programu.
3. Zapisz program w pliku pod nazwą *Wybrany _element.*

Wskazówka: Należy sprawdzić, czy wprowadzona wartość *k* jest większa lub równa 0 oraz mniejsza od *N*.

Warto zapamiętać

- Aby zaprogramować algorytm, w którym w trakcie działania programu musimy pamiętać wszystkie wprowadzone dane, możemy je zapisywać w liście.
- W języku Python nie określamy typu elementów listy. Interpreter określa typ elementów na podstawie rodzaju danych, jaki im przypiszemy.
- Do elementów listy, np. o nazwie *lista* i liczbie elementów *N*, odwołujemy się poprzez podanie nazwy i indeksu umieszczonego w nawiasach kwadratowych *lista*[0], *lista*[1], …, *lista*[*N* - 1].

Pytania i polecenia

1. Czym jest zmienna indeksowana?
2. W jakim celu definiujemy listę? Podaj przykład użycia listy.
3. Wyjaśnij na przykładach sposób definiowania listy w języku Python.

Zadania

1. Otwórz program *Elementy _listy* zapisany w ćwiczeniu 2. Zmodyfikuj program, aby na ekranie wyświetlić tylko pierwszy i ostatni element. Zapisz program w pliku pod nazwą *Element_pierwszy _i_ostatni*.
2. Otwórz program *Element_pierwszy _i_ostatni* zapisany w zadaniu 1. Zmodyfikuj program, aby sprawdzać, który element jest większy: pierwszy czy ostatni. Wyświetlaj tylko element większy oraz komunikat, który to element. W przypadku elementów równych wyświetlaj komunikat „Elementy równe". Zapisz program w pliku pod nazwą *Element_wiekszy*.

Temat 76.

Wykonywanie operacji na elementach listy w języku Python

1. Odwoływanie się do elementów listy
2. Zamiana miejscami wartości elementów listy

Warto powtórzyć

1. W jaki sposób w języku Python możemy określić liczbę elementów listy?
2. Dlaczego do wprowadzania danych do listy zdefiniowaliśmy funkcję niezwracającą wartości?
3. W jaki sposób wywołujemy w programie głównym funkcję niezwracającą wartości bez parametrów, a w jaki – z parametrami?
4. Na czym polegał problem dostaw artykułów przywożonych do magazynu RTV (temat 73.)?

1. Odwoływanie się do elementów listy

Zliczając telewizory przywożone do magazynu RTV (temat 73.), nie korzystaliśmy z oddzielnych zmiennych do pamiętania, ile sztuk telewizorów przywieziono w poszczególnych dostawach. Jakiej struktury danych powinniśmy użyć w programie, aby komputer pamiętał wprowadzone dane?



Do rozwiązania opisanego problemu zastosujemy listy. Jeśli wszystkie wprowadzone dane pamiętamy w liście (tu: liczby telewizorów z każdej dostawy), możemy wykonywać na nich różne operacje, np. odwoływać się do danej dostawy, wyświetlić wielkości wszystkich dostaw, wyświetlić na ekranie wielkość wybranej dostawy, wyświetlić dostawy w odwrotnej kolejności, zamienić miejscami wielkości dostaw, zsumować wybrane elementy czy uporządkować dostawy według wielkości (zastosowanie list w algorytmach porządkowania poznamy w dalszych tematach).

Przykład 1. Wprowadzamy wielkości dostaw telewizorów do listy

Na początku programu zdefiniujemy listę o nazwie *tv*. Liczbę elementów listy podamy za pomocą wcześniej zdefiniowanej stałej *T*.



Definicja:

```
T = 5
tv = [0] * T
```

oznacza zdefiniowanie listy składającej się z pięciu elementów (każdy równy zero) o indeksach od 0 do 4.

Następnie zdefiniujemy funkcję niezwracającą wartości bez parametrów *wprowadz_telewizory()*, wprowadzającą liczby telewizorów w kolejnych dostawach do listy.

Ponadto założymy, że dostawy będą numerowane od 1 (aby pierwszą dostawę określać jako „pierwszą ", a nie „zerową" itd.).

Dostawy_tv_lista.py - D:/Dysk C/Python/Dostawy_tv_lista.py (3.10.0)

File Edit Format Run Options Window Help

```
T = 5
tv = [0] * T

def wprowadz_telewizory():
    for i in range(T):
        print("Podaj liczbę telewizorów w", i + 1, "dostawie: ", end = "")
        tv[i] = int(input())
```

Ćwiczenie 1. Wprowadzamy wielkości dostaw telewizorów do listy i wyświetlamy je na ekranie

1. Utwórz nowy plik źródłowy. Zdefiniuj funkcję *wprowadz_telewizory()* wprowadzającą elementy (tu: liczby telewizorów w kolejnych dostawach) do listy (przykład 1.). Funkcję wywołaj w programie głównym. Dlaczego do wartości zmiennej *i* dodajemy 1?
2. Zdefiniuj funkcję *wyprowadz_telewizory*() wyprowadzającą elementy (tu: liczby telewizorów w kolejnych dostawach) z listy *tv* na ekran monitora. Funkcję wywołaj w programie głównym.
3. Zapisz program w pliku pod nazwą *Dostawy_tv_lista*. Uruchom i przetestuj program dla kilku różnych wartości elementów wprowadzanych do listy *tv*.

Ćwiczenie 2. Wyświetlamy na ekranie wielkość wybranej dostawy telewizorów

1. Uzupełnij program zapisany w ćwiczeniu 1. o wyświetlanie na ekranie liczby telewizorów z wybranej dostawy. Numer dostawy *k* wprowadzaj z klawiatury.
2. Zapisz program w pliku pod nazwą *Wybrana_dostawa*. Uruchom i przetestuj program dla kilku różnych wartości elementów listy *tv* i zmiennej *k*.

Wskazówka: Pamiętaj, że elementy listy są numerowane od zera, natomiast numeracja dostaw zaczyna się od 1 (takie przyjęliśmy założenie).

Ćwiczenie 3. Porównujemy wielkości dwóch dostaw

1. Do programu zapisanego w ćwiczeniu 2. dodaj sprawdzanie, która dostawa telewizorów jest większa: druga czy czwarta (numery dostaw zaczynają się od 1). Zależnie od wyniku, wyprowadź na ekran komunikat: „Większa jest druga dostawa" lub „Większa jest czwarta dostawa". W przypadku dostaw o tej samej liczbie telewizorów wyprowadź komunikat „Dostawy równe".
2. Zapisz program w pliku pod nazwą *Wieksza_dostawa*.

2. Zamiana miejscami wartości elementów listy

Okazało się, że w magazynie RTV pomylono półki podczas umieszczania telewizorów z danych dostaw. Jak zamienić te dostawy miejscami? Jak to zrobić w programie komputerowym, gdy wielkości dostaw wprowadzono do listy?

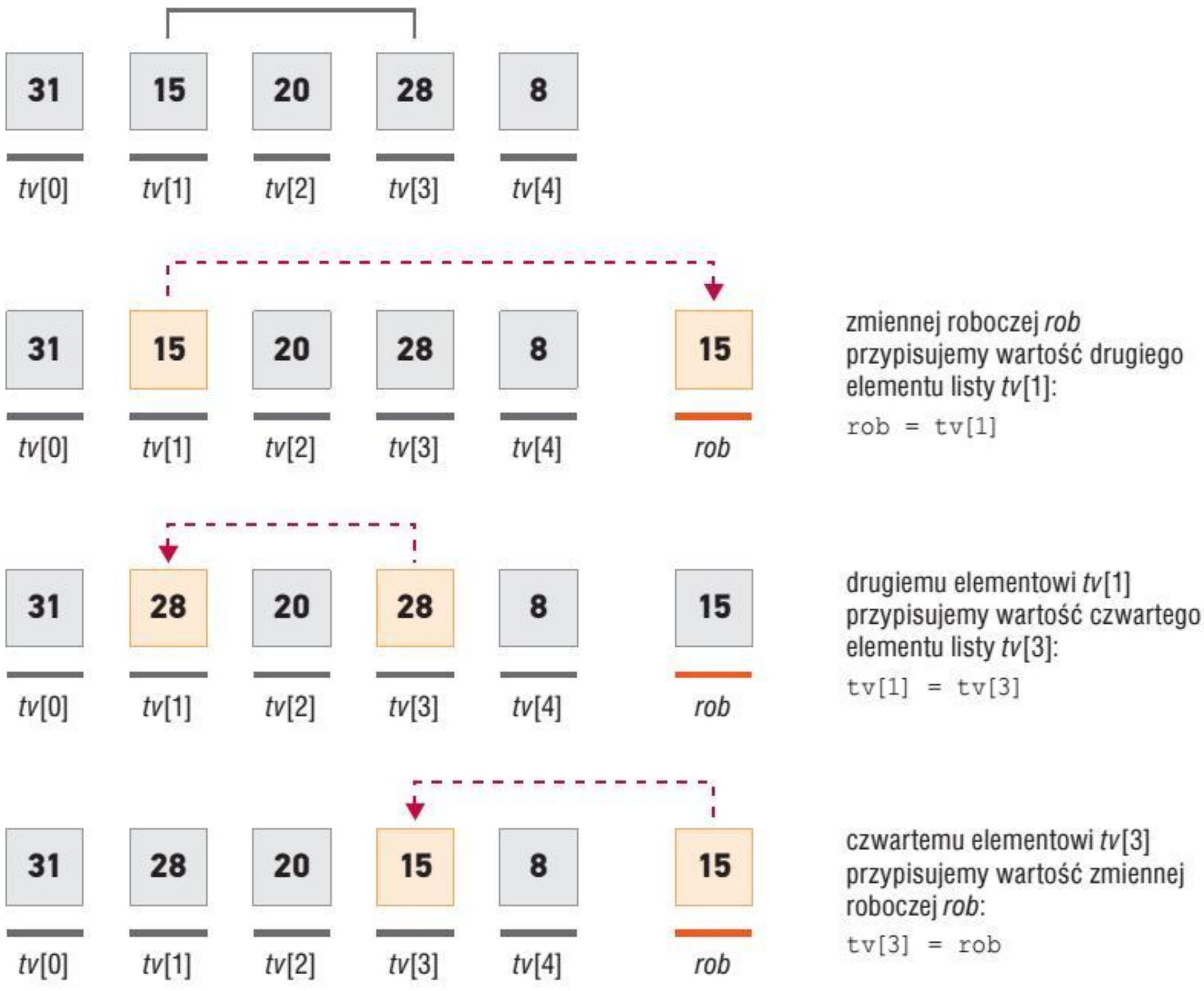
Aby przełożyć telewizory z półki A na półkę B, musimy telewizory z półki A gdzieś tymczasowo odłożyć, np. na półkę roboczą C, następnie telewizory z półki B przełożyć na półkę A, a te z półki roboczej C na B.

Podobnie postępujemy w przypadku programu komputerowego, w którym dane (wielkości poszczególnych dostaw) pamiętamy w liście – również używamy tzw. zmiennej roboczej do chwilowego zapamiętania „odłożonej" wartości danego elementu listy.

Przykład 2. Zamienianie miejscami wartości elementów listy

Na liście *tv*, w której pamiętamy dostawy telewizorów, chcemy zamienić miejscami drugą dostawę z czwartą, czyli wartości elementów listy o indeksach 1 i 3 (*tv*[1] i *tv*[3], jak na rysunku).

Aby zamienić miejscami elementy listy, posłużymy się zmienną roboczą *rob* (podobnie jak półką roboczą C). Przypisanie `rob = tv[1]` oznacza, że wartość zmiennej `rob` zmieni się na 15, ale jednocześnie wartość 15 pozostanie w elemencie *tv*[1]. Dopiero przypisanie `tv[1] = tv[3]` zmienia wartość *tv*[1] na 28.



Rys. 1. Zamiana miejscami wartości elementów tablicy

Wskazówki:

- Do analizy przykładu możesz posłużyć się odpowiednio przygotowanymi pomocami (np. kartkami z zapisanymi liczbami) lub zapisać dane na zwykłej tablicy i odpowiednio je wymazywać.
- Aby zamienić miejscami wartości elementów *tv*[1] z *tv*[3], należy umieścić w programie kolejno instrukcje:

```
rob = tv[1]
tv[1] = tv[3]
tv[3] = rob
```

! Uwaga

Zamianę elementów listy wykorzystamy do porządkowania elementów metodą bąbelkową (temat 92.).

Ćwiczenie 4. Zamieniamy miejscami wartości elementów listy

1. Przeanalizuj przykład 2. Dlaczego po zamianie miejscami elementów *tv*[1] i *tv*[3] w zmiennej *rob* jest nadal pamiętana wartość 15?
2. Zmodyfikuj program zapisany w ćwiczeniu 3, aby w przypadku, gdy czwarta dostawa jest większa od drugiej, zamienić je miejscami. Wyprowadź na ekran wszystkie dostawy po tej zmianie poprzedzone komunikatem „Dostawy po zamianie miejscami drugiej z czwartą, gdy czwarta jest większa od drugiej".
3. Zapisz program w pliku pod nazwą *Zamiana_dostaw*. Uruchom i przetestuj program dla kilku różnych wartości elementów listy *tv*.

Warto zapamiętać

- Zapisanie elementów w liście umożliwia m.in. wyświetlenie na ekranie wartości wybranego elementu listy, wartości elementów w odwrotnej kolejności, zamianę wartości elementów miejscami, zsumowanie wybranych elementów.
- Aby zamienić miejscami elementy listy, należy posłużyć się zmienną do tymczasowego przechowania jednego z elementów.

Pytania i polecenia

1. W jaki sposób odwołujemy się do wybranego elementu listy?
2. Wyjaśnij na przykładzie sposób zmiany miejscami wartości elementów listy.

Zadania

1. Zdefiniuj listę składającą się z pięciu elementów, będących liczbami godzin lekcyjnych w planie lekcji danej klasy od poniedziałku do piątku. Utwórz program wypisujący w kolumnie elementy listy, czyli liczbę lekcji każdego dnia. Zapisz program w pliku pod nazwą *Lekcje_w_tygodniu*.
   Wskazówka: Zastosuj sposób definiowania listy pokazany w pierwszym wierszu tabeli 1. z tematu 75.
2. Korzystając z listy zdefiniowanej w zadaniu 1., utwórz program, który dla podanego z klawiatury numeru dnia tygodnia wyświetli liczbę lekcji tego dnia. Zapisz program w pliku pod nazwą *Lekcje_w_dniu*.

Dla zainteresowanych

3. Napisz program, który wczytuje liczby całkowite do ośmioelementowej listy i wypisuje na ekran indeks pierwszego elementu listy, którego wartość wynosi zero. Jeśli nie ma takiego elementu, program powinien wypisać komunikat „Brak elementu zero”. Zapisz program w pliku pod nazwą *Indeks_elementu_zero*.

Temat 77.

Dane tekstowe w języku Python

1. Wprowadzanie i wyprowadzanie danych tekstowych
2. Łączenie napisów
3. Wyodrębnianie fragmentów tekstu
4. Używanie danych tekstowych

Warto powtórzyć

1. Na jakiej podstawie interpreter określa typ zmiennej w programie w języku Python?
2. Jakiego typu zmienne stosowaliśmy w programach?
3. Jakie są zasady dotyczące nazw zmiennych?

1. Wprowadzanie i wyprowadzanie danych tekstowych

Zapisując wybrany algorytm w języku programowania, m.in. dobieramy odpowiednie struktury danych. Jeśli algorytmy dotyczą działań na liczbach, stosujemy dane liczbowe, jeśli pracy na tekstach – tekstowe typy danych.

Jak wiemy, w języku Python nie deklarujemy zmiennych – zmienna przyjmuje typ wartości, która zostanie do niej przypisana. Na przykład instrukcja:

```
nazwisko = "Kowalski"
```

zmiennej *nazwisko* przypisze wartość, która jest łańcuchem tekstowym, czyli zmienna *nazwisko* będzie typu łańcuchowego `str`.

Typ tekstowy (`str`) reprezentuje ciągi danych znakowych (napisy, łańcuchy znaków).

Przykład 1. Wprowadzanie danych tekstowych w języku Python

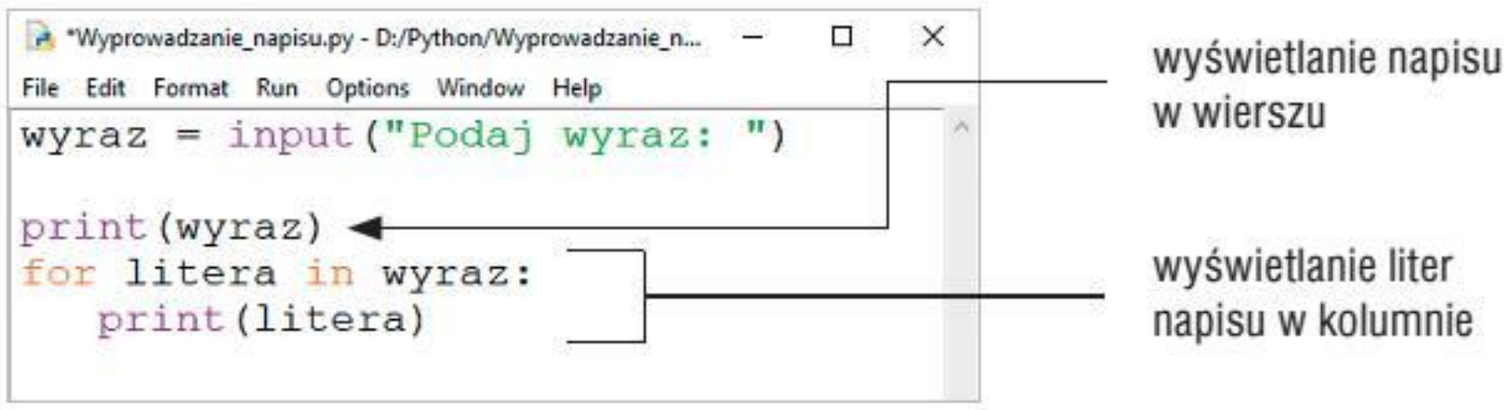
Dane tekstowe można wprowadzać z klawiatury:

```
napis = input("Wprowadź tekst: ")
```

Można również zastosować instrukcję przypisania:

```
napis = "akumulator"
```

Do wyprowadzenia napisu na ekran użyjemy (tak jak w przypadku liczb i komunikatów) funkcji `print()`. Możemy też litery wyprowadzić na ekran w kolumnie, używając instrukcji iteracyjnej **`for`** (w języku Pyton łańcuchy znaków są traktowane podobnie jak listy zawierające znaki).



Rys. 1. Wyprowadzenie napisu na ekran – w wierszu i kolumnie

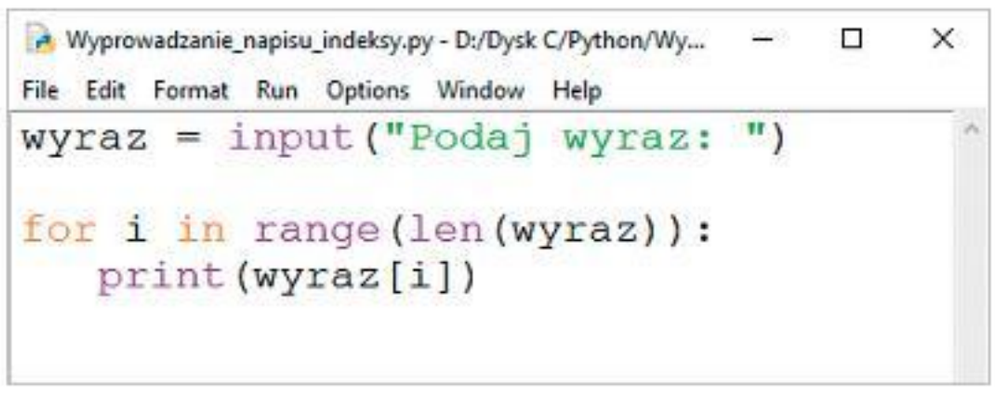
Ćwiczenie 1. Wyprowadzamy dane tekstowe na ekran

1. Napisz program wyprowadzający w wierszu wyraz wprowadzony z klawiatury, a następnie w kolumnie – litery tego wyrazu. Skorzystaj z rysunku 1.
2. Zapisz program w pliku pod nazwą *Wyprowadzanie_napisow*.

Aby w języku Python wyznaczyć długość łańcucha, możemy użyć standardowej funkcji `len()`. Na przykład `len(napis)` oznacza długość łańcucha pamiętanego w zmiennej *napis*.

W języku Python możemy odwoływać się do pojedynczego znaku łańcucha tak samo jak do elementu listy, czyli używając indeksu, np. *napis*[i]. Znaki łańcucha są numerowane od zera. Indeks ostatniego znaku obliczamy, odejmując 1 od długości napisu, np.:
`len(napis) - 1`.

Program pokazany na rysunku 2. również umożliwia wprowadzenie napisu z klawiatury i wyprowadzenie na ekran w kolumnie wszystkich jego liter.



Rys. 2. Wyprowadzenie liter napisu w kolumnie z wykorzystaniem zmiennych indeksowanych

Ćwiczenie 2. Wyprowadzamy dane tekstowe na ekran

1. Przepisz program pokazany na rysunku 2. Czym się różnią zapisy instrukcji **`for`** na rysunkach 1. i 2.?
2. Zapisz program w pliku pod nazwą *Wyprowadzanie_napisow_indeksy*. Uruchom i przetestuj program dla kilku różnych wartości zmiennej *wyraz*.
3. Zmodyfikuj program, aby w kolejnych wierszach wyświetlała się co druga litera wprowadzonego wyrazu (zaczynając od pierwszej litery).
4. Zapisz plik pod tą samą nazwą. Przetestuj program dla kilku różnych wartości zmiennej *wyraz*.

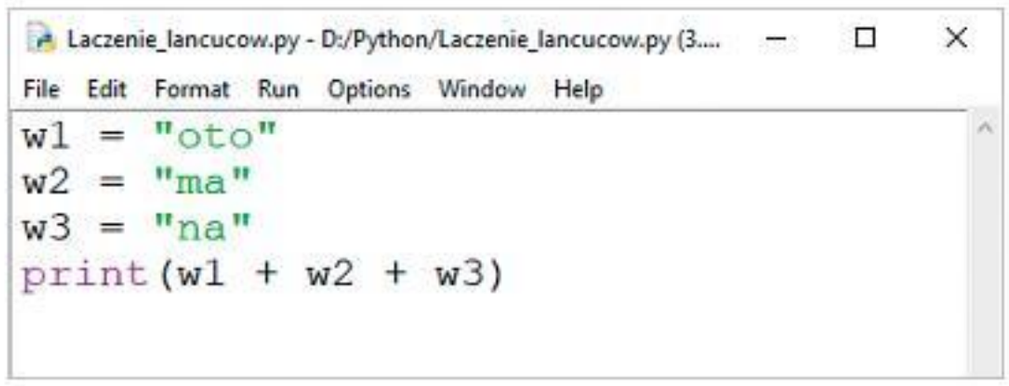
2. Łączenie napisów

Napisy możemy ze sobą łączyć, używając operatora +. Na przykład instrukcja:

```
print("sa" + "mo" + "chód")
```

wyprowadzi na ekran napis „samochód”.

Łączenie łańcuchów nazywamy **konkatenacją**. Możemy również łączyć łańcuchy, które wcześniej przypiszemy do zmiennych.



Rys. 3. Przykład łączenia napisów

Ćwiczenie 3. Łączymy napisy

1. Napisz program łączący dwa łańcuchy, które przypiszemy do zmiennych w programie (skorzystaj ze wzoru na rysunku 3., ale podaj inne napisy).
2. Zapisz program w pliku pod nazwą *Laczenie_lancuchow*.

3. Wyodrębnianie fragmentów tekstu

W języku Python możemy **wyodrębnić** fragment napisu, wskazując indeksy pierwszego i ostatniego znaku fragmentu i rozdzielając je dwukropkiem.

Aby w języku Python wyodrębnić fragment tekstu z łańcucha *napis*:

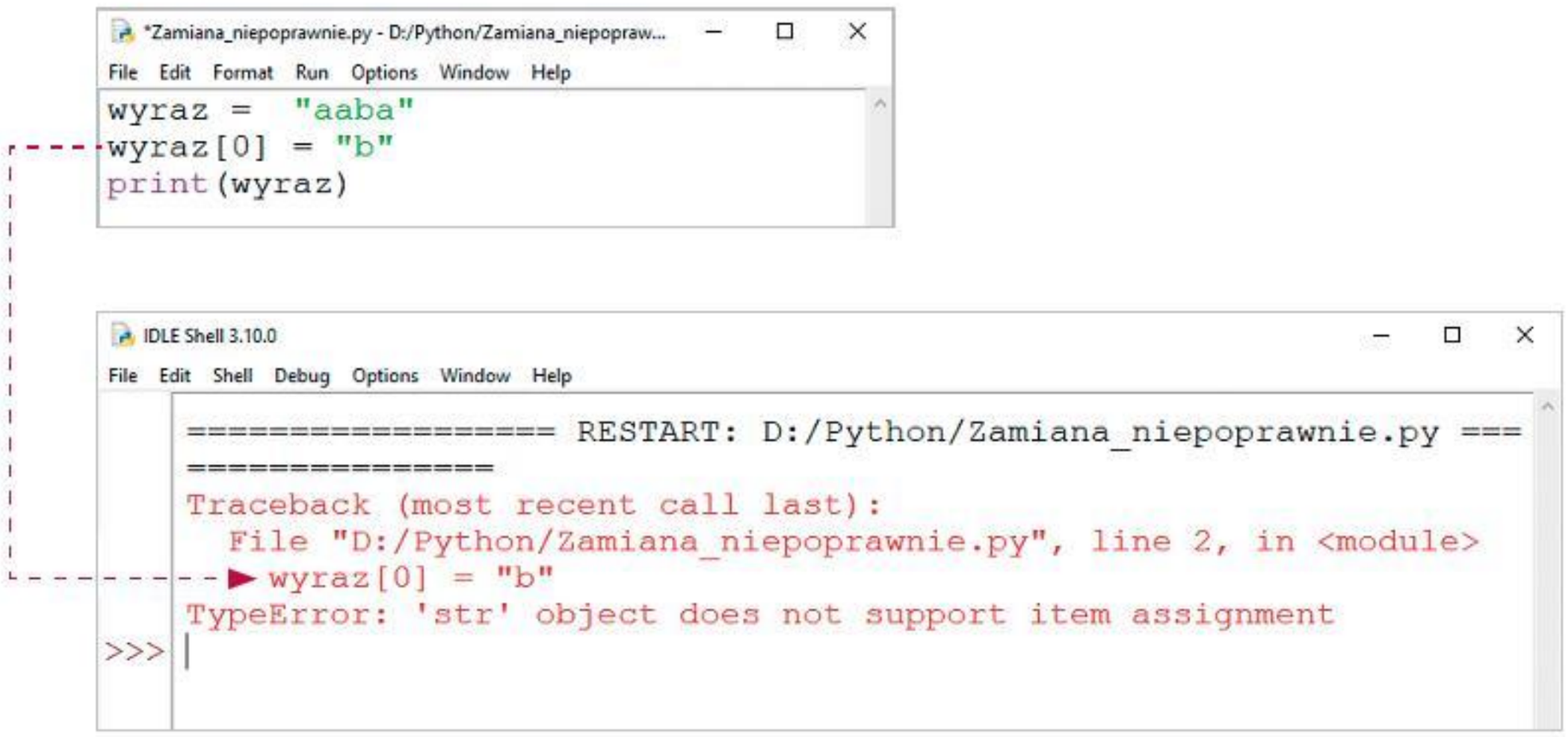
- od znaku o indeksie *indeks1* do znaku o indeksie *indeks2* – 1 (włącznie) stosujemy zapis: `napis[indeks1:indeks2]`,
- od pierwszego znaku tekstu do znaku o indeksie *indeks2* – 1 (włącznie) stosujemy zapis: `napis[:indeks2]`,
- od znaku o indeksie *indeks1* do końca tekstu stosujemy zapis: `napis[indeks1:]`.

Program pokazany na rysunku 4. wyświetla pierwszy znak tekstu (o indeksie 0) i dwa ostatnie znaki (fragment tekstu od znaku o indeksie 3 do znaku o indeksie 4).



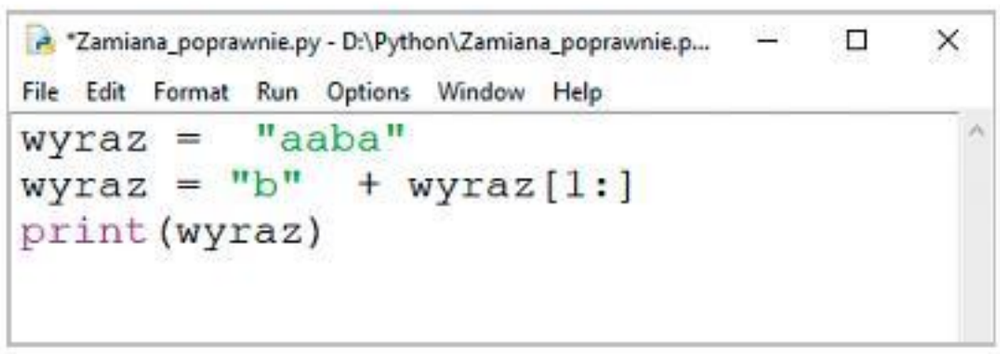
Rys. 4. Przykład wyodrębniania fragmentów tekstu

Załóżmy, że w wyrazie „aaba” chcemy zamienić pierwszą literę „a” na „b”. W języku Python nie możemy zamienić znaków w zmiennej typu tekstowego, używając instrukcji przypisania. W programie pokazanym na rysunku 5. interpreter wskaże błąd.



Rys. 5. Błędny przykład wyodrębniania fragmentów tekstu

Możemy poradzić sobie z tym problemem w następujący sposób:
zamiast instrukcji: `wyraz[0] = "b"`
możemy napisać: `wyraz = "b" + wyraz[1:]`,
co oznacza: zmiennej wyraz przypisz konkatencję łańcucha składającą się z litery „b” i fragmentu łańcucha *wyraz*, począwszy od drugiej litery do ostatniej.



Rys. 6. Przykład wyodrębniania fragmentów tekstu i zamiany liter

Ćwiczenie 4. Wyodrębniamy fragmenty tekstu

1. Zapoznaj się z zasadami wyodrębniania fragmentów tekstu w języku Python. Sprawdź je, pisząc krótkie programy podobne do pokazanych na rysunkach 4-6.
2. Zapisz programy w plikach, uruchom i przetestuj.

4. Używanie danych tekstowych

```
Wyraz.py - D:/Python/Wyraz.py (3.10.0)
File  Edit  Format  Run  Options  Window  Help
wyraz = input("Podaj wyraz: ")

for i in range(len(wyraz)):
    if wyraz[i] == "a":
        wyraz = wyraz[:i] + "b" + wyraz[i+1:]

print("Wyraz po przetworzeniu:", wyraz)

input("\n\nAby zakończyć, naciśnij Enter")
```

Rys. 7. Przykładowy program z wykorzystaniem danych tekstowych

Ćwiczenie 5. Analizujemy gotowy program

1. Przeanalizuj program pokazany na rysunku 7. Sformułuj treść zadania realizowanego przez ten program i napisz specyfikację zadania (określ dane i wyniki). Wyjaśnij działanie poszczególnych instrukcji.
2. Przepisz program, uruchom go i przetestuj dla kilku różnych danych.

Ćwiczenie 6. Stosujemy dane tekstowe

1. Napisz program wyprowadzający na ekran pierwszą i ostatnią literę wyrazu wprowadzonego z klawiatury.
2. Zapisz program w pliku pod nazwą *Wyraz_pierwsza_ostatnia*.
3. Przetestuj program dla kilku różnych danych.

Ćwiczenie 7. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 6. Program powinien zamieniać miejscami pierwszą literę z ostatnią, gdy są różne, a następnie wyprowadzić na ekran zmieniony wyraz. W przypadku jednakowych liter powinien się wyświetlić odpowiedni komunikat.
2. Zapisz program w pliku pod nazwą *Wyraz_zamiana_pierwsza_ostatnia*.
3. Przetestuj działanie programu dla kilku różnych danych.

Warto zapamiętać

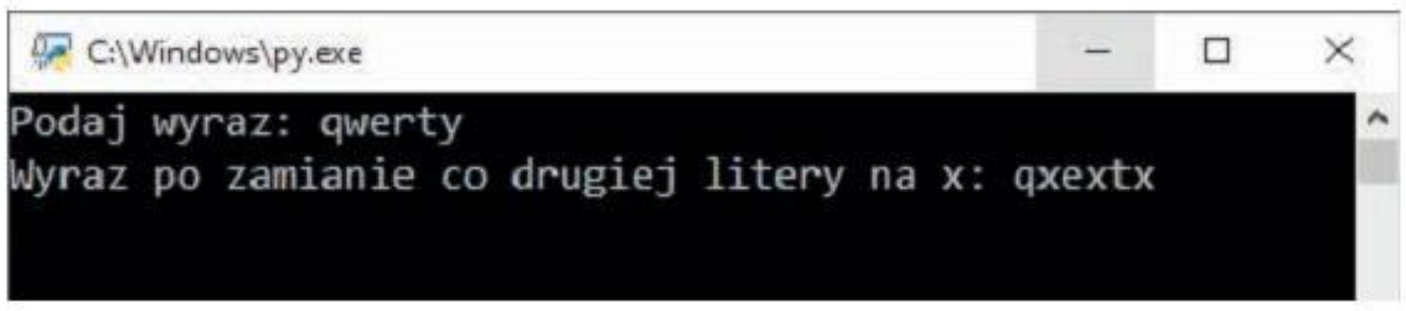
- Typy danych dobiera się do rozwiązywanego zadania, uwzględniając rodzaj informacji, które mają być przetwarzane, oraz sposób ich przetwarzania.
- W języku Python typ tekstowy (`str`) reprezentuje ciągi danych tekstowych (napisy, łańcuchy znaków).
- Do pojedynczego znaku łańcucha odwołujemy się tak samo, jak do elementu listy, czyli używając indeksu, np. *napis*[*i*].

Pytania i polecenia

1. Jakie dane w języku Python reprezentuje typ tekstowy (`str`)?
2. Jak odwołujemy się do pojedynczego elementu łańcucha? Pokaż na przykładzie.
3. W jaki sposób wyodrębnić fragment łańcucha w języku Python?

Zadania

1. Napisz program, który umożliwi zamianę wszystkich znaków „a” na „b” oraz wszystkich znaków „b” na „a” we wprowadzonym tekście. Skorzystaj z programu pokazanego na rysunku 7. Zapisz program w pliku pod nazwą *Zamiana*.
2. Napisz program zamieniający co drugą literę wyrazu wprowadzonego z klawiatury na literę „x” i wyprowadzający zmieniony wyraz na ekran. Zapisz program w pliku pod nazwą *Wyraz_zamiana_na_x*. Przetestuj działanie programu dla kilku różnych danych. **Wskazówka:** Zastanów się, którą instrukcję pętli najlepiej tu zastosować.



Rys. 8. Przykładowy wynik działania programu – zadanie 2.

3. Zadanie dla dwóch osób: Każda z osób projektuje zadanie według własnego pomysłu – formułuje treść zadania (można wzorować się na zadaniach z poznanych tematów). Następnie pisze specyfikację i przekazuje zadanie drugiej osobie, aby napisała program. Potem program wraca do autora zadania, który testuje go dla różnych danych.

Dla zainteresowanych

4. Zaproponuj własne zadanie, w którym zastosujesz dane tekstowe i poznane techniki: warunki i iteracje. Zapisz rozwiązanie w języku Python.

Rozdział XIV

Wybrane algorytmy i techniki algorytmiczne

Temat 78. Wybrane algorytmy na tekstach
Temat 79. Wyszukiwanie wzorca w tekście metodą naiwną
Temat 80. Szyfrowanie tekstu metodą podstawieniową
Temat 81. Szyfrowanie tekstu metodą przestawieniową
Temat 82. Algorytmy porządkowania liczb
Temat 83. Metoda połowienia
Temat 84. Metoda zachłanna
Temat 85. Rekurencja a iteracja
Temat 86. Wybrane algorytmy na liczbach
Temat 87. Działania na ułamkach z wykorzystaniem NWD i NWW

Temat 78. Wybrane algorytmy na tekstach

1. Porównywanie tekstów
2. Szukanie i zliczanie wystąpień wybranego znaku w tekście

Warto powtórzyć

1. Gdzie można spotkać się z porównywaniem tekstów?
2. W jaki sposób porównujemy dwa dokumenty tekstowe w edytorze tekstu Microsoft Word?

1. Porównywanie tekstów

Porównywanie tekstów i wyszukiwanie określonego znaku lub fragmentu tekstu to czynności, które wykonujemy zarówno podczas pracy nad tekstem pisanym na kartce (bez użycia komputera), jak i tworzonym w edytorze tekstu (z wykorzystaniem komputera). Najpierw wskażemy kilka przykładów tych operacji i poszukamy sposobów ich wykonywania – sformułujemy algorytmy. Następnie zapiszemy te algorytmy w wybranym języku programowania.

Nauczyciel języka angielskiego przygotował kartkówkę ze słówek. W jaki sposób może usprawnić sprawdzanie prac?

Nauczyciel przygotował wzorcowe rozwiązanie, z którym porównywał odpowiedzi uczniów. Przykład pokazano na rysunku 1.

Jan Kowalski, Klasa IIIb

tynkarz *plasterer*	spawacz *welder*
ratownik *livegard*	stolarz *carpenter*
pielęgniarka *nurse*	hydraulik *plumer*
geograf *geografer*	

Wzór odpowiedzi:

tynkarz – **plasterer**
ratownik – **lifeguard**
pielęgniarka – **nurse**
geograf – **geographer**
spawacz – **welder**
stolarz – **carpenter**
hydraulik – **plumber**

Rys. 1. Przykład porównywania słówek – kartkówka ucznia z języka angielskiego i wzór odpowiedzi przygotowany przez nauczyciela

Ćwiczenie 1. Szukamy rozwiązania problemu porównywania słówek

W tabeli 1. przedstawiono odpowiedzi ucznia z kartkówki ze słówek oraz poprawne odpowiedzi. Wskaż błędy w odpowiedziach ucznia. Wyjaśnij sposób porównywania słowa napisanego przez ucznia ze wzorem odpowiedzi. Czy musisz zawsze porównać wszystkie litery z odpowiedzi ucznia ze wzorem? Kiedy kończymy porównywanie?

Słówko w języku polskim	Odpowiedź ucznia	Odpowiedź poprawna
tynkarz	*plasterer*	plasterer
ratownik	*livegard*	lifeguard
pielęgniarka	*nurse*	nurse
geograf	*geografer*	geographer
spawacz	*welder*	welder
stolarz	*carpenter*	carpenter
hydraulik	*plumer*	plumber

Tabela 1. Zestawienie do ćwiczenia 1.

Ćwiczenie 2. Korzystamy z rozwiązania jednego problemu do rozwiązania innego problemu

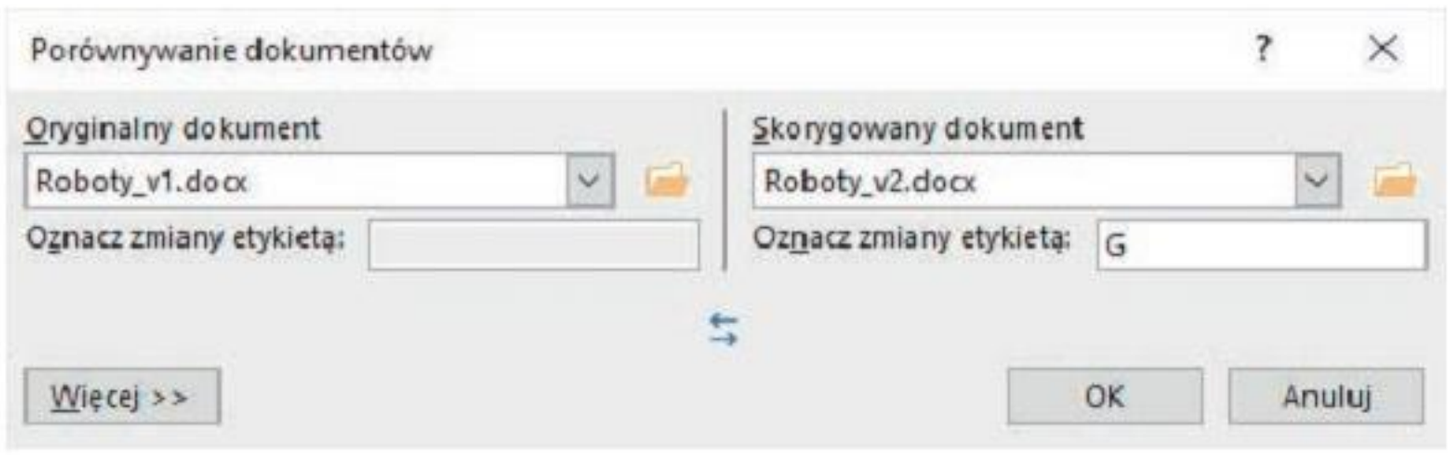
Pewien uczeń chce wysłać zaproszenia do zagranicznych gości. Zaadresował je odręcznie. Chcemy pomóc mu sprawdzić, czy nie popełnił błędu w zapisie nazwisk. W tabeli 2. pokazano poprawną formę zapisu oraz nazwiska, które widnieją na zaproszeniach. Jak sprawdzić, czy dane nazwisko zostało poprawnie zapisane? Czy można zastosować sposób z ćwiczenia 1.? Uzasadnij odpowiedź.

Zapis nazwiska na zaproszeniu	Poprawny zapis nazwiska
Kobayashi	Kobayashi
Rodrígez	Ródríguez
Yorck won Wartenburg	Yorck von Wartenburg
Bhaktwedanta	Bhaktiwedanta
Aberystwyth	Aberystwyth
von Kauniz-Rietberg	von Kaunitz-Rietberg

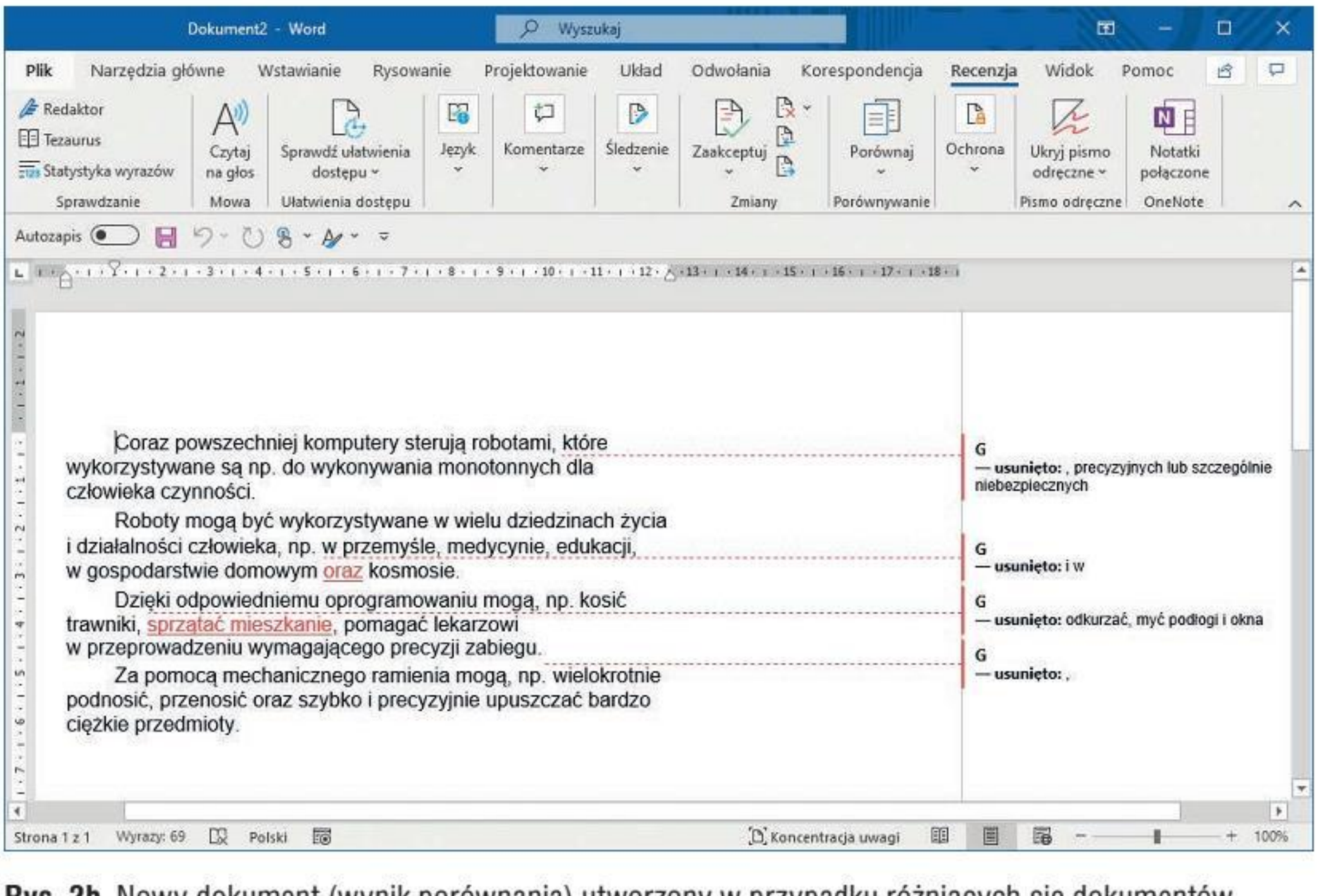
Tabela 2. Zestawienie do ćwiczenia 2.

Napisaliśmy tekst o robotach. Pracę sprawdziła inna osoba. Nie wiemy jednak, w których miejscach wprowadziła poprawki, ponieważ nie zaznaczyła ich w tekście. W jaki sposób można porównać oba teksty?

W przypadku dokumentów pisanych ręcznie musielibyśmy czytać wiersz po wierszu, aby znaleźć różnice. Edytory tekstu (m.in. Microsoft Word) mają funkcję porównywania dokumentów.



Rys. 2a. Porównywanie dokumentów w programie Microsoft Word



Rys. 2b. Nowy dokument (wynik porównania) utworzony w przypadku różniących się dokumentów

Ćwiczenie 3. Porównujemy dwa teksty w edytorze tekstu

1. Odszukaj w edytorze tekstu Microsoft Word opcję porównywania tekstów.
2. Porównaj dwa dokumenty tekstowe (zapisane w plikach, które otrzymasz od nauczyciela).

Porównywanie tekstów to w najprostszym ujęciu sprawdzenie (np. znak po znaku), czy teksty są takie same. Algorytmy porównywania mogą uwzględniać lub ignorować wielkość liter, np. „iteracja" i „Iteracja" – mogą być traktowane jako ten sam tekst (przy nieuwzględnianiu wielkości liter) lub jako dwa różne wyrazy (przy uwzględnianiu wielkości liter).

! Uwaga

W profesjonalnych rozwiązaniach stosuje się bardziej złożone algorytmy porównywania tekstów.

Algorytm porównywania dwóch tekstów (np. *tekst1* i *tekst2*) polega na porównywaniu kolejnych znaków, zaczynając od pierwszego znaku każdego z tekstów.

Sprawdzanie powtarzamy do momentu, gdy znaki znajdujące się w tekstach na tej samej pozycji będą różne lub do osiągnięcia końca jednego tekstu lub obu tekstów.

2. Szukanie i zliczanie wystąpień wybranego znaku w tekście

W pracy na temat robotów chcemy wyszukać znaki średnika i zamienić je na przecinki. W jaki sposób można to zrobić?

Jeśli w tekście mielibyśmy wyszukać znaki średnika tradycyjnie, przeglądalibyśmy tekst znak po znaku. Jeśli tekst wprowadziliśmy w edytorze tekstu, to znajdowanie znaków, ich zliczanie oraz zamianę jednego znaku na inny umożliwiają odpowiednie opcje programu.

Edytor Microsoft Word wyróżnia kolorem żółtym wszystkie znalezione znaki i podaje liczbę ich wystąpień.

Ćwiczenie 4. W dokumencie tekstowym szukamy znaku i zastępujemy go innym

W pliku otrzymanym od nauczyciela wyszukaj wystąpienia średnika (znaku ;). Sprawdź liczbę tych znaków w tekście. Zamień wszystkie średniki na przecinki, stosując odpowiednie opcje edytora.

Ćwiczenie 5. Szukamy wystąpień znaku w tekście

1. Wyszukaj i zlicz litery „a" w słowie „matematyka". Przeprowadź analizę rozwiązania, tworząc i uzupełniając tabelę podobną do tabeli 3. Odpowiedz na pytania:
   a. Ile razy powtarza się operacja porównania, a ile zliczania?
   b. Czy trzeba sprawdzać wszystkie znaki tekstu? Uzasadnij odpowiedź.
2. Sformułuj algorytm szukania wystąpień wybranego znaku.

Wskazówka: Przyjmij na początku *licznik* = 0.

Lp.	Pytanie	Odpowiedź	Działanie
1.	Czy pierwszy znak tekstu jest taki sam jak szukany znak?	NIE	Przejdź do sprawdzenia następnego znaku tekstu.
2.	Czy drugi znak tekstu jest taki sam jak szukany znak?	TAK	Zwiększ *licznik* o 1: *licznik* = *licznik* + 1. Przejdź do sprawdzenia następnego znaku tekstu.
3.			

Tabela 3. Przykładowa analiza rozwiązania – wzór tabeli do ćwiczenia 5.

Warto zapamiętać

- Algorytm porównywania dwóch tekstów może polegać na porównywaniu kolejnych znaków tych tekstów, od pierwszego znaku każdego z nich.
- Aby szukać danego znaku w tekście i zliczać jego wystąpienia, przeglądamy tekst, porównując jego wszystkie znaki z szukanym znakiem i zliczając każde wystąpienie.

Pytania i polecenia

1. Na czym polega algorytm porównywania tekstów? Wyjaśnij na przykładzie.
2. W jaki sposób można wyszukać i zliczać wybrany znak w tekście?

Zadania

1. Wyjaśnij, czy algorytm porównywania tekstów z ćwiczeń 1. i 2. można zastosować w następujących sytuacjach:
   a. sprawdzanie numeru konta, które wpisaliśmy do pola formularza bankowego,
   b. sprawdzanie adresu e-mail, który wpisaliśmy do pola adresu.
2. W pliku otrzymanym od nauczyciela wyszukaj wystąpienia wykrzyknika (znaku !). Sprawdź liczbę tych znaków w tekście. Zamień wszystkie wykrzykniki na kropki, stosując odpowiednie opcje edytora.

Dla zainteresowanych

3. Podaj dwa przykłady sytuacji, w których porównuje się teksty: jeden przykład bez użycia komputera, a drugi – z jego użyciem.
4. Przeprowadź analizę algorytmu szukania i zliczania wystąpień znaku w tekście na wybranym przez siebie słowie i znaku.

Temat 79. Wyszukiwanie wzorca w tekście metodą naiwną

1. Przykład szukania wzorca w tekście
2. Analiza algorytmu szukania wzorca w tekście

Warto powtórzyć

1. Jakie poznaliśmy algorytmy na tekstach?
2. W jaki sposób szukaliśmy wybranego znaku w tekście?

1. Przykład szukania wzorca w tekście

Napisaliśmy pracę domową z informatyki na temat robotów. Okazało się jednak, że w niektórych miejscach w skrócie „np." wstawiliśmy na końcu przecinek zamiast kropki. Jak szybko wyszukać i poprawić błędne zapisy?

Gdyby praca była zapisana ręcznie na kartce, podobnie jak podczas szukania pojedynczego znaku czytalibyśmy tekst wiersz po wierszu, aby znaleźć konkretne zapisy.

Jeśli pracę wykonaliśmy w edytorze tekstu, to znajdowanie ciągów znaków, ich zliczanie oraz zamianę na inne umożliwia ta sama opcja, która służy do wyszukania pojedynczego znaku.

Ćwiczenie 1. Szukamy w dokumencie tekstowym ciągu znaków i zastępujemy go innym

Uruchom program Microsoft Word. Korzystając z pliku otrzymanego od nauczyciela, wyszukaj w tekście wystąpienia ciągu znaków „np,". Popraw tekst, stosując odpowiednią opcję edytora tekstów.

Wyjaśnimy, na czym polega szukanie ciągu znaków, czyli szukanie **wzorca** w tekście.

W **Wzorzec**

Spójny ciąg znaków.

Przykład 1. Szukanie wystąpień wzorca w tekście

Chcemy znaleźć wystąpienia wzorca „mat” w słowie „matematyka”.

Szukanie zaczynamy od początku tekstu. Sprawdzamy, czy pierwszy znak tekstu jest taki sam jak pierwszy znak wzorca.

Jeśli znaki są różne, przesuwamy wzorzec o jedną pozycję w prawo względem tekstu. Jeśli znaki są takie same, sprawdzamy drugi znak tekstu i drugi znak wzorca, trzeci znak itd. – aż do sprawdzenia wszystkich znaków wzorca. Jeśli są różne, przesuwamy wzorzec o jedną pozycję w prawo.

W przypadku znalezienia wzorca, kolejne wyszukiwanie wzorca zaczynamy od następnego znaku tekstu. Może się zdarzyć, że wzorzec pojawia się w tekście wielokrotnie.



Rys. 1. Szukanie wystąpień wzorca w tekście – znaleziono pierwsze wystąpienie wzorca

Ćwiczenie 2. Szukamy wystąpień wzorca w tekście

1. Korzystając z przykładu 1., dokończ wyszukiwanie wystąpień wzorca „mat” w słowie „matematyka”. Wykonaj ćwiczenie, przygotowując odpowiednie pomoce dydaktyczne (kartoniki z literami).
2. Wykonaj jeszcze kilka podobnych praktycznych ćwiczeń, wyszukując inny wzorzec w wyrazie „matematyka” i w innych wyrazach. Sprawdź też przypadek, gdy nie ma żadnego wystąpienia danego wzorca.

Naiwne wyszukiwanie wzorca polega na porównywaniu kolejnych znaków tekstu i wzorca. Zaczynamy od sprawdzenia pierwszego znaku tekstu i pierwszego znaku wzorca: jeśli są zgodne, sprawdzamy zgodność drugiego znaku tekstu i drugiego znaku wzorca, gdy nie są zgodne, sprawdzamy zgodność drugiego znaku tekstu i pierwszego znaku wzorca itd. – aż do znalezienia wzorca.

2. Analiza algorytmu szukania wzorca w tekście

Analizę algorytmu szukania wzorca w tekście przeprowadzimy na przykładzie tekstu „matematyka” i wzorca „mat”.

Lp.	Pytanie	Odpowiedź	Działanie
1.	Czy pierwszy znak tekstu jest taki sam jak pierwszy znak wzorca?	TAK	Przejdź do sprawdzenia następnego znaku tekstu i wzorca.
2.	Czy drugi znak tekstu jest taki sam jak drugi znak wzorca?	TAK	Przejdź do sprawdzenia następnego znaku tekstu i wzorca.
3.	Czy trzeci znak tekstu jest taki sam jak trzeci znak wzorca?	TAK	Wzorzec znaleziono na pozycji 1. Zaczynamy szukanie następnego wystąpienia wzorca. Przesuwamy wzorzec o jedną pozycję w prawo względem tekstu.
4.	Czy drugi znak tekstu jest taki sam jak pierwszy znak wzorca?	NIE	Przesuwamy wzorzec o jedną pozycję w prawo.
5.			
6.			

Tabela 1. Przykładowa analiza rozwiązania dla słowa „matematyka” i wzorca „mat” – wzór tabeli do ćwiczenia 3.

Ćwiczenie 3. Analizujemy rozwiązanie

1. Przeprowadź analizę rozwiązania dla słowa „matematyka” i wzorca „mat”, tworząc i uzupełniając tabelę podobną do tabeli 1.
2. Odpowiedz na pytania:
   a. Ile jest wystąpień szukanego wzorca w danym tekście i na jakiej pozycji je znaleziono?
   b. Kiedy kończymy przeglądanie tekstu? Czy musimy sprawdzać ostatnie dwa znaki tekstu? Uzasadnij odpowiedź.
   c. Jakie powtarzające się operacje zauważasz w rozwiązaniu?

Warto zapamiętać

- W algorytmie naiwnego wyszukiwanie wzorca porównujemy kolejne znaki tekstu i wzorca. Najpierw sprawdzamy pierwszy znak tekstu i pierwszy znak wzorca: jeśli są zgodne, sprawdzamy zgodność drugiego znaku tekstu i drugiego znaku wzorca, gdy nie są zgodne, sprawdzamy zgodność drugiego znaku tekstu i pierwszego znaku wzorca itd. – aż do znalezienia wzorca.
- Wyszukiwanie wzorca w tekście kończymy, gdy liczba niesprawdzonych znaków danego tekstu jest mniejsza od długości wzorca.

Pytania i polecenia

1. W jaki sposób w edytorze tekstu wyszukujemy ciąg znaków w tekście?
2. Na czym polega naiwny algorytm wyszukiwania wzorca w tekście? Wyjaśnij na przykładzie.
3. W jakim przypadku dla wzorca o długości czterech znaków nie musimy sprawdzać ostatnich trzech znaków tekstu?

Zadania

1. Wyszukaj wzorzec „rok” w słowie „mmmrokmmrokmmmmmm”.
2. Przygotuj analizę rozwiązania zadania 1. podobną do przedstawionej w tabeli 1.

Dla zainteresowanych

3. Podaj przykłady sytuacji, w których porównuje się teksty: jeden przykład bez użycia komputera, a drugi z jego użyciem.
4. Przeprowadź analizę algorytmu szukania i zliczania wystąpień wybranego znaku w teście. Samodzielnie ustal wyraz i znak.

Temat 80. Szyfrowanie tekstu metodą podstawieniową

1. W jakim celu stosujemy szyfry?
2. Podstawowe pojęcia dotyczące szyfrowania
3. Szyfrowanie tekstu metodą podstawieniową na przykładzie szyfru Cezara

Warto powtórzyć

1. Jakie znasz przykłady szyfrowania informacji?
2. W jakim celu szyfruje się informacje?

1. W jakim celu stosujemy szyfry?

Wyobraźmy sobie, że nie mamy dostępu do nowych technologii. Jak przekazać ważną wiadomość, aby jej treść nie trafiła w niepowołane ręce?

Możemy zapisać wiadomość w taki sposób, aby tylko nadawca i odbiorca mogli ją zrozumieć – wykorzystamy w tym celu **szyfr**.

Na przykład wiadomość „JUTROIDZIEMYDOKINA" możemy zaszyfrować jako „QFGILRWARVNBWLPRMZ", wykorzystując schemat szyfrowania pokazany w tabeli 1.

S **Szyfr**

Algorytm szyfrowania i deszyfrowania.

A	Y	C	W	E	U	G	S	I	Q	K	O	M
Z	B	X	D	V	F	T	H	R	J	P	L	N

Tabela 1. Przykład prostego schematu szyfrowania

Ćwiczenie 1. Szyfrujemy wiadomości

1. Wyjaśnij, na czym polega schemat szyfrowania pokazany w tabeli 1.
2. Zaszyfruj wiadomość do koleżanki lub kolegi. Wykorzystaj schemat szyfrowania z tabeli 1.

Ćwiczenie 2. Deszyfrujemy wiadomość

1. Rozszyfruj wiadomość WLALYZXAVMRZLGIAVXRVQ, wykorzystując tabelę 1.
2. Wyjaśnij sposób rozszyfrowywania wiadomości.

Szyfrowanie S

Proces przetwarzania wiadomości na **szyfrogram**, pozwalający na ukrycie jej tekstu.

Deszyfrowanie D

Proces odtwarzania tekstu oryginalnego na podstawie treści szyfrogramu.

Szyfry stosuje się w celu utajnienia wiadomości, aby nie mogła zostać odczytana przez osobę postronną. Pierwsze szyfry wykorzystywano do przekazywania rozkazów i innych informacji wojennych, a czasami także... listów miłosnych. Z czasem sposoby szyfrowania stawały się coraz bardziej skomplikowane, aby utrudnić złamanie szyfru.

W dzisiejszych czasach szyfry wykorzystuje się zdecydowanie częściej niż kiedyś. Spotykamy się z nimi codziennie – służą na przykład do ochrony wrażliwych danych (dane osobowe, hasła dostępu, dane finansowe). Zupełnie nieświadomie korzystamy z szyfrowania, rozmawiając przez telefon komórkowy, na stronach internetowych czy wysyłając e-mail.

2. Podstawowe pojęcia dotyczące szyfrowania

Szyfrowanie to technika umożliwiająca bezpieczne przesyłanie wiadomości. Nadawca wiadomości chciałby mieć pewność, że nikt niepowołany nie pozna jej treści, ani też jej nie zmodyfikuje.

Niezaszyfrowaną wiadomość nazywamy **tekstem jawnym**, natomiast zaszyfrowaną – **szyfrogramem**.



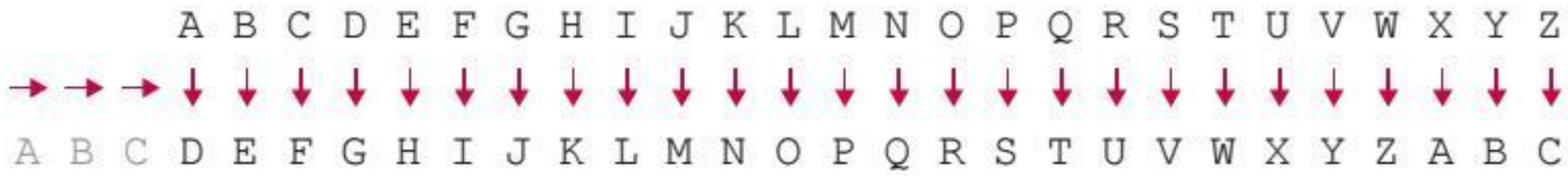
Rys. 1. Schemat procesu szyfrowania i deszyfrowania

Nauka zajmująca się zabezpieczaniem wiadomości to **kryptografia**, natomiast **kryptoanaliza** to nauka o łamaniu tych zabezpieczeń. Razem tworzą one **kryptologię** – osobną dziedzinę matematyki. Szyfrowanie i deszyfrowanie wykonuje się za pomocą odpowiednich operacji i funkcji matematycznych, często przy użyciu komputera.

3. Szyfrowanie tekstu metodą podstawieniową na przykładzie szyfru Cezara

Szyfr podstawieniowy to taki, w którym każdy znak tekstu jawnego jest zastępowany innym znakiem. Najbardziej znanym przykładem szyfru podstawieniowego jest **szyfr Cezara**.

Autorstwo szyfru Cezara przypisuje się Juliuszowi Cezarowi. Szyfr ten polega na zastąpieniu każdej litery tekstu jawnego znakiem stojącym w alfabecie o trzy pozycje w prawo względem znaku źródłowego. I tak A zostaje zastąpione przez D, B przez E, ... W przez Z, X przez A, Y przez B i Z przez C. Na rysunku 2. pokazano szyfr Cezara dla alfabetu łacińskiego, który zawiera 26 liter.



Rys. 2. Szyfr Cezara

Szyfr Cezara jest szyfrem **przesuwającym**, w którym wykorzystano liczbę 3 do określenia przesunięcia względem znaku wejściowego. W tej metodzie szyfrowania można zastosować liczbę inną niż 3. W ten sposób dla tego samego tekstu jawnego możemy uzyskać różne szyfrogramy, w zależności od tego, jaką liczbę zastosujemy (tabela 2.). Aby odszyfrować tekst, odbiorca powinien znać liczbę określającą przesunięcie, pełniącą tym samym funkcję **klucza** szyfru (rys. 3.).

Tekst jawny	Klucz	Szyfrogram
KOTEK	3	NRWHN
KOTEK	5	PTYJP
KOTEK	13	XBGRX

Tabela 2. Przykład wiadomości zaszyfrowanych metodą podstawieniową (szyfr przesuwający) z różnymi kluczami

K **Klucz**

Informacja służąca do szyfrowania lub deszyfrowania tekstu.



Rys. 3. Schemat procesu szyfrowania i deszyfrowania z kluczem

Ćwiczenie 3. Szyfrujemy wyraz szyfrem Cezara

1. Zaszyfruj napis MATEMATYKA szyfrem Cezara.
2. Wyjaśnij zastosowany sposób szyfrowania.

Ćwiczenie 4. Odszyfrowujemy słowa zapisane metodą Cezara

1. Na podstawie wskazówek przedstawionych na rysunku 2. rozszyfruj słowa:
   a. VCBIU
   b. LQIRUPDWBND
   c. NUBSWRJUDILD.
2. Wyjaśnij zastosowany sposób rozszyfrowywania.

Ćwiczenie 5. Szyfrujemy zdanie szyfrem Cezara

Korzystając z szyfru Cezara, zaszyfruj zdanie: „Szyfr Cezara to najbardziej znany szyfr podstawieniowy".

Warto zapamiętać

- Aby zabezpieczać informacje, można posługiwać się różnymi algorytmami szyfrującymi, m.in. algorytmem podstawieniowym.
- Szyfr, w którym każdy znak tekstu jawnego jest zastępowany innym znakiem, nazywamy szyfrem podstawieniowym. Najbardziej znany przykład szyfru podstawieniowego to szyfr Cezara.

Pytania i polecenia

1. Czym jest szyfr?
2. Czym różni się tekst jawny od szyfrogramu?
3. Czym zajmuje się kryptografia, a czym kryptoanaliza?
4. W jaki sposób działają techniki szyfrowania wiadomości? Podaj kilka przykładów.
5. Wyjaśnij na wymyślonym przez siebie przykładzie, na czym polega szyfrowanie tekstu metodą Cezara.

Zadania

1. Odszyfruj tekst: „VCBIUFHCDUDMHVWSURVWB".
2. Korzystając z metody szyfrowania podstawieniowego, zaszyfruj swoje dane osobowe (imię, nazwisko, miejsce zamieszkania).
3. Zadanie do wykonania w grupie dwuosobowej. Każda z osób zaszyfrowuje tekst szyfrem Cezara, następnie przekazuje zaszyfrowany tekst drugiej osobie do odszyfrowania.

Dla zainteresowanych

4. Steganografia to nauka o ukrywaniu informacji wewnątrz innej informacji. Zapoznaj się z tym zagadnieniem, korzystając z dodatkowej literatury i źródeł internetowych.

Temat 81.

Szyfrowanie tekstu metodą przestawieniową

1. **Szyfrowanie tekstu metodą przestawieniową**
2. **Miniszyfrowanie – tworzenie anagramów**
3. **Więcej o szyfrowaniu**
   - 3.1. Szyfrowanie symetryczne i asymetryczne
   - 3.2. Maszyna Enigma
   - 3.3. Algorytm RSA

Warto powtórzyć

1. Czym jest tekst jawny, a czym szyfrogram?
2. Na czym polega szyfr Cezara?
3. Jaką rolę pełni klucz w szyfrze Cezara?

1. Szyfrowanie tekstu metodą przestawieniową

Zanim poznamy kolejny sposób szyfrowania, wykonajmy ćwiczenie 1.

Ćwiczenie 1. Odszyfrowujemy cytaty

1. Przeanalizuj sposób szyfrowania pokazany w tabeli 1. Wyjaśnij, na czym polega ta metoda.
2. Korzystając z przykładów z tabeli 1., odszyfruj cytaty, posługując się podanym kluczem:
   a. NAWOEIWATJEERERMJAJZOŚZSEŻĆYAKNDDMI (klucz 5/7),
   b. NOAZSUIWMETSEOJRTŁCKEAWYHOSZYSSCZZCZŁHCUHĘ (klucz 6/7).

Uwaga

Autorzy cytatów:
a. Heraklit z Efezu,
b. Adam Mickiewicz („Pan Tadeusz").

Tekst jawny	Klucz	Zapis wierszami	Szyfrogram
ALA MA KOTA	3/3	A L A M A K O T A	AMOLATAKA
SZKOŁA PONADPODSTAWOWA	3/7	S Z K O Ł A P O N A D P O D S T A W O W A	SOSZNTKAAODWŁPOAOWPDA

Tabela 1. Przykłady wiadomości zaszyfrowanych metodą przestawieniową – ćwiczenie 1.

W ćwiczeniu 1. posłużyliśmy się szyfrem przestawieniowym, który najprościej zrealizować, zapisując tekst jawny wierszami o ustalonej długości, a czytając go kolumnami (przy ustalonej długości wierszy powstają kolumny o określonej liczbie znaków). Kluczem może być liczba wierszy lub kolumn (w przykładach powyżej podane zostały obie te liczby) – wszystko zależy od ustaleń.

Szyfr przestawieniowy to taki, w którym w tekście zaszyfrowanym pojawiają się wszystkie znaki tekstu jawnego, ale w innej kolejności.

Uwaga

„To be or not to be – that is the question”

– W. Szekspir, Hamlet, akt III, scena 1.

Ćwiczenie 2. Stosowanie szyfru przestawieniowego

Zaszyfruj cytat „To be or not to be – that is the question”, stosując szyfrowanie przestawieniowe. Ustaw znaki tekstu jawnego w dziesięciu kolumnach.

Wskazówka: W szyfrowaniu przestawieniowym pomijamy spacje. Tekst jawny: TOBEORNOTTOBETHATISTHEQUESTION

2. Miniszyfrowanie – tworzenie anagramów

Anagram tworzymy poprzez przestawienie liter lub sylab wyrazu bądź zdania, wykorzystując wszystkie litery (głoski lub sylaby) wyrazu bądź zdania źródłowego.

Najprostszy anagram można utworzyć, układając litery w odwrotnej kolejności, np. *baobab – baboab*. Można też przestawić tylko sylaby, co np. w wyrazie *ma·ra*, daje anagram: *ra·ma*.

Możemy w ten sposób utworzyć szyfr, którego działanie będzie polegało na przykład na wyprowadzeniu wyrazu (ciągu znaków) wspak, np. „poezja” – jako „ajzeop”.

Ten rodzaj szyfru to też szyfrowanie przestawieniowe – polega na zmianie kolejności znaków tekstu.

Ćwiczenie 3. Tworzymy anagramy i szukamy sposobu utworzenia anagramu

1. W kilku grupach twórzcie różne anagramy, np. jedna osoba wymyśla anagram słowa, a pozostałe osoby z grupy próbują odgadnąć, jakie to słowo oraz jaki zastosowano klucz (sposób, w jaki utworzono anagram).
2. Przedstawcie wybrane rozwiązania na forum klasy.

3. Więcej o szyfrowaniu

3.1. Szyfrowanie symetryczne i asymetryczne

Podczas szyfrowania danych z kluczem możemy spotkać się z dwoma rodzajami algorytmów szyfrujących: **symetrycznym** i **asymetrycznym**.

Szyfrowanie symetryczne wykorzystuje jeden klucz – ten sam zarówno do szyfrowania, jak i deszyfrowania wiadomości, np. dla szyfru przestawieniowego kluczem jest liczba kolumn.

W **algorytmach asymetrycznych** wiadomość dla danego adresata zaszyfrować może każdy, używając **klucza publicznego** tego adresata. Natomiast odczytanie wiadomości możliwe jest tylko przy użyciu innego klucza – **klucza prywatnego**, który zna jedynie odbiorca. Z algorytmów asymetrycznych korzysta się np. przy uwierzytelnianiu transakcji bankowych czy w certyfikatach bezpieczeństwa stron HTTPS. Popularne protokoły szyfrowania, które zapobiegają przechwytywaniu i zmienianiu przesyłanych danych, wykorzystywane m. in. na stronach internetowych, to SSL lub TLS.



Pamiętaj, aby sprawdzać, czy strona, na której podajesz dane wrażliwe, tj. hasło lub dane karty kredytowej, np. strona płatności w sklepie internetowym, wykorzystuje aktualny certyfikat szyfrowania HTTPS (znak kłódki lub *https://* w adresie strony). Jeśli strona go nie ma, twoje dane mogą zostać przechwycone podczas ich przesyłania.

3.2. Maszyna Enigma

Metodę podstawieniową wykorzystano w słynnej maszynie szyfrującej Enigma, która w sposób mechaniczny dokonuje kolejnych podstawień alfabetycznych. Podstawowy model maszyny Enigma wynalazł Arthur Scherbius w Berlinie w 1918 roku. Szyfrowała ona wiadomość, wykonując kolejno pewną liczbę podmian znaków i wykorzystując przy tym połączenia elektryczne. Jej najważniejszymi elementami były obracające się na jednej osi bębny szyfrujące.

W okresie międzywojennym Enigmą zainteresowało się niemieckie wojsko, które po zmodyfikowaniu jej konstrukcji i zasad działania wprowadziło maszynę do własnego użytku.



Rys. 1. Maszyna Enigma

W kolejnych latach w różnych krajach trwały prace nad złamaniem niemieckich szyfrów. Panowało przekonanie, że najlepsi w tej dziedzinie będą językoznawcy i lingwiści. Polacy jako pierwsi zatrudnili w tym celu profesjonalnych matematyków, byli to Jerzy Różycki, Henryk Zygalski oraz Marian Rejewski, którym w 1932 roku udało się złamać szyfr Enigmy. Polacy mogli odczytywać korespondencję niemiecką, jednak nie było to proste, bowiem Niemcy stale udoskonalali maszynę i procedury szyfrowania. Podczas II wojny światowej łamaniem szyfrów Enigmy zajmował się w Wielkiej Brytanii zespół matematyków kierowany przez Alana Turinga, jednego z pionierów informatyki.

3.3. Algorytm RSA

Jednym z aktualnie najpopularniejszych asymetrycznych algorytmów szyfrowania jest algorytm RSA. Jego nazwa pochodzi od pierwszych liter nazwisk jego twórców: Rona Rivesta, Adiego Shamira oraz Leonarda Adlemana, którzy zaprojektowali go w 1977 roku.

Jest to pierwszy algorytm, który może być stosowany zarówno do szyfrowania, jak i do podpisów elektronicznych.

W celu wygenerowania pary kluczy prywatnego i publicznego w algorytmie stosuje się działania i funkcje matematyczne na dużych liczbach pierwszych.

Obecnie RSA stosuje się na przykład w systemie międzynarodowych przekazów bankowych SWIFT oraz w przeglądarkach internetowych. Również wspomniane wcześniej certyfikaty bezpieczeństwa stron HTTPS bazują na tego typu algorytmie.

Warto zapamiętać

- Szyfr, w którym w tekście zaszyfrowanym pojawiają się wszystkie znaki tekstu jawnego, ale w innej kolejności, nazywamy szyfrem przestawieniowym.
- Gdy w wyrazie lub zdaniu przestawimy litery lub sylaby, wykorzystując wszystkie litery (głoski lub sylaby) tego wyrazu lub zdania – powstaje anagram.
- Szyfrowanie symetryczne wykorzystuje jeden klucz – ten sam zarówno do szyfrowania, jak i deszyfrowania wiadomości. W algorytmach asymetrycznych wiadomość zaszyfrować może każdy, używając klucza publicznego. Wiadomość można odczytać przy użyciu klucza prywatnego, który zna jedynie odbiorca.

Pytania i polecenia

1. Wyjaśnij na wymyślonym przez siebie przykładzie, na czym polega szyfrowanie przestawieniowe.
2. Czym jest anagram?
3. Czym się różni szyfrowanie symetryczne od asymetrycznego?
4. Do czego służyła maszyna Enigma? Która metoda szyfrowania była w niej wykorzystywana?

Zadania

1. Korzystając z metody szyfrowania przestawieniowego, zaszyfruj swoje dane osobowe (imię, nazwisko, miejsce zamieszkania).
2. Zadanie do wykonania w grupie dwuosobowej. Każda z osób wymyśla własny algorytm szyfrowania przestawieniowego, następnie przekazuje zaszyfrowany tekst drugiej osobie do odszyfrowania.
   Wskazówka: Można podać drugiej osobie algorytm szyfrowania.

Dla zainteresowanych

3. Korzystając z Internetu i dodatkowej literatury, znajdź informacje na temat Enigmy.
   Wskazówka: Zadanie można wykonać w grupie kilku osób, które następnie mogą w dyskusji podzielić się wiedzą.
4. Korzystając z Internetu i dodatkowej literatury, dowiedz się więcej na temat algorytmu z kluczem jawnym RSA.

Temat 82. Algorytmy porządkowania liczb

1. Porządkowanie metodą bąbelkową
2. Porządkowanie przez wstawianie

Warto powtórzyć

1. Czemu powinien być podporządkowany wybór algorytmu?
2. Podaj dwa przykłady porządkowania elementów z życia codziennego.
3. Według jakich kryteriów można uporządkować pliki w komputerze?

1. Porządkowanie metodą bąbelkową

Sprzedawca chce ustawić towar na sklepowych półkach tak, aby na najwyższej stał najdroższy produkt, a na najniższej – najtańszy. Jak może uporządkować ceny od najwyższej do najniższej?



Z porządkowaniem elementów spotykamy się często w życiu codziennym. Na przykład porządkujemy wyniki zawodów sportowych od najwyższych do najniższych, oceny z informatyki od najniższych do najwyższych czy alfabetycznie nazwiska znanych malarzy. Ustawiamy według wielkości pudełka jedno na drugim od największego do najmniejszego.

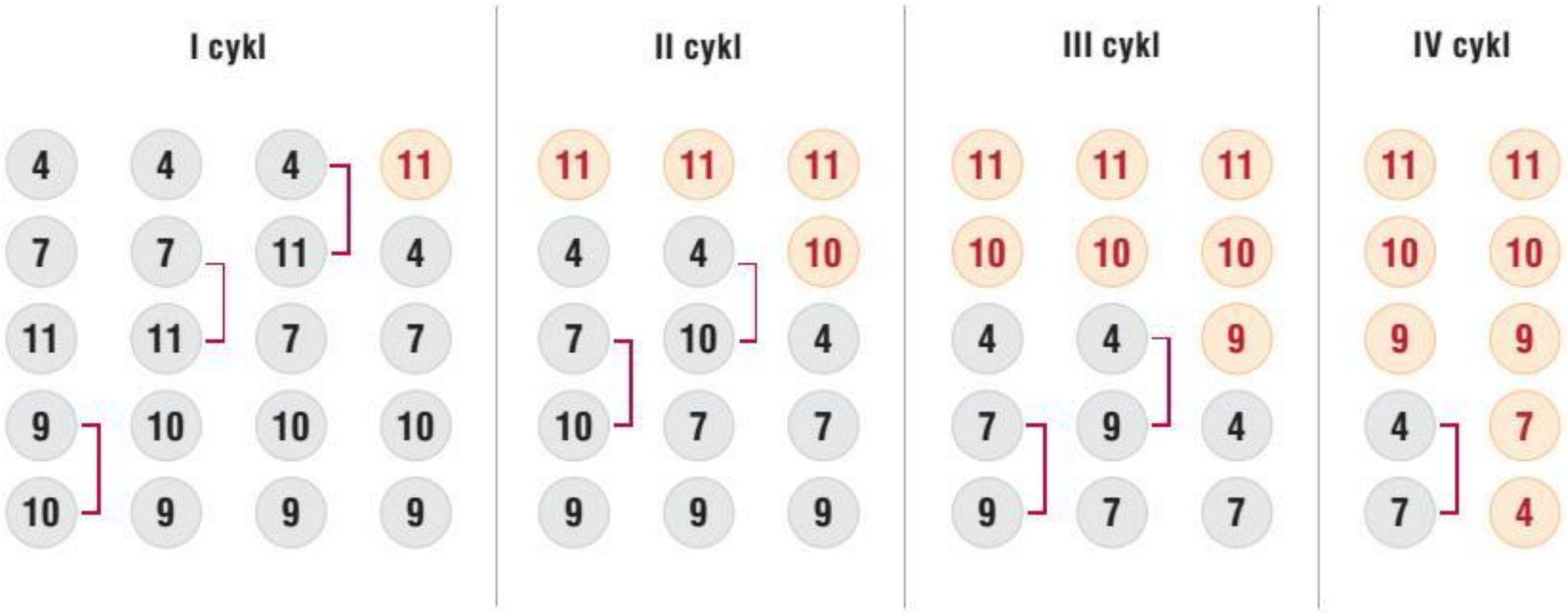
Na danych uporządkowanych szybciej wykonuje się różne operacje, np. wyszukiwanie (o czym przekonamy się w temacie 83.). W tym temacie zajmiemy się porządkowaniem ciągów liczb.

Elementy ciągu liczb możemy uporządkować (w terminologii informatycznej – **posortować**), stosując wybrany algorytm porządkowania (sortowania), np. **metodę bąbelkową** lub **przez wstawianie**.

Metodę bąbelkową wyjaśnimy na przykładzie porządkowania pięciu liczb naturalnych (np. będących cenami).

W algorytmie sortowania metodą bąbelkową w celu uporządkowania ciągu liczb – w tym przypadku **malejąco** – porównujemy sąsiednie liczby, rozpoczynając od porównania ostatniej i przedostatniej. Jeśli ostatnia liczba jest większa od przedostatniej, zamieniamy je miejscami. Następnie liczbę, która stała się przedostatnią, porównujemy z trzecią od końca i przestawiamy, jeśli jest większa itd.

Na rysunku 1. pokazano przykład sortowania bąbelkowego pięciu liczb (4, 7, 11, 9, 10). Sortowanie przebiega od dołu ku górze, w cyklach. Po pierwszym cyklu ciąg nie jest jeszcze uporządkowany. Należy wykonywać kolejne cykle, aż ciąg zostanie uporządkowany i nie trzeba już będzie przestawić żadnego elementu.



Rys. 1. Przykład zastosowania algorytmu sortowania bąbelkowego do uporządkowania ciągu liczb – malejąco (od największej do najmniejszej)

Nietrudno zauważyć, że już w pierwszym cyklu sortowania największa liczba (w tym przypadku liczba 11) zostaje ustawiona na właściwym miejscu. Aby ustawić we właściwej kolejności pozostałe liczby, należy powtarzać cykl sortowania, dopóki nie zostaną wykonane wszystkie przestawienia.

Rysunek 1. jest poglądową ilustracją pokazującą działanie algorytmu sortowania bąbelkowego. Elementy o większych wartościach są wynoszone wyżej, co przypomina bąbelki powietrza (większe szybciej przemieszczają się ku górze) – stąd też wzięła się nazwa algorytmu.

Oczywiście tą metodą możemy uporządkować zbiór, zaczynając od góry, czyli od porównania elementu pierwszego z drugim, drugiego z trzecim itd.



W algorytmie sortowania bąbelkowego porównujemy parami kolejne elementy zbioru i przestawiamy je, jeśli są ustawione w niewłaściwej kolejności. Zbiór jest przeglądany w tym samym kierunku tak długo, jak długo występuje w nim para elementów ustawionych w niewłaściwym porządku.

Ćwiczenie 1. Stosujemy sortowanie bąbelkowe

1. Uporządkuj metodą bąbelkową malejąco (od największej do najmniejszej) ciąg liczb (34, 87, 45, 12, 11, 0, -56, -28, -11, 72). Podaj liczbę porównań wykonanych w celu uporządkowania tego ciągu.
2. Wykonaj podobne praktyczne ćwiczenie, porządkując inny ciąg liczb. Podaj liczbę porównań wykonanych w celu uporządkowania tego ciągu.

Wskazówki:

- Przygotuj kartki z zapisanymi przykładowymi liczbami.
- Ułóż kartki na stole lub na podłodze i przestawiaj parami. Zacznij od ostatniej.

2. Porządkowanie przez wstawianie

Metodę sortowania (porządkowania) **przez wstawianie** (inaczej **przez umieszczanie**) stosujemy często w życiu codziennym – na przykład gdy w bibliotece odkładamy książkę na półkę, znajdujemy odpowiednie miejsce, patrząc na nazwiska autorów książek umieszczonych na półce.

W metodzie sortowania przez wstawianie porządkowany ciąg dzielimy na dwie części:

- uporządkowaną (na początku działania algorytmu nie zawiera ona żadnych elementów),
- nieuporządkowaną (na początku zawiera wszystkie elementy ciągu).

Z części nieuporządkowanej pobieramy kolejne elementy i umieszczamy je w odpowiednich miejscach w części uporządkowanej.

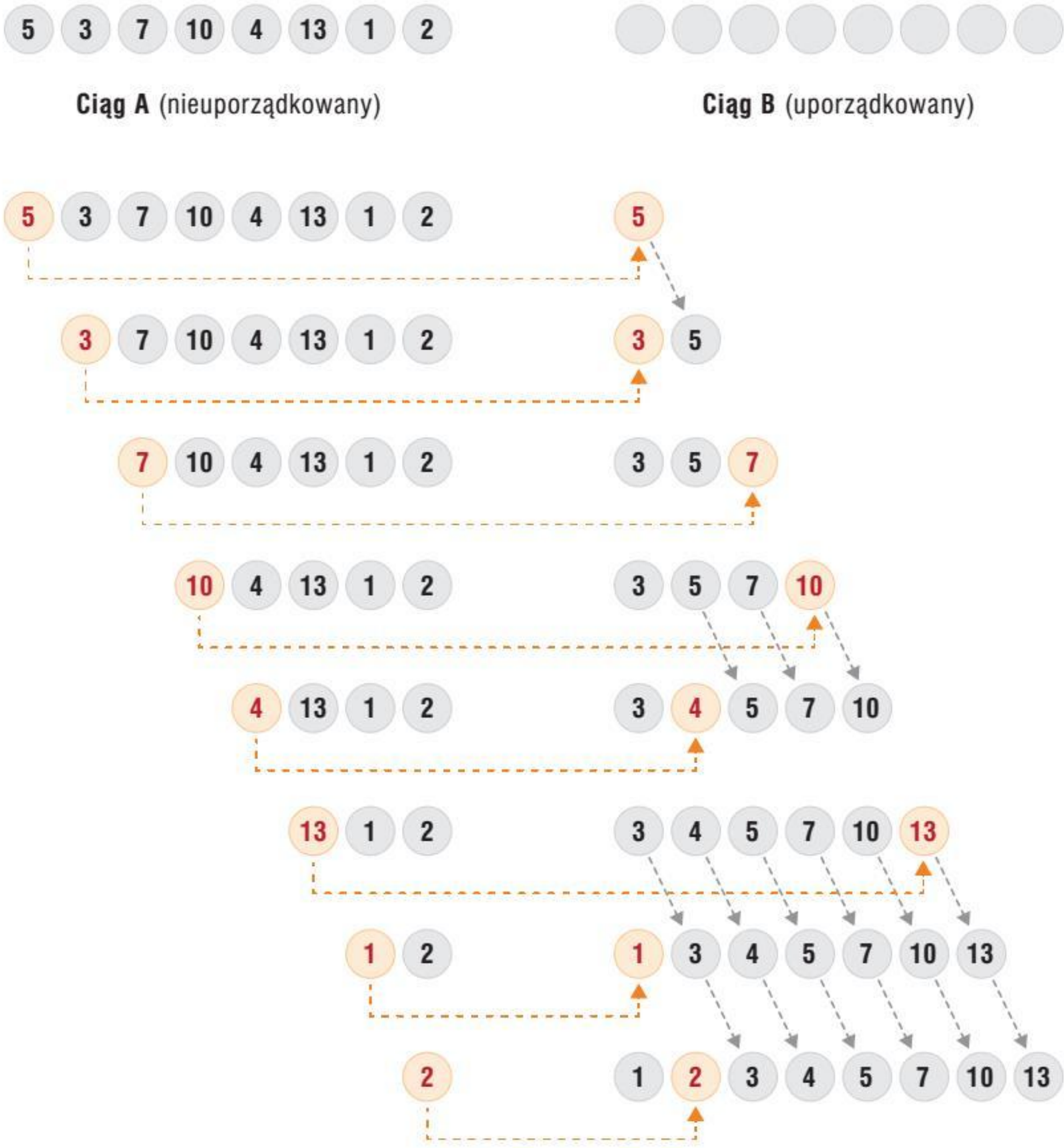
Aby znaleźć miejsce umieszczenia nowego elementu w ciągu uporządkowanym, zastosujemy **metodę liniową**, w której porównuje się wstawiany element z kolejnymi elementami ciągu uporządkowanego.

Na rysunku 2. pokazano przykład porządkowania liczb rosnąco.

Ćwiczenie 2. Analizujemy algorytm sortowania metodą przez wstawianie

1. Przeanalizuj algorytm porządkowania przez wstawianie, wykonując go dla ciągu liczb (3, 4, 10, 7, 1, 5, 2, 8). Podaj liczbę porównań wykonanych w celu uporządkowania tego ciągu.
2. Wykonaj podobne praktyczne ćwiczenie, porządkując inny ciąg liczb. Podaj liczbę porównań wykonanych w celu uporządkowania tego ciągu.

Wskazówka: Przygotuj niezbędne pomoce dydaktyczne, np. kartoniki z zapisanymi liczbami.



Rys. 2. Przykład działania algorytmu sortowania przez wstawianie do uporządkowania ciągu liczb rosnąco (od najmniejszej do największej)

Warto zapamiętać

- Sortowanie bąbelkowe ciągu *n* liczb polega na porównywaniu parami kolejnych liczb i przestawianiu ich, jeśli występują w niewłaściwej kolejności.
- Sortowanie przez wstawianie ciągu *n* liczb polega na porównywaniu wstawianego elementu z kolejnymi elementami ciągu już uporządkowanego i umieszczaniu ich w odpowiednim miejscu.

Pytania i polecenia

1. Na czym polega algorytm sortowania bąbelkowego?
2. Dlaczego w sortowaniu bąbelkowym (rosnąco) największy element już w pierwszym przebiegu przechodzi na właściwą pozycję?
3. Od czego zależy liczba wykonanych operacji zamiany miejscami elementów w algorytmie sortowania bąbelkowego?
4. Na czym polega algorytm sortowania przez wstawianie?

Zadania

1. Uporządkuj liczby pokazane na rysunku 1. od najmniejszej do największej metodą bąbelkową. Wykonaj podobny rysunek.
2. Uporządkuj liczby pokazane na rysunku 2. od największej do najmniejszej metodą przez wstawianie. Wykonaj podobny rysunek.
3. Ilu porównań wymaga wykonanie algorytmu sortowania bąbelkowego i algorytmu sortowania przez wstawianie dla następujących ciągów liczb:
   a. (4, 3, 12, 3, 20, 2, –2, 7),
   b. (2, 5, 12, 18, 90, 112, 300, 350),
   c. (67, 34, 12, 10, 6, 0, -3, -13)?

Dla zainteresowanych

4. Wieżę z dziesięciu sześciennych pudełek o różnej wielkości można ustawić tylko wtedy, gdy każde kolejne pudełko ustawimy na większym. Zaprezentuj w wybrany przez siebie sposób budowę takiej wieży (np. narysuj, wykonaj prezentację, zademonstruj z wykorzystaniem odpowiednich pomocy dydaktycznych). Zastanów się nad doborem algorytmu sortowania.

Temat 83. Metoda połowienia

1. Stosowanie metody połowienia do wyszukiwania elementu w zbiorze uporządkowanym
2. Analiza algorytmu wyszukiwania metodą połowienia

Warto powtórzyć

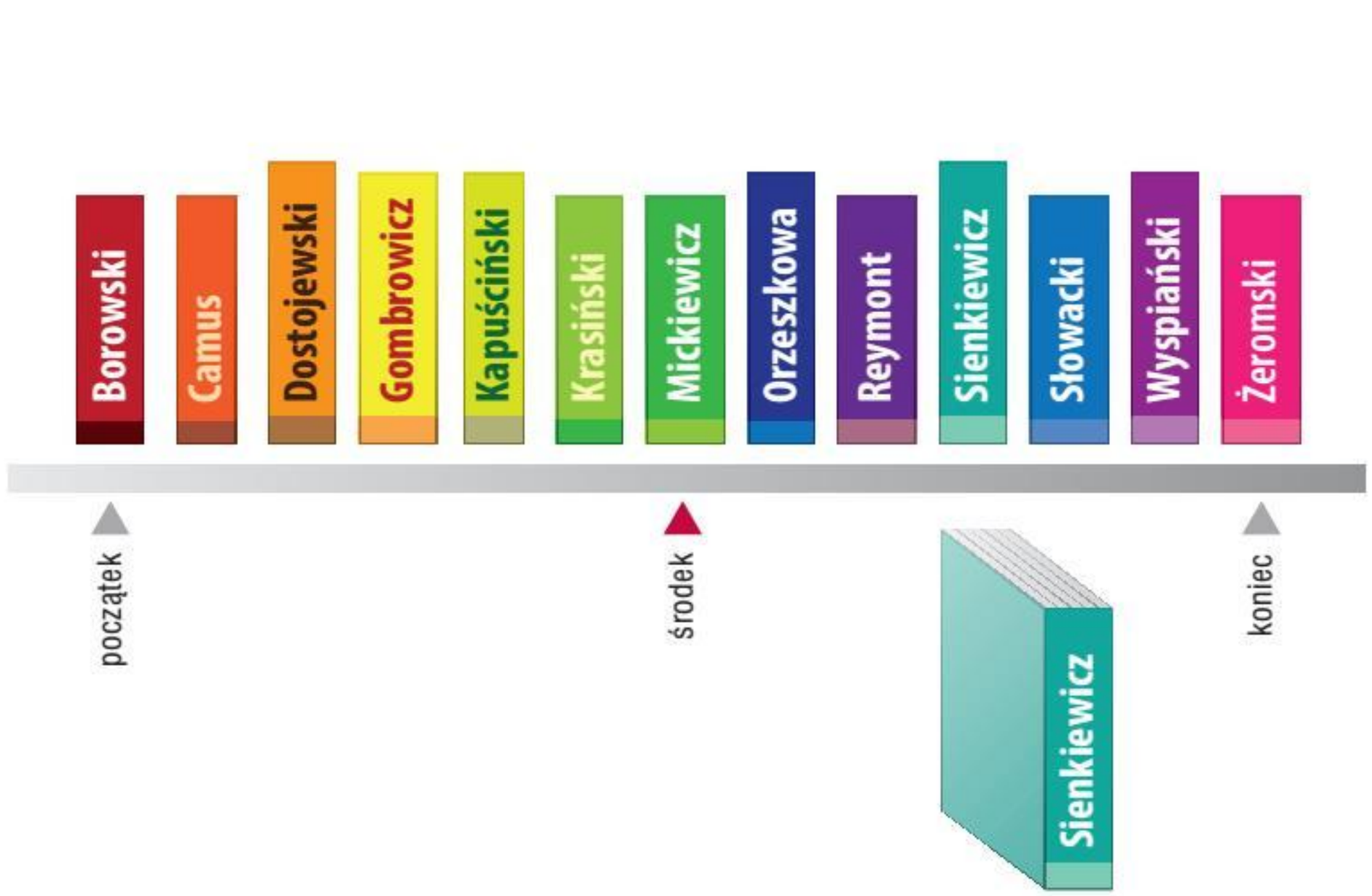
1. Jakie algorytmy nazywamy iteracyjnymi?
2. W jaki sposób wyszukujemy dany element w zbiorze nieuporządkowanym? Omów na przykładzie.
3. Na czym polega algorytm wyszukiwania największej liczby w zbiorze nieuporządkowanym?

1. Stosowanie metody połowienia do wyszukiwania elementu w zbiorze uporządkowanym

Z wyszukiwaniem przez połowienie spotykamy się w życiu codziennym, np. gdy szukamy jakiejś rzeczy w zbiorze, w którym panuje ustalony porządek.

Wyobraźmy sobie półkę z książkami, które ułożono alfabetycznie według nazwisk autorów, zaczynając od lewej strony (rys. 1.). Chcemy znaleźć książkę autora, którego nazwisko zaczyna się na wybraną literę. W jaki sposób jej szukamy?

Załóżmy, że szukamy książki autora, którego nazwisko zaczyna się na literę S. Czy przeglądamy półkę od samego początku? Raczej nie. Zwykle odruchowo sprawdzamy, czy szukanej książki nie ma gdzieś w okolicach środka. Jeśli w wybranym przez nas miejscu trafimy na inną książkę, np. autora, którego nazwisko zaczyna się na literę M, to nie przeszukujemy lewej połowy półki, tylko będziemy szukać po prawej stronie, znów dzieląc pozostałą część półki na dwie części. Szukamy dotąd, aż znajdziemy potrzebną książkę. Bezwiednie postępujemy zgodnie z algorytmem wyszukiwania przez połowienie.



Rys. 1. Przykładowa ilustracja do ćwiczenia 1.

Ćwiczenie 1. Szukamy książki na półce

1. Wspólnie z koleżankami i kolegami ułóżcie na półce trzynaście książek różnych autorów w porządku alfabetycznym według nazwiska twórcy. Następnie wybrane osoby powinny wyszukać książkę konkretnego autora.
2. Wyjaśnij sposób szukania książki autora, którego nazwisko zaczyna się na literę S, np. Henryka Sienkiewicza. Ponumeruj miejsca położenia książek na półce i podaj pozycję znalezionej książki.

Uwaga: Przyjmij, że na półce znajduje się tylko jedna książka danego autora.

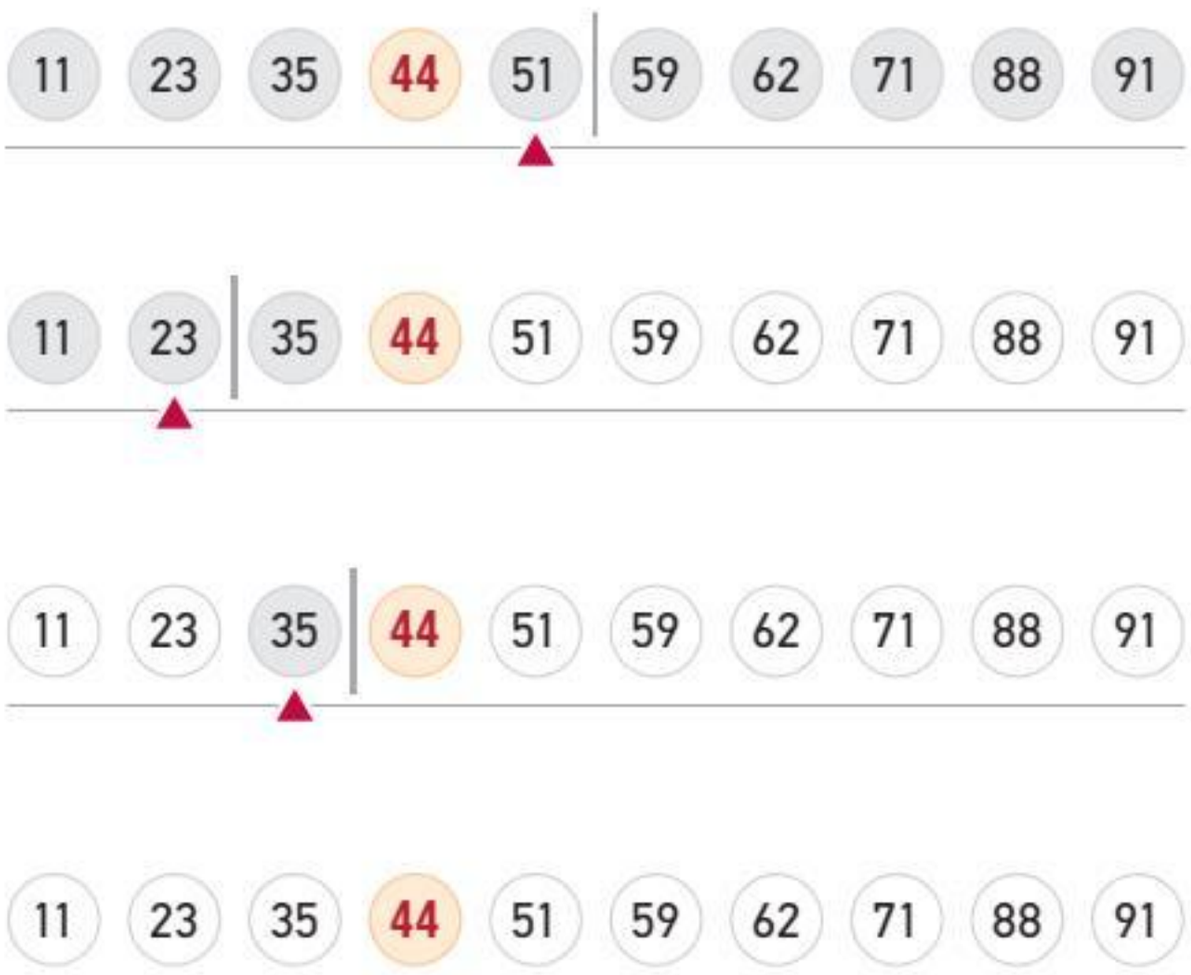
Algorytm wyszukiwania przez połowienie jest przykładem metody „dziel i zwyciężaj". Polega ona na dzieleniu przeszukiwanego zbioru na dwie części i zawężeniu przeszukiwania do jednej z tych części. Metoda podziału pomaga szybko znaleźć poszukiwany element, czyli zwyciężyć.

W algorytmie wyszukiwania elementu w zbiorze uporządkowanym szukamy danego elementu i równocześnie jego położenia w tym zbiorze.

2. Analiza algorytmu wyszukiwania metodą połowienia

Załóżmy, że mamy zbiór dziesięciu liczb uporządkowany w kolejności od najmniejszej do największej (rys. 2.). Poszukiwana liczba to 44. Jeżeli w zbiorze jest kilka takich samych wartości, algorytm znajdzie pozycję jednego z tych elementów (nie jest określone, którego).

Dzielimy zbiór liczb na dwie części, rozdzielone środkowym elementem. Zaczynamy od sprawdzenia tego elementu (liczby zaznaczonej na rys. 2. strzałką). W przypadku, gdy liczba elementów zbioru jest parzysta, to spośród dwóch środkowych elementów (w naszym przypadku liczb 51 i 59) jako „środkowy" wybieramy ten, który jest pierwszy w kolejności (czyli 51). Jeśli środkowy element jest szukaną liczbą, kończymy wyszukiwanie. Jeśli środkowy element jest większy od poszukiwanej liczby, przeszukujemy podzbiór na lewo od elementu środkowego. Jeśli środkowy element jest mniejszy od poszukiwanej liczby, przeszukujemy podzbiór na prawo od elementu środkowego.



Rys. 2. Przykład wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia (kolorem szarym zaznaczono elementy przeszukiwane w danym kroku, a pionowa kreska oznacza połowę aktualnie przeszukiwanej części)

Szukamy liczby 44.				
Lp.	**„Środkowy element"**	**Pytanie**	**Odpowiedź**	**Działanie**
1.	„Środkowym" elementem zbioru jest liczba 51.	Czy liczba 51 jest większa od 44?	TAK	Dalej przeszukujemy tylko lewą część zbioru, a odrzucamy prawą oraz element środkowy 51.
2.				
3.				
4.	–	–	–	Pozostał nam jeden element i jest to szukana liczba 44, która jest czwartą wartością w zbiorze.

Tabela 1. Przykładowa analiza rozwiązania – wzór tabeli do ćwiczenia 2.

Ćwiczenie 2. Sprawdzamy działanie algorytmu wyszukiwania liczby w zbiorze uporządkowanym metodą przez połowienie

1. Przygotuj odpowiednie pomoce dydaktyczne (rys. 2.) i przedstaw wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania liczby w zbiorze uporządkowanym metodą przez połowienie.
2. Przeprowadź analizę rozwiązania, tworząc i uzupełniając tabelę podobną do tabeli 1. Skorzystaj z rysunku 2. i podanego opisu algorytmu. Odpowiedz na pytania:
   a. Jakie powtarzające się operacje zauważasz w rozwiązaniu?
   b. Kiedy kończymy powtarzanie operacji połowienia zbioru?

Warto zapamiętać

- Algorytm wyszukiwania przez połowienie jest przykładem metody „dziel i zwyciężaj". Polega ona na dzieleniu przeszukiwanego zbioru na dwie części i zawężeniu przeszukiwania do jednej z tych części. Metoda podziału pomaga szybko znaleźć poszukiwany element, czyli zwyciężyć.
- W algorytmie wyszukiwania przez połowienie, poza szukaniem danego elementu, określamy równocześnie jego położenie w zbiorze.

Pytania i polecenia

1. Dlaczego metoda połowienia jest przykładem metody „dziel i zwyciężaj"?
2. W jaki sposób szukamy danego elementu w zbiorze uporządkowanym? Wyjaśnij na przykładzie.
3. Jaka jest różnica w określaniu elementu środkowego uporządkowanego zbioru w przypadku parzystej i nieparzystej liczby elementów?

Zadania

1. Wykonaj ćwiczenie 2. dla nieparzystej liczby elementów zbioru: {13, 29, 35, 41, 57, 62, 75, 81, 99}. Wykonaj analizę rozwiązania w tabeli podobnej do tabeli 1.
2. Podaj (inny niż opisany w tym temacie) przykład wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia. Rozwiązanie objaśnij za pomocą rysunku, prezentacji multimedialnej lub wykonanych pomocy dydaktycznych. Przeprowadź analizę rozwiązania podobną do pokazanej w tabeli 1.

Temat 84. Metoda zachłanna

1. **Jak znaleźć najlepszy sposób osiągnięcia celu?**
2. **Stosowanie metody zachłannej do wydawania reszty.**
   2.1. Opis algorytmu wydawania reszty metodą zachłanną
   2.2. Analiza algorytmu wydawania reszty metodą zachłanną

Warto powtórzyć

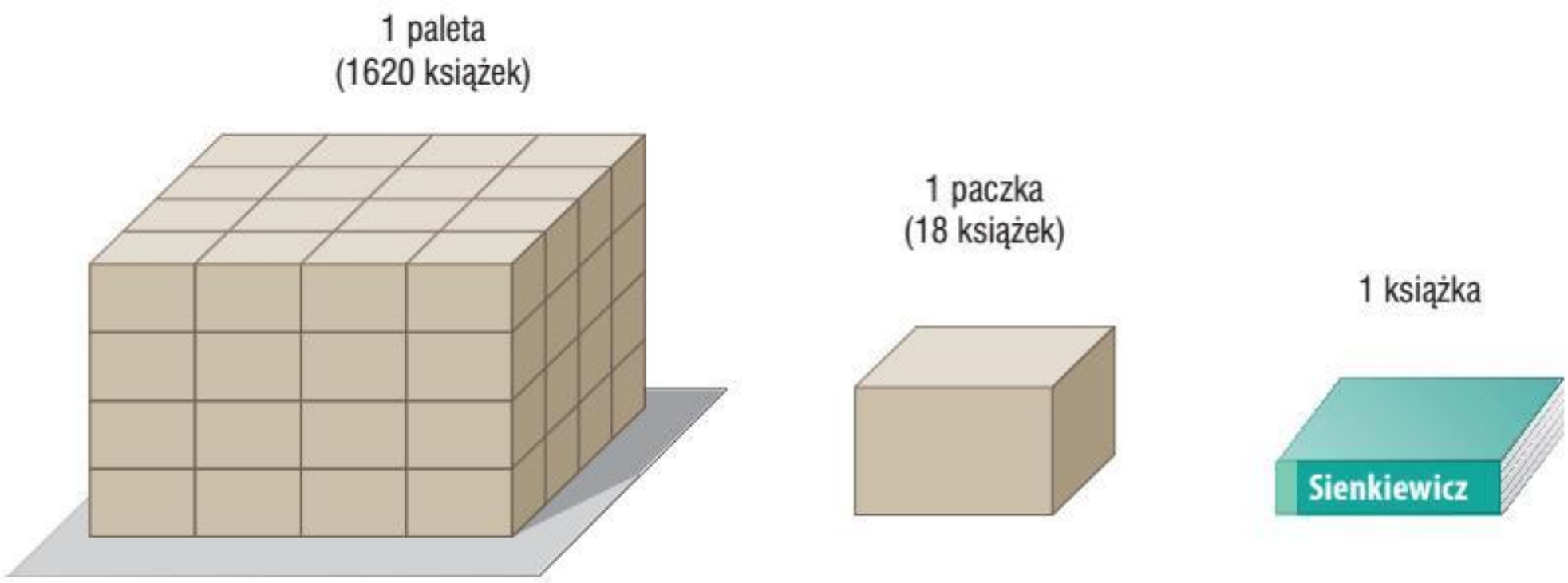
1. Na czym polega metoda połowienia?
2. W jaki sposób wykonujemy dzielenie z resztą? Pokaż na przykładzie.
3. Jaką technikę stosujemy, gdy w algorytmie pewne czynności się powtarzają?

1. Jak znaleźć najlepszy sposób osiągnięcia celu?

Jeśli można usprawnić i przyspieszyć jakieś działanie, to zwykle staramy się to zrobić. Próbujemy wymyślić sposób jak najszybszego osiągnięcia wyznaczonego celu. Zacznijmy od następującego problemu.

Jesteś kierowcą samochodu dostawczego i musisz przewieźć pewną liczbę książek z drukarni do magazynu. Książki zapakowano w paczki, które umieszczono na paletach. W samochodzie możesz umieszczać pełne palety lub paczki bądź pojedyncze egzemplarze. Jak sprawnie i szybko załadować książki do samochodu?

Czy załadunek lepiej zacząć od pojedynczych książek? Czy może od paczek z książkami? Przede wszystkim chcemy jak najmniej razy umieszczać książki w samochodzie, a więc w pierwszej kolejności powinniśmy korzystać z opakowań mieszczących jak najwięcej książek, czyli palet (oczywiście paletę umieszcza się w samochodzie, wykorzystując specjalny podnośnik). Jeśli nie uda się umieścić w samochodzie potrzebnej liczby książek wyłącznie w paletach, trzeba będzie dodać te w mniejszych opakowaniach, czyli paczki, a następnie pojedyncze książki, aż umieścimy w samochodzie potrzebną liczbę książek.



Rys. 1. Przykładowa ilustracja – ćwiczenie 1.

Ćwiczenie 1. Szukamy rozwiązania problemu umieszczenia książek w samochodzie

1. Musisz przewieźć 3500 książek z drukarni do magazynu. Książki zapakowano w paczki, które umieszczono na paletach. W jednej paczce mieści się 18 książek, natomiast na palecie – 1620 książek.
2. Oblicz, ile pełnych palet, ile paczek, a ile pojedynczych książek znajdzie się w samochodzie? Pokaż sposób wykonywania obliczeń.
3. Jakie zauważasz powtarzające się operacje? Wyróżnij problem cząstkowy.

Wskazówka: Podczas wykonywania obliczeń zastosuj dzielenie z resztą.

Staraliśmy się znaleźć najkorzystniejsze rozwiązanie problemu, postępując w sposób, który ma cechy zachłanności.

Algorytm zachłanny – algorytm, który w celu wyznaczenia rozwiązania w każdym kroku dokonuje zachłannego, tj. najkorzystniejszego w danym kroku, rozwiązania częściowego.

2. Stosowanie metody zachłannej do wydawania reszty

2.1. Opis algorytmu wydawania reszty metodą zachłanną

Z problemem wydawania reszty spotkamy się wielokrotnie. Mimo że dziś często używamy kart płatniczych, to umiejętność obliczenia reszty przydaje się, nawet jeśli kasa podpowiada sprzedawcy, ile reszty ma wydać.

Jesteś sprzedawcą w sklepie. Klient zapłacił gotówką. Czy możesz algorytm zachłanny wykorzystać podczas wydawania reszty klientowi?

Ćwiczenie 2. Wydajemy resztę klientowi

Artykuł kosztował 457 zł. Klient dał trzy banknoty po 200 zł. Wyjaśnij, jak wydasz resztę klientowi. Czy stosujesz algorytm zachłanny? Uzasadnij odpowiedź.

Resztę wydajemy, używając jak najmniejszej liczby banknotów lub bilonu, czyli możemy w tym przypadku zastosować ten sam algorytm, co podczas ładowania książek do samochodu. Zauważmy, że ten sam algorytm możemy wykorzystać w różnych sytuacjach.

Metoda zachłanna zakłada użycie do wydania reszty największych dostępnych nominałów (mniejszych od lub równych wydawanej wartości). Zanim zaczniemy wydawać resztę, musimy określić dostępne nominały (zbiór nominałów).

2.2. Analiza algorytmu wydawania reszty metodą zachłanną

Algorytm zaczynamy od sprawdzenia, czy w zbiorze nominałów znajduje się wartość N mniejsza od reszty R lub jej równa.

Jeśli odnaleziony nominał N jest równy R, to algorytm się kończy. Jeśli nie, to wybieramy największą możliwą liczbę L sztuk znalezionego nominału N mniejszego od R (zakładamy, że zbiór nominałów jest posortowany malejąco). Liczbę tę otrzymujemy, dzieląc R przez wartość odnalezionego nominału (liczba nominałów to oczywiście część całkowita wyniku dzielenia).

Następnie od reszty R odejmujemy wartość znalezionego nominału pomnożoną przez liczbę nominałów wynikającą z dzielenia.

Szukanie powtarzamy wśród kolejnych nominałów dla pomniejszonej wartości R do czasu, aż wypłacimy całą sumę. Mamy tu więc odniesienie do iteracji.

Korzystając z podanego opisu, znajdziemy nominały, którymi zostanie wydana reszta w wysokości 69 zł.

Reszta $R = 69$. Zbiór dostępnych nominałów to: {50, 20, 10, 5, 2, 1}.					
1.	Największy nominał, nie większy niż R to 50 zł.	Część całkowita z dzielenia 69 przez 50 to: 1.	Wypłacamy jeden banknot o nominale 50 zł.	$R = 69 - 50 \cdot 1 = 19$	Zostało 19 zł reszty do wydania.
2.	Największy nominał, nie większy niż aktualne R to 10 zł.	Część całkowita z dzielenia 19 przez 10 to: 1.	Wypłacamy jeden banknot o nominale 10 zł.	$R = 19 - 10 \cdot 1 = 9$	Zostało 9 zł reszty do wydania.
3.					
4.					

Tabela 1. Przykładowa analiza rozwiązania – wzór tabeli do ćwiczenia 3.

Ćwiczenie 3. Sprawdzamy działanie algorytmu wydawania reszty metodą zachłanną

1. Przygotuj odpowiednie pomoce dydaktyczne i przedstaw wspólnie z koleżankami i kolegami z klasy algorytm wydawania reszty metodą zachłanną.
2. Przeprowadź analizę rozwiązania, tworząc i uzupełniając tabelę podobną do tabeli 1. Skorzystaj z podanego opisu algorytmu. Odpowiedz na pytania:
   a. Jakie operacje powtarzają się w rozwiązaniu?
   b. Jak ostatecznie wydano resztę?

Warto zapamiętać

- Algorytm zachłanny to algorytm, który w celu wyznaczenia rozwiązania w każdym kroku dokonuje zachłannego, tj. najkorzystniejszego w danym kroku rozwiązania częściowego.
- Algorytm zachłanny można zastosować m.in. do wydawania reszty.

Pytania i polecenia

1. Na czym polega algorytm zachłanny?
2. Czy do zaplanowania sposobu spakowania książek w ćwiczeniu 1. stosowaliśmy algorytm zachłanny? Uzasadnij odpowiedź.
3. Wyjaśnij, dlaczego metoda zachłanna jest przydatna do wydawania reszty?

Zadania

1. Rozwiąż problem podobny do opisanego w ćwiczeniu 1., tylko dla innych danych. Omów sposób wykonywania obliczeń.
2. Musisz przewieźć 50 palet ładunku (którego nie możesz rozdzielać). Dysponujesz pojazdami o ładowności 20 palet, 8 palet i 1 paleta. Ile i jakich samochodów użyjesz, aby jak najlepiej je wykorzystać? Czy można zastosować sposób z ćwiczenia 1.? Uzasadnij odpowiedź.
3. Omów metodę zachłanną wydawania reszty wynoszącej 178 zł. Dostępne nominały to 100 zł, 50 zł, 20 zł, 10 zł, 5 zł, 2 zł, 1 zł.

Dla zainteresowanych

4. Sformułuj samodzielnie problem, do którego rozwiązania można zastosować podejście zachłanne. Rozwiązanie objaśnij za pomocą rysunku, prezentacji multimedialnej lub wykonanych pomocy dydaktycznych. Przeprowadź analizę rozwiązania podobną do pokazanej w tabeli 1.

Temat 85. Rekurencja a iteracja

1. Na czym polega rekurencja?
2. Algorytm obliczania silni liczby naturalnej
3. Obliczanie wartości elementów ciągu Fibonacciego

Warto powtórzyć

1. Na czym polega iteracja?
2. Do jakich czynności ma zastosowanie technika iteracji? Wyjaśnij na przykładzie.

1. Na czym polega rekurencja?

Podczas rozwiązywania problemu możemy skorzystać z rozwiązania, które wypracowaliśmy wcześniej przy podobnym problemie. Możemy też podzielić problem na mniejsze podproblemy, które będzie nam łatwiej rozwiązać, a później przejść do rozwiązania całego problemu.

Możemy również w problemie i jego algorytmie (sposobie rozwiązania) wyodrębnić części podobne do całości i zmienić algorytm tak, aby korzystał z siebie samego.

Na przykład, w definicji zbioru liczb naturalnych następuje odwołanie do tej definicji.

a) 0 *jest liczbą naturalną*,

b) *każda liczba naturalna ma swój następnik*,

c) 0 *nie jest następnikiem żadnej liczby naturalnej.*

Z **rekurencją** mamy do czynienia, gdy, określając jakieś pojęcie, odwołujemy się w definicji do niego samego.

Rekurencję stosujemy często w definicjach matematycznych.

Zjawiska rekurencyjne występują również w naszym otoczeniu. Na rysunku 1. rekurencja jest przedstawiona w postaci „zagłębiających się w siebie", podobnych, coraz to mniejszych figur geometrycznych.

Podobny efekt możemy uzyskać dzięki odbiciom w lustrze. Wystarczy, patrząc w jedno lustro, nakierować na nie drugie, najlepiej mniejsze, owalne. Uzyskamy wówczas ciąg odbić mieszczących się jedno w drugim.



Rys. 1. Wizualny efekt zjawiska rekurencji

Ćwiczenie 1. Podajemy przykłady zjawisk rekurencyjnych

Podaj przykłady zjawisk rekurencyjnych, które można spotkać w życiu codziennym lub które znasz z przedmiotów szkolnych.

2. Algorytm obliczania silni liczby naturalnej

Aby zrozumieć, na czym polega rekurencja, porównamy realizację iteracyjną i rekurencyjną tego samego algorytmu. Porównanie przeprowadzimy na przykładzie algorytmu obliczania **silni** liczby naturalnej.

Iteracyjna definicja silni liczby naturalnej n:

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n & \text{dla } n > 0 \end{cases}$$

Silnia S

Silnia liczby naturalnej n to iloczyn wszystkich liczb naturalnych dodatnich nie większych niż n.

Obliczenia silni wynikające z definicji iteracyjnej dla $n = 3$:
$3! = 1 \cdot 2 \cdot 3 = 6$.
Warto zauważyć, że wartości silni bardzo szybko rosną, bo już:
$17! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot 16 \cdot 17 = 355687428096000$.

Ćwiczenie 2. Obliczamy iteracyjnie silnie liczb naturalnych

1. Korzystając z iteracyjnej definicji silni, oblicz: 2!, 5!, 10!.
2. Zastanów się, jakie operacje powtarzają się w algorytmie iteracyjnym obliczania silni.

Rekurencyjna definicja silni liczby naturalnej n:

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ (n-1)! \cdot n & \text{dla } n > 0 \end{cases}$$

Zgodnie z zasadą rekurencji, w rekurencyjnej definicji silni liczby n pojawia się definicja tego samego pojęcia, czyli silni, tyle że liczby $n - 1$. W definicji rekurencyjnej musi być umieszczony przypadek nierekurencyjny (tu: 1 dla $n = 0$). W przeciwnym przypadku obliczenia się nigdy nie skończą.

Obliczenia silni wynikające z definicji rekurencyjnej dla $n = 3$:
$3! = 2! \cdot 3 = (1! \cdot 2) \cdot 3 = ((0! \cdot 1) \cdot 2) \cdot 3 = ((1 \cdot 1) \cdot 2) \cdot 3 = 6$.

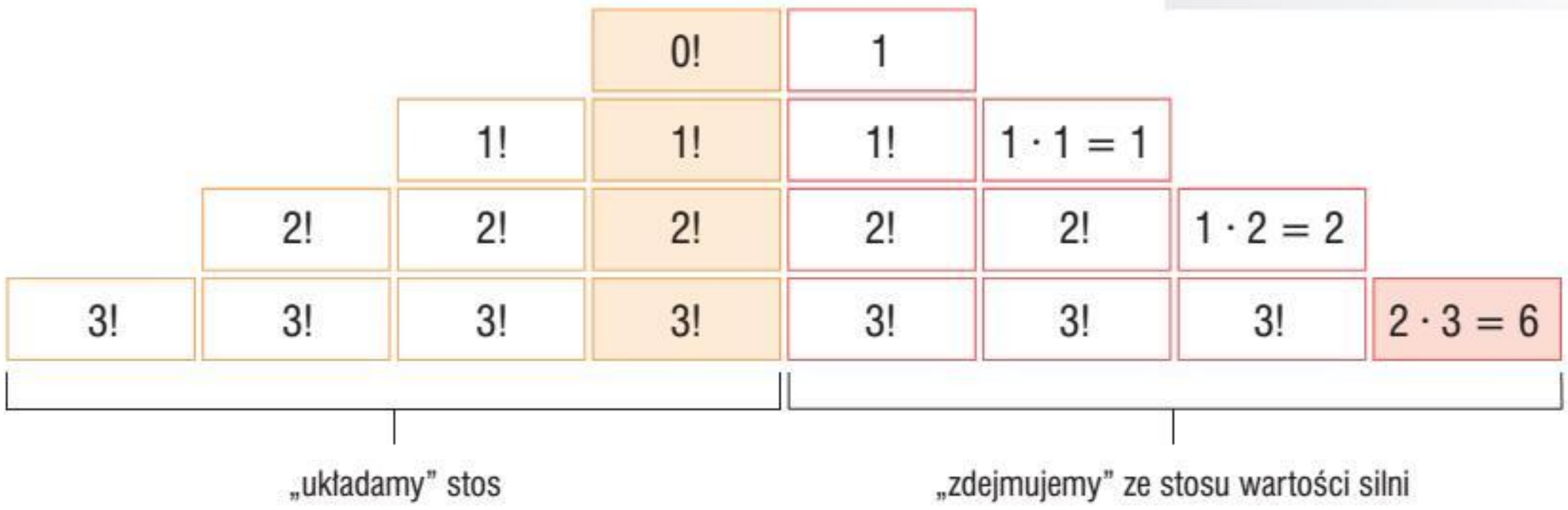
Na rysunku 2. pokazano schemat rekurencyjnych obliczeń 3!:

- aby obliczyć 3!, musimy znać wartość 2!;
- aby obliczyć 2!, musimy znać 1!;
- aby obliczyć 1!, musimy wiedzieć, ile wynosi 0!.

W ten sposób powstaje tzw. **stos**, na którym dostępny jest tylko element znajdujący się na wierzchołku, czyli 0!. „Zdejmując" z góry wartości silni ze stosu, dojdziemy do wartości 3!, czyli 6.

S **Stos w informatyce**

Struktura przechowywania danych, w której dostępny jest tylko element znajdujący się na wierzchołku stosu (tak jak w stosie książek na biurku – dostępna jest książka położona na wierzchu jako ostatnia).



Rys. 2. Schemat rekurencyjnego obliczenia silni dla $n = 3$

Ćwiczenie 3. Obliczamy rekurencyjnie silnie liczb naturalnych

Korzystając z rekurencyjnej definicji silni, oblicz: 2!, 5!, 10!.

Znajomość definicji rekurencyjnej obliczania silni przydaje się do wykonywania niektórych obliczeń. Na przykład, jeśli wiemy, że $9! = 8! \cdot 9$, to łatwo skrócić wspólne czynniki w liczniku i mianowniku wyrażenia:

$$\frac{9!}{8! \cdot 3} = \frac{8! \cdot \overset{3}{\cancel{9}}}{\underset{1}{\cancel{8!}} \cdot \cancel{3}} = 3$$



Rekurencja i iteracja różnią się zasadniczo. Powtórzenia w rekurencji są innego rodzaju niż powtórzenia właściwe dla iteracji. Powtórzenia w rekurencji „zagłębiają się w siebie" na zasadzie powrotu do tej samej definicji.

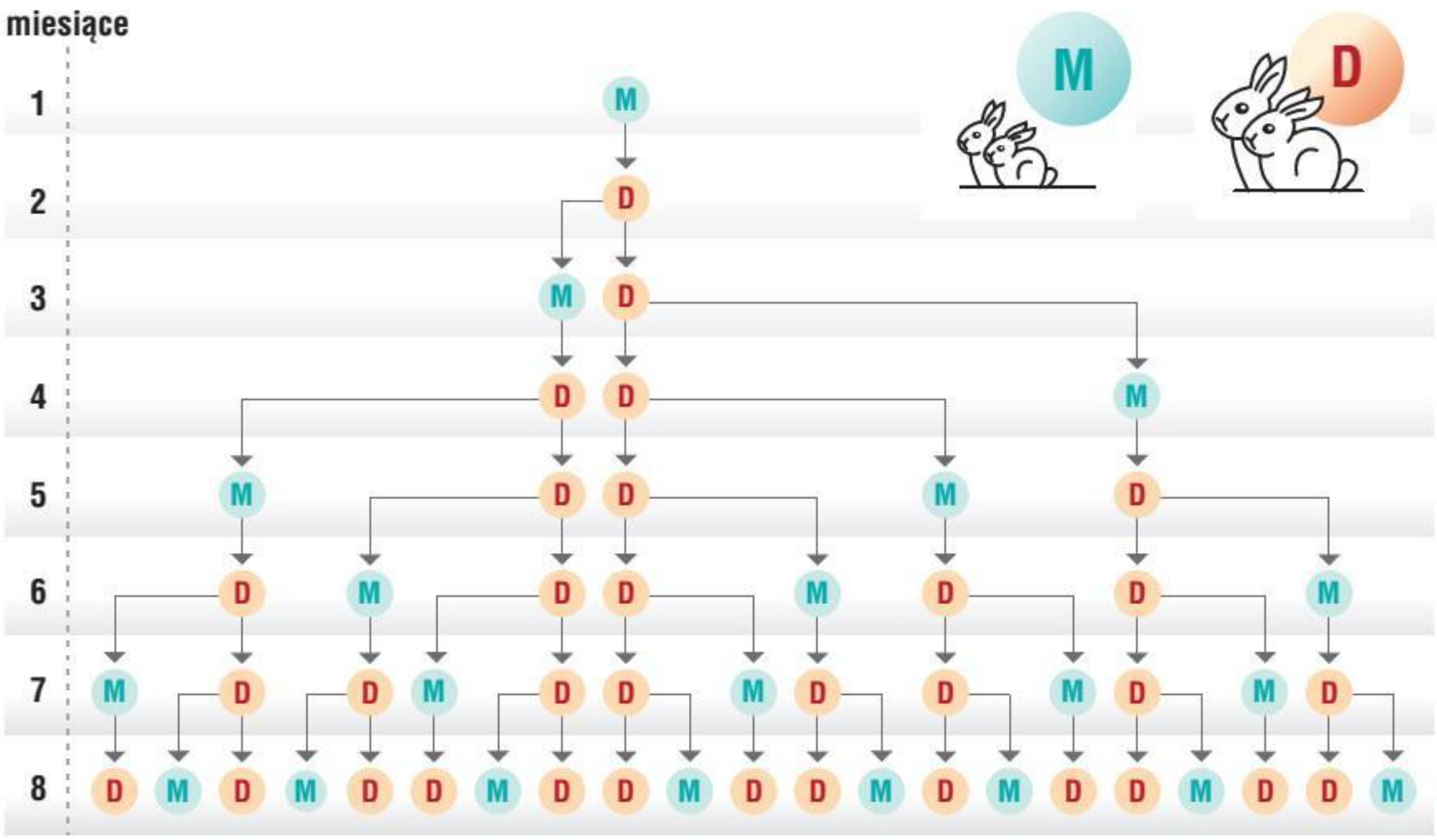
3. Obliczanie wartości elementów ciągu Fibonacciego

W roku 1202 wybitny włoski matematyk okresu średniowiecza, Leonardo Pisano, zwany Fibonaccim, przedstawił problem, który do tej pory dostarcza natchnienia rzeszom matematyków. Dotyczy on liczebności populacji królików.

Opis problemu: Początkowo rodzi się para królików, która po miesiącu osiąga dojrzałość. Po następnym miesiącu wydaje na świat parę królików, po miesiącu następną parę i tak dalej, tzn. pierwsza para co miesiąc wydaje na świat potomstwo (parę królików). Z kolei każda nowa para po miesiącu osiąga dojrzałość i po każdym następnym miesiącu również wydaje na świat kolejną parę królików. Proces ten nie ma końca. Zakładamy, że króliki żyją wiecznie.

Pytanie brzmi: jaka jest liczba par królików po n miesiącach?

Opis algorytmu: Zauważmy, że liczba par królików w danym miesiącu (z wyjątkiem pierwszego i drugiego miesiąca) zależy od dwóch wartości: liczby par królików dorosłych i liczby par królików młodych w poprzednim miesiącu. Tych ostatnich jest z kolei tyle, ile było par królików dorosłych dwa miesiące wcześniej.



Rys. 3. Schemat przedstawia liczebność populacji królików w pierwszych ośmiu miesiącach doświadczenia (M – para królików młodych, D – para królików dojrzałych); ósma liczba Fibonacciego to 21 par



Jeśli oznaczymy całkowitą liczbę par królików w n-tym miesiącu jako F_n, to będziemy ją mogli określić następująco:

$$F_n = \begin{cases} 1 & \text{dla } n \in \{1, 2\} \\ F_{n-1} + F_{n-2} & \text{dla } n \geq 3. \end{cases}$$

Liczba F_n opisuje całkowitą wielkość populacji królików po upływie n miesięcy i nazywana jest **liczbą Fibonacciego**. Natomiast ciąg liczb naturalnych, w którym każda liczba (z wyjątkiem pierwszej i drugiej) jest sumą dwóch poprzednich, nazywamy **ciągiem Fibonacciego**.

Podana definicja ciągu Fibonacciego jest **rekurencyjna**, gdyż odwołuje się do siebie samej. Oto kilka początkowych liczb tego ciągu:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, … .

Obliczenia czwartej liczby Fibonacciego wynikające z definicji rekurencyjnej, czyli dla $n = 4$:

$$F_4 = F_3 + F_2 = (F_2 + F_1) + 1 = (1 + 1) + 1 = 2 + 1 = 3.$$

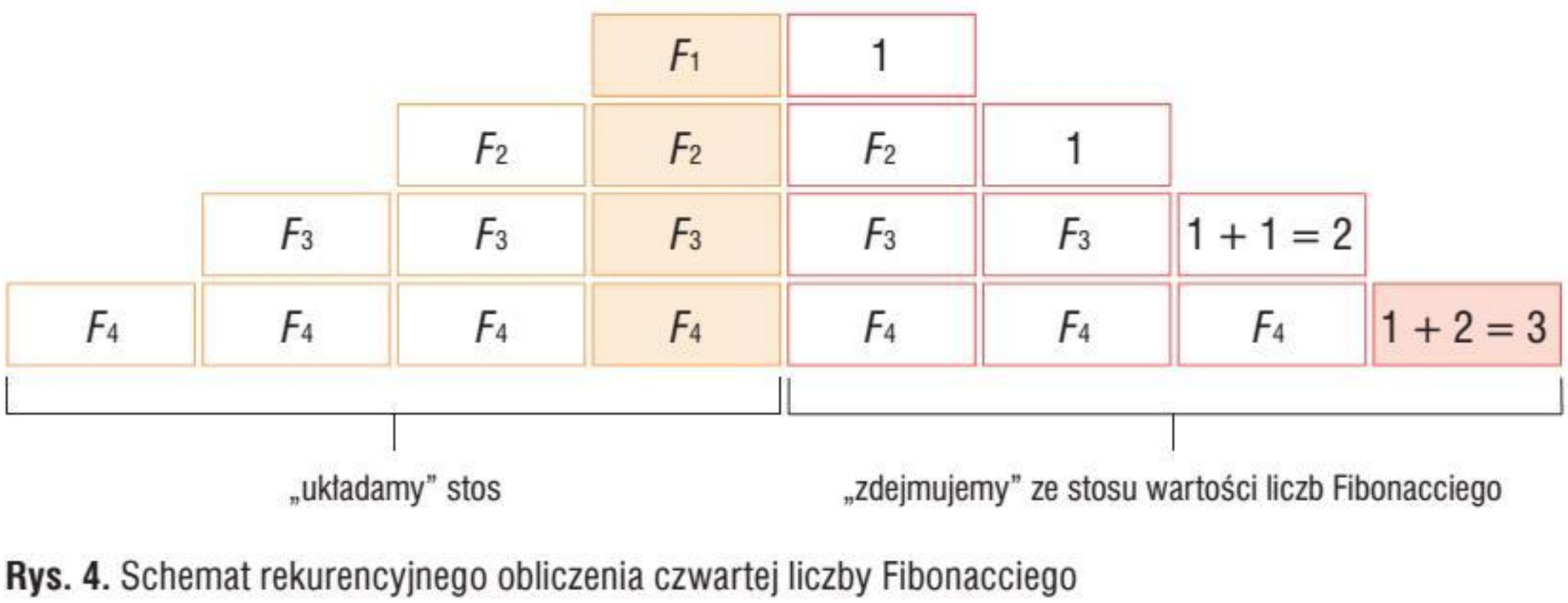
Na rysunku 4. pokazano schemat rekurencyjnych obliczeń czwartej liczby Fibonacciego (F_4):

- aby obliczyć F_4, musimy znać wartość F_3 i F_2;
- aby obliczyć F_3, musimy znać wartość F_2 i F_1.

W ten sposób powstaje tzw. **stos**, na którym dostępny jest tylko element znajdujący się na wierzchołku, czyli wartość pierwszej liczby Fibonacciego = 1. „Zdejmując" z góry wartości liczb Fibonacciego ze stosu, dojdziemy do wartości czwartej liczby, czyli 3.

Ćwiczenie 4. Obliczamy wybrane liczby Fibonacciego

Oblicz trzecią i szóstą liczbę ciągu Fibonacciego. Czy możesz obliczyć szóstą liczbę bez wcześniejszego obliczenia czwartej i piątej? Uzasadnij odpowiedź.



Rys. 4. Schemat rekurencyjnego obliczenia czwartej liczby Fibonacciego

Aby obliczyć poszczególne liczby Fibonacciego, można posłużyć się zarówno algorytmem iteracyjnym, jak i rekurencyjnym. Jednak w tym przypadku algorytm w postaci rekurencyjnej wymaga wykonania dużo większej liczby kroków obliczeniowych niż ten sam algorytm w postaci iteracyjnej.

W postaci iteracyjnej obliczenia zaczynamy „od początku”, przyjmując, że wartość dwóch pierwszych liczb Fibonacciego wynosi 1. Następnie obliczamy wartość kolejnych (trzeciej, czwartej itd.) liczb Fibonacciego. Na przykład, aby poznać szóstą liczbę, obliczamy trzecią, czwartą i piątą liczbę:

$F_1 = 1$
$F_2 = 1$
$F_3 = 1 + 1 = \mathbf{2}$
$F_4 = \mathbf{2} + 1 = \mathbf{3}$
$F_5 = \mathbf{3} + \mathbf{2} = \mathbf{5}$
$F_6 = \mathbf{5} + \mathbf{3} = \mathbf{8}$

W algorytmie iteracyjnym musimy pamiętać (w zmiennych) wartości dwóch poprzednich liczb Fibonacciego.

Ćwiczenie 5. Objaśniamy iteracyjną realizację algorytmu obliczania liczb Fibonacciego

Zastanów się, które operacje powtarzają się podczas obliczania *n*-tej liczby Fibonacciego w algorytmie iteracyjnym. Objaśnij na przykładzie obliczania wartości dziesiątej liczby Fibonacciego.

Warto zapamiętać

- Z rekurencją spotykamy się w definicjach, w których definiowane pojęcie odwołuje się do samego siebie. W algorytmach rekurencyjnych (podobnie jak w iteracyjnych) musi być wprowadzony warunek zakończenia wywołań rekurencyjnych.
- Ciągiem Fibonacciego nazywamy ciąg liczb naturalnych, w którym każda liczba (z wyjątkiem pierwszej i drugiej) jest sumą dwóch poprzednich.
- Algorytmy obliczania silni i wyznaczania liczb Fibonacciego można realizować rekurencyjnie i iteracyjnie.
- W technice iteracji i rekurencji mamy do czynienia z powtórzeniami, jednak są to powtórzenia innego rodzaju. W przypadku iteracji stosujemy pętle, natomiast w rekurencji – nie (powtórzenia „zagłębiają się w siebie” na zasadzie powrotu do tej samej definicji).

Pytania i polecenia

1. Kiedy mamy do czynienia z rekurencją?
2. Jakie znasz przykłady zjawisk rekurencyjnych?
3. Czym jest silnia?
4. Czym różni się definicja iteracyjna od definicji rekurencyjnej obliczania silni liczby naturalnej *n*?

5. Objaśnij problem rekurencyjnego obliczania liczebności populacji królików na podstawie opisu z punktu 3. i rysunku 2.

6. W jaki sposób obliczamy k-tą liczbę Fibonacciego rekurencyjnie i iteracyjnie? Objaśnij na przykładach.

Zadania

1. Oblicz, na ile sposobów można wybrać k-osobową reprezentację spośród n uczniów jednej klasy. Przyjmij $n = 12$, $k = 3$. Skorzystaj z symbolu Newtona:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

2. Przedstaw obliczenia $5!$ podobnie jak pokazano na rysunku 2. obliczenia $3!$

3. Przedstaw obliczenia szóstej liczby Fibonacciego podobnie jak pokazano na rysunku 4. obliczenia czwartej liczby.

Dla zainteresowanych

4. Znajdź w Internecie lub innych źródłach więcej informacji na temat liczb Fibonacciego. Przygotuj opracowanie w wybranej przez siebie formie.

Przeczytaj, jeśli chcesz wiedzieć więcej…

Przydomek **Fibonacci** oznacza: Filius (z łac. „syn") Bonacciego. Fibonacci nazywał się w rzeczywistości Leonardo Pisano (Leonardo z Pizzy). Żył w latach ok. 1170-1240. Studiował księgi Muhammada Al-Chuwarizmiego i w 1202 roku opublikował dzieło „Liber Abaci", dzięki któremu Europejczycy poznali cyfry arabskie i dziesiętny system liczenia. Właśnie w tym dziele Fibonacci zawarł rozważania na temat liczebności populacji królików.

n	F_n	F_{n+1}/F_n
1	1	1,000000
2	1	2,000000
3	2	1,500000
4	3	1,666667
5	5	1,600000
6	8	1,625000
7	13	1,615385
8	21	1,619048
9	34	1,617647
10	55	1,618182
11	89	1,617978
12	144	1,618056
13	233	1,618026
14	377	1,618037
15	610	1,618033

Tabela 1. Ilorazy kolejnych liczb Fibonacciego

Ciąg Fibonacciego, z pozoru prosty, ma wiele interesujących własności. Istnieje nawet towarzystwo Fibonacci Association, zajmujące się publikowaniem wyników licznych prac badawczych dotyczących tego zagadnienia.

Ciekawą własność mają ilorazy kolejnych wyrazów ciągu Fibonacciego. W tabeli 1. przedstawiono je z dokładnością do sześciu miejsc po przecinku. Łatwo zaobserwować, jak zbliżają się do liczby

$$1{,}61803398874989484820\ldots = \frac{1+\sqrt{5}}{2}$$

zwanej **złotą średnią** lub **złotym podziałem** (z łac. *divina proportio* – „boski podział").

Liczba ta jest proporcją najczęściej spotykaną w przyrodzie i stosowaną już przez starożytnych mistrzów. W ciele ludzkim – i to zarówno w całej jego postaci, jak i w wielu jego częściach – występują złote proporcje. Złoty podział można także odnaleźć chociażby w układzie listków na gałęzi.

Temat 86. Wybrane algorytmy na liczbach

1. **Algorytm badania pierwszości liczby**
2. **Algorytm Euklidesa wyznaczania NWD**
   - 2.1. Algorytm Euklidesa w wersji z odejmowaniem
   - 2.2. Algorytm Euklidesa w wersji z dzieleniem

Warto powtórzyć

1. Podaj przykłady liczb pierwszych.
2. Kiedy liczba x jest podzielna przez liczbę y?
3. Na czym polega dzielenie z resztą?

1. Algorytm badania pierwszości liczby

Algorytm badania pierwszości liczby polega na sprawdzaniu wszystkich jej dzielników większych od 1 i mniejszych od niej samej. Jeżeli liczba nie posiada takich dzielników, jest **liczbą pierwszą**.

Liczba pierwsza L

Liczba naturalna większa od 1, która ma dokładnie dwa dzielniki naturalne: jedynkę i siebie samą.
Na przykład:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29.
Liczba 1 nie jest liczbą pierwszą.

Aby określić, czy dana liczba n jest pierwsza, należy sprawdzić jej dzielniki. Nie musimy jednak sprawdzać wszystkich dzielników od 2 do n – wystarczy sprawdzić, czy dana liczba ma dzielniki w zbiorze $\langle 2; \sqrt{n} \rangle$. Jeśli ma, nie jest liczbą pierwszą. Sprawdzanie polega na dzieleniu badanej liczby n przez kolejne liczby naturalne należące do przedziału $\langle 2; \sqrt{n} \rangle$. Jeśli reszta z dzielenia będzie równa zero, oznacza to, że dana liczba n nie jest pierwsza.

Na przykład:

- liczba 9 w zbiorze $\langle 2; \sqrt{9} \rangle$ ma jeden dzielnik: 3 – czyli nie jest liczbą pierwszą,
- liczba 17 w zbiorze $\langle 2; \sqrt{17} \rangle$ nie ma dzielników – czyli jest liczbą pierwszą.

Ćwiczenie 1. Sprawdzamy, czy liczba jest liczbą pierwszą

Sprawdź, która z podanych liczb: 18, 31, 42, 89 jest liczbą pierwszą.

2. Algorytm Euklidesa wyznaczania NWD

Jesteś stolarzem i musisz z dwóch desek otrzymać deseczki o takiej samej, ale jak największej, długości. Jak obliczyć długość tej deseczki, aby podczas cięcia każdej z dwóch desek wykorzystać każdą z nich w całości?

Euklides (matematyk ze starożytnej Grecji) w swoim dziele „Elementy" przedstawił sposób znajdowania największego wspólnego dzielnika dwóch liczb naturalnych, wychodząc od badań nad długościami odcinków. Szukał wspólnej miary dwóch odcinków, czyli najdłuższego odcinka, który można odłożyć całkowitą liczbę razy w obydwu odcinkach bez pozostawiania ścinków.

Euklides oparł swój algorytm na spostrzeżeniu, że wspólna miara dwóch odcinków odkłada się również całkowitą liczbę razy w różnicy długości tych odcinków. Odejmował od długości dłuższego odcinka długość krótszego, aż do otrzymania odcinków o równej długości. Zatem wspólną miarą odcinków okazała się długość dowolnego z końcowych odcinków.

N **NWD**
(ang. GCD – *Greatest Common Divisor*)

Największy wspólny dzielnik dwóch lub więcej liczb naturalnych (różnych od zera) to największa liczba naturalna, przez którą dzieli się bez reszty każda z tych liczb.

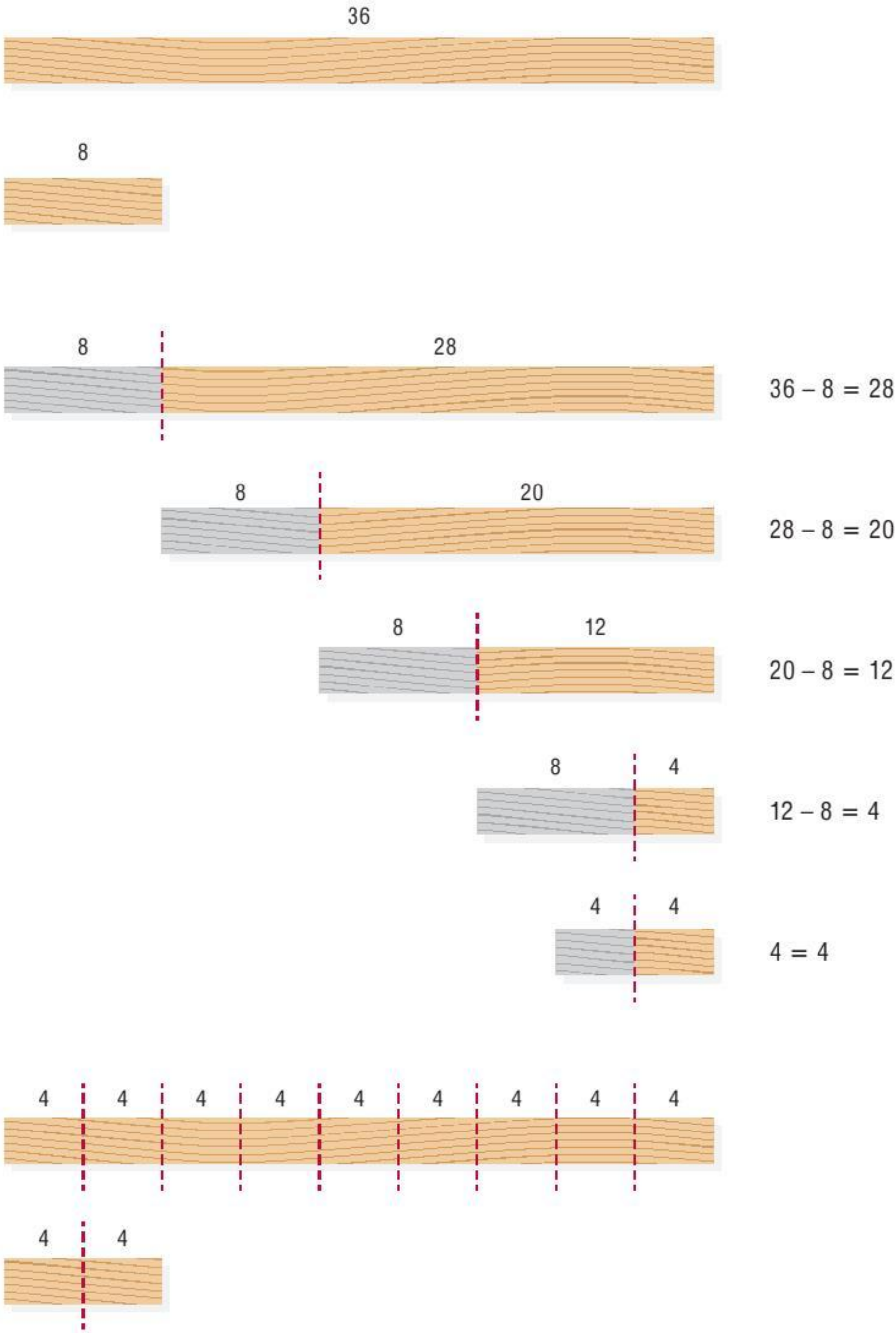
Algorytm Euklidesa służy do znajdowania największego wspólnego dzielnika (**NWD**) dwóch liczb naturalnych.

Omówimy działanie algorytmu Euklidesa w dwóch wersjach: z odejmowaniem i dzieleniem.

2.1. Algorytm Euklidesa w wersji z odejmowaniem

Algorytm Euklidesa w wersji z odejmowaniem polega na odejmowaniu mniejszej liczby od większej i zastępowaniu większej liczby otrzymaną różnicą, dopóki większa liczba i otrzymana różnica nie będą równe.

Przyjmijmy, że mamy dwie deski o wymiarach 36 cm i 8 cm. Chcemy otrzymać z obydwu desek możliwie najdłuższe deseczki o takich samych długościach. Szukamy zatem wspólnej miary obydwu desek. Deski będziemy przycinać zgodnie z algorytmem Euklidesa.



Rys. 1. Przykład zastosowania algorytmu Euklidesa w praktyce

Każdą z desek należy pociąć na deseczki o długości 4 cm.

a	b	Czy a > b?	Działania	Działania
36	**8**	TAK	od większej liczby odejmujemy mniejszą $a - b = 36 - 8 = 28$	zastępujemy większą liczbę otrzymaną różnicą: $a = 28, b = 8$
28	**8**	TAK	od większej liczby odejmujemy mniejszą: $a - b = 28 - 8 = 20$	zastępujemy większą liczbę otrzymaną różnicą: $a = 20, b = 8$
20	**8**	TAK	od większej liczby odejmujemy mniejszą: $a - b = 20 - 8 = 12$	zastępujemy większą liczbę otrzymaną różnicą: $a = 12, b = 8$
12	**8**	TAK	od większej liczby odejmujemy mniejszą: $a - b = 12 - 8 = 4$	zastępujemy większą liczbę otrzymaną różnicą: $a = 4, b = 8$
4	**8**	NIE	od większej liczby odejmujemy mniejszą: $b - a = 8 - 4 = 4$	zastępujemy większą liczbę otrzymaną różnicą: $a = 4, b = 4$
4	**4**	kończymy algorytm, ponieważ $a = b$; przyjmujemy, że NWD jest równe pierwszej z liczb		

Tabela 1. Analiza algorytmu Euklidesa w wersji z odejmowaniem

Ćwiczenie 2. Analizujemy działanie algorytmu Euklidesa w wersji z odejmowaniem

1. Przeanalizuj za pomocą tabeli 1. przykład działania algorytmu Euklidesa w wersji z odejmowaniem.
2. Sprawdź działanie tego algorytmu i wyznacz *NWD* dla liczb: 9 i 33. Zapisz rozwiązanie w tabeli podobnej do tabeli 1. Wykonaj również rysunek na wzór rysunku 1.

2.2. Algorytm Euklidesa w wersji z dzieleniem

W wersji z odejmowaniem algorytmu Euklidesa odejmowanie posłużyło znalezieniu reszty z dzielenia dwóch liczb. W jaki sposób możemy zamiast odejmowania zastosować dzielenie z resztą?

Wersja z dzieleniem algorytmu Euklidesa polega na wykonywaniu kolejnych dzieleń liczby a (dzielnej) przez liczbę b (dzielnik), aż reszta z dzielenia osiągnie wartość 0.

Działanie algorytmu Euklidesa w wersji z dzieleniem pokażemy na tym samym przykładzie, czyli dla liczb 36 i 8.

a	b	Działania	Czy reszta = 0?	Działania
36	**8**	dzielimy (z resztą) liczbę *a* przez *b* $a : b = 36 : 8 = 4$ reszta 4	NIE	liczbę *a* zastępujemy liczbą *b*, a liczbę *b* resztą z dzielenia
8	**4**	dzielimy (z resztą) liczbę *a* przez *b* $a : b = 8 : 4 = 2$ reszta 0	TAK	kończymy algorytm, ponieważ reszta z dzielenia wynosi 0; przyjmujemy, że NWD jest równe ostatniej niezerowej reszcie, czyli NWD = 4

Tabela 2. Analiza algorytmu Euklidesa w wersji z dzieleniem

Ćwiczenie 3. Analizujemy działanie algorytmu Euklidesa w wersji z dzieleniem

1. Przeanalizuj przykład działania algorytmu Euklidesa w wersji z dzieleniem z tabeli 2.
2. Sprawdź działanie tego algorytmu i wyznacz *NWD* dla liczb: 9 i 33. Zapisz rozwiązanie w tabeli podobnej do tabeli 2.

Znajdowanie największego wspólnego dzielnika (NWD) ma zastosowanie w obliczeniach, w których posługujemy się ułamkami zwykłymi (co omówimy w tematach 87. i 98.). Na przykład NWD możemy wykorzystać do skracania ułamków zwykłych – licznik i mianownik dzielimy przez ich NWD. Jeśli licznik i mianownik ułamka $\frac{15}{20}$ podzielimy przez 5, otrzymamy $\frac{3}{4}$.

Warto zapamiętać

- Do znajdowania liczb pierwszych można zastosować podejście polegające na sprawdzaniu kolejnych potencjalnych dzielników danej liczby.
- Największy wspólny dzielnik (NWD) dwóch liczb naturalnych możemy znaleźć, stosując algorytm Euklidesa. Algorytm ten możemy zrealizować w dwóch wersjach:
  - z odejmowaniem – od większej liczby odejmujemy mniejszą i zastępujemy większą liczbę otrzymaną różnicą, dopóki większa liczba i otrzymana różnica nie będą równe;
  - z dzieleniem – dzielimy liczbę *a* (dzielną) przez liczbę *b* (dzielnik), aż reszta z dzielenia osiągnie wartość 0.

Pytania i polecenia

1. W jaki sposób można sprawdzić, czy dana liczba jest liczbą pierwszą?
2. Do czego służy algorytm Euklidesa?
3. Na czym polega algorytm Euklidesa w wersji z odejmowaniem? Dlaczego warunkiem zakończenia tego algorytmu jest $a = b$?
4. Na czym polega algorytm Euklidesa w wersji z dzieleniem?
5. Czym różni się wersja z odejmowaniem algorytmu Euklidesa od wersji z dzieleniem?

Zadania

1. Sprawdź, która z podanych liczb: 11, 24, 97 jest liczbą pierwszą.
2. Korzystając z algorytmu Euklidesa, znajdź NWD liczb 55 i 15:
   a. w wersji z odejmowaniem,
   b. w wersji z dzieleniem.
   W każdym z przypadków wykonaj tabelę podobną odpowiednio do tabel 1. i 2.

Dla zainteresowanych

3. Podaj inny, niż omówiony w tym temacie, przykład zastosowania algorytmu Euklidesa w praktyce. Wykonaj schematyczny rysunek.
4. Dowiedz się więcej na temat Euklidesa i jego algorytmu znajdowania NWD. Przedstaw te informacje w wybranej formie.

Temat 87.

Działania na ułamkach zwykłych z wykorzystaniem NWD i NWW

1. Zależność między NWD i NWW
2. Skracanie ułamków zwykłych
3. Dodawanie i odejmowanie ułamków zwykłych
4. Mnożenie i dzielenie ułamków zwykłych

Warto powtórzyć

1. Do czego służy algorytm Euklidesa?
2. W jaki sposób obliczamy NWD według algorytmu Euklidesa w wersji z odejmowaniem?
3. Na czym polega wersja z dzieleniem algorytmu Euklidesa?

1. Zależność między NWD i NWW

Znajdowanie największego wspólnego dzielnika (NWD) ma zastosowanie w obliczeniach, w których posługujemy się ułamkami zwykłymi.

W działaniach na ułamkach zwykłych będziemy potrzebowali nie tylko największego wspólnego dzielnika NWD, ale również przyda się najmniejsza wspólna wielokrotność **NWW**.

NWW N

Najmniejsza wspólna wielokrotność dwóch lub więcej liczb całkowitych jest to najmniejsza liczba różna od zera, która jest jednocześnie wielokrotnością tych liczb.

Aby wyznaczyć najmniejszą wspólną wielokrotność (NWW) dwóch liczb całkowitych, wystarczy pomnożyć je przez siebie, a następnie podzielić przez ich NWD (największy wspólny dzielnik).

Zależność między NWD a NWW liczb a i b przedstawia wzór:

$$\mathrm{NWW}(a, b) = \frac{a \cdot b}{\mathrm{NWD}(a, b)} \qquad [1.]$$

Na przykład: $\mathrm{NWW}(63, 6) = \frac{63 \cdot 6}{\mathrm{NWD}(63, 6)} = \frac{378}{3} = 126$

Uwaga !

Odpowiednie funkcje obliczające NWD i NWW: *nwd()* i *nww()*, potrzebne do zaprogramowania działań na ułamkach zwykłych omówimy w tematach 97. i 98.

Ćwiczenie 1. Obliczamy NWW dwóch liczb

Oblicz najmniejszą wspólną wielokrotność (NWW) liczb 18 i 8 według wzoru [1.].

2. Skracanie ułamków zwykłych

W następnych punktach tematu przypomnimy podstawowe działania arytmetyczne na ułamkach zwykłych (w tym skracanie ułamków), aby ułatwić zaprogramowanie tych działań w wybranym języku programowania.

Aby skrócić ułamek zwykły, należy licznik i mianownik podzielić przez ich NWD. Jeżeli NWD jest równy 1, to ułamek jest nieskracalny i oczywiście nie trzeba wykonywać dzielenia.

$$\frac{a}{b} = \frac{\frac{a}{\mathrm{NWD}(a, b)}}{b} \qquad [2.]$$

Na przykład: $\frac{45}{125} = \frac{\frac{45}{\mathrm{NWD}(45, 125)}}{\frac{125}{\mathrm{NWD}(45, 125)}} = \frac{\frac{45}{5}}{\frac{125}{5}} = \frac{9}{25}$

Ćwiczenie 2. Skracamy ułamki zwykłe

Skróć ułamek $\frac{38}{18}$ według wzoru [2.].

3. Dodawanie i odejmowanie ułamków zwykłych

Wzór na obliczenie sumy ułamków zwykłych to:

$$\frac{l_1}{m_1} + \frac{l_2}{m_2} = \frac{l_1 \cdot m_2 + l_2 \cdot m_1}{m_1 \cdot m_2} \qquad [3.]$$

Na przykład: $\frac{7}{8} + \frac{3}{5} = \frac{7 \cdot 5 + 3 \cdot 8}{8 \cdot 5} = \frac{35 + 24}{40} = \frac{59}{40}$

Można również wyznaczyć najpierw NWW mianowników m_1 i m_2: $NWW(m_1, m_2)$. Wtedy wzór będzie miał postać:

$$\frac{l_1}{m_1} + \frac{l_2}{m_2} = \frac{l_1 \cdot \frac{\mathrm{NWW}(m_1, m_2)}{m_1} + l_2 \cdot \frac{\mathrm{NWW}(m_1, m_2)}{m_2}}{\mathrm{NWW}(m_1, m_2)} \qquad [4.]$$

Na przykład: $\frac{7}{8} + \frac{3}{5} = \frac{7 \cdot \frac{40}{8} + 3 \cdot \frac{40}{5}}{40} = \frac{7 \cdot 5 + 3 \cdot 8}{40} = \frac{35 + 24}{40} = \frac{59}{40}$

Jeśli wynik jest w postaci skracalnej, można go skrócić.

Podobnie obliczamy różnicę ułamków, zamieniając we wzorach [3.] i [4.] dodawanie na odejmowanie.

Ćwiczenie 3. Obliczamy sumę i różnicę ułamków zwykłych

Oblicz sumę i różnicę ułamków: $\frac{5}{9}$ i $\frac{3}{4}$, korzystając ze wzorów [3.] i [4.].

4. Mnożenie i dzielenie ułamków zwykłych

Wzór na obliczenie iloczynu ułamków zwykłych to:

$$\frac{l_1}{m_1} \cdot \frac{l_2}{m_2} = \frac{l_1 \cdot l_2}{m_1 \cdot m_2} \qquad [5.]$$

Na przykład: $\frac{7}{8} \cdot \frac{3}{5} = \frac{7 \cdot 3}{8 \cdot 5} = \frac{21}{40}$

Jeśli wynik jest w postaci skracalnej, można go skrócić.

Aby obliczyć iloraz ułamków, należy obliczyć iloczyn pierwszego ułamka i odwrotności drugiego ułamka:

$$\frac{l_1}{m_1} : \frac{l_2}{m_2} = \frac{l_1}{m_1} \cdot \frac{m_2}{l_2} = \frac{l_1 \cdot m_2}{m_1 \cdot l_2} \qquad [6.]$$

Na przykład: $\frac{7}{8} : \frac{3}{5} = \frac{7}{8} \cdot \frac{5}{3} = \frac{7 \cdot 5}{8 \cdot 3} = \frac{35}{24}$

Jeśli wynik jest w postaci skracalnej, można go skrócić.

Ćwiczenie 4. Obliczamy iloczyn i iloraz ułamków zwykłych

Oblicz iloczyn i iloraz ułamków: $\frac{5}{9}$ i $\frac{3}{4}$, korzystając odpowiednio ze wzorów [5.] i [6.].

Pytania i polecenia

1. Czym jest NWW?
2. Jaka jest zależność między NWD a NWW?
3. W jaki sposób wykorzystujemy NWD w działaniach na ułamkach zwykłych? Pokaż na przykładzie.

Zadania

1. Objaśnij dodawanie i odejmowanie ułamków zwykłych na przykładzie ułamków: $\frac{7}{15}$ i $\frac{2}{9}$.
2. Objaśnij mnożenie i dzielenie ułamków zwykłych na przykładzie ułamków: $\frac{4}{11}$ i $\frac{8}{9}$.



Rozdział XV

Programowanie wybranych algorytmów

Temat 88. Programowanie wybranych algorytmów na tekstach

Temat 89. Programowanie algorytmu wyszukiwania wzorca w tekście metodą naiwną

Temat 90. Programowanie algorytmu szyfrowania tekstu metodą podstawieniową

Temat 91. Programowanie algorytmu szyfrowania tekstu metodą przestawieniową

Temat 92. Programowanie algorytmów porządkowania

Temat 93. Programowanie algorytmu wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia

Temat 94. Programowanie algorytmów zachłannych

Temat 95. Programowanie algorytmów rekurencyjnych i iteracyjnych

Temat 96. Programowanie zamiany reprezentacji liczb między pozycyjnymi systemami liczbowymi

Temat 97. Programowanie wybranych algorytmów na liczbach

Temat 98. Programowanie działań na ułamkach zwykłych z wykorzystaniem NWD i NWW

Temat 88. Programowanie wybranych algorytmów na tekstach

1. Zapisywanie algorytmu porównywania tekstów w językach C++ i Python
2. Zapisywanie algorytmu szukania i zliczania wystąpień znaku w tekście w językach C++ i Python

Warto powtórzyć

1. W jaki sposób deklarujemy zmienne typu tekstowego w języku C++?
2. Jak można odwołać się do pojedynczego znaku łańcucha w językach C++ lub Python?
3. Jak wyznaczamy długość łańcucha w językach C++ lub Python?
4. Na czym polega algorytm porównywania tekstów?
5. W jaki sposób można wyszukać i zliczać wystąpienia wybranego znaku w tekście?

1. Zapisywanie algorytmu porównywania tekstów w językach C++ i Python

Chcemy napisać program realizujący algorytm porównywania tekstów w wybranym języku programowania. Jakie dobrać typy danych i instrukcje?

Program realizujący algorytm porównywania tekstów napiszemy zgodnie z opisem podanym w temacie 78. W algorytmie porównywania tekstów pracujemy na tekstach i mamy do czynienia z sytuacją warunkową oraz powtarzaniem operacji. Dlatego w programach będziemy stosować zmienne typu tekstowego, instrukcję warunkową oraz instrukcję iteracyjną – niezależnie od wyboru języka programowania.

Wynikiem działania algorytmu porównywania tekstów jest prawda albo fałsz – teksty są takie same lub różne.

Aby zapisać algorytm porównywania tekstów w wybranym języku programowania, zdefiniujemy funkcję zwracającą wartość logiczną z dwoma parametrami (będącymi tymi tekstami).

Ćwiczenie 1. Analizujemy funkcję porównującą teksty

Objaśnij działanie funkcji pokazanej na rysunku 1a lub 1b, odpowiadając na pytania:

a. Co określają parametry funkcji?
b. Dlaczego w funkcji *teksty_sa_identyczne()* zastosowano instrukcję iteracyjną **`while`**, a nie **`for`**? Uzasadnij odpowiedź.
c. Dlaczego w instrukcji **`while`** zastosowano złożony warunek logiczny (koniunkcję)? Objaśnij poszczególne elementy tego warunku (omów, co sprawdzają).
d. Jaką wartość zwraca funkcja?
e. Jakie będzie działanie funkcji dla tekstów różniących się tylko wielkością liter, np. „abc" i „ABC"?

C++

```
 5  bool teksty_sa_identyczne(string t1, string t2)
 6  {
 7      int i, dl1, dl2;
 8      i = 0;
 9      dl1 = t1.length();
10      dl2 = t2.length();
11      while(i < dl1 && i < dl2 && t1[i] == t2[i])
12          i++;
13      return i == dl1 && i == dl2;
14  }
```

Rys. 1a. Funkcja porównująca teksty (C++) – ćwiczenia 1. i 2.

Python

```
def teksty_sa_identyczne(t1, t2):
    i = 0
    dl1 = len(t1)
    dl2 = len(t2)
    while i < dl1 and i < dl2 and t1[i] == t2[i]:
        i = i + 1
    return i == dl1 and i == dl2
```

Rys. 1b. Funkcja porównująca teksty (Python) – ćwiczenia 1. i 2.

Ćwiczenie 2. Piszemy program realizujący algorytm porównywania tekstów

1. Korzystając z opisu algorytmu porównywania tekstów (temat 78.), napisz specyfikację zadania. Na tej podstawie utwórz program realizujący algorytm porównywania dwóch tekstów. Zdefiniuj funkcję *teksty_sa_identyczne()* pokazaną na rysunku 1a lub 1b. Wywołaj ją w programie głównym z parametrami aktualnymi (argumentami) *tekst1* i *tekst2*. Wyprowadzaj na ekran odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Porownywanie*.
3. Przetestuj działanie programu dla kilku różnych danych.

Wskazówka: W funkcji *teksty_sa_identyczne()* przed instrukcją pętli możesz sprawdzać, czy teksty są takiej samej długości: jeśli nie, to wartością funkcji będzie *fałsz*.

2. Zapisywanie algorytmu szukania i zliczania wystąpień znaku w tekście w językach C++ i Python

Chcemy napisać program realizujący algorytm szukania i zliczania wystąpień znaku w tekście w wybranym języku programowania. Jakie dobrać typy danych i instrukcje?

W algorytmie szukania i zliczania wystąpień znaku w tekście powtarzamy sprawdzenie, czy kolejny znak tekstu jest równy podanemu znakowi – jeśli tak, zliczamy go. Mamy tu do czynienia z sytuacją warunkową i iteracją. Aby zaprogramować ten algorytm, zastosujemy instrukcję warunkową (do sprawdzenia, czy kolejny znak tekstu jest równy podanemu znakowi), instrukcję przypisania (do zwiększania wartości licznika) i iteracyjną (do powtarzania tych obydwu operacji).

Do zliczenia liczby wystąpień konkretnego znaku (np. „a") możemy zdefiniować funkcję z jednym parametrem (*tekst*) zwracającą do programu głównego liczbę znalezionych znaków „a" w tekście *tekst* (wartość zmiennej *liczba_znakow*).

Uwaga

Języki programowania często udostępniają funkcje i metody związane z operacjami na tekstach, np. w języku Python metoda `count()` szuka wystąpień znaków lub ciągów znaków w tekście i je zlicza; w języku C++ funkcja `strcmp()` porównuje dwa łańcuchy znaków.

C++

```
for(int i = 0; i < tekst.length(); i++)
    if(tekst[i] == 'a')
        liczba_znakow++;
```

Rys. 2a. Przykład zastosowania instrukcji **`for`** w funkcji wyszukującej i zliczającej znaki – ćwiczenie 3.

Python

```
for i in range(len(tekst)):
    if tekst[i] == 'a':
        liczba_znakow = liczba_znakow + 1
```

lub

```
for znak in tekst:
    if znak == 'a':
        liczba_znakow = liczba_znakow + 1
```

Rys. 2b. Przykłady zastosowania instrukcji **`for`** w funkcji wyszukującej i zliczającej *zlicz_znak()* – ćwiczenie 3.

Ćwiczenie 3. Piszemy program zliczający wystąpienia znaku „a" w tekście

1. Napisz program wyszukujący i zliczający literę „a" w tekście wprowadzanym z klawiatury. Zapisz specyfikację tego zadania. Zdefiniuj funkcję *zlicz_znak* z jednym parametrem *tekst*. Wywołaj funkcję w programie głównym z parametrem aktualnym (argumentem) *napis*. Wyprowadzaj odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Zliczanie_znaku_a*.
3. Przetestuj działanie programu dla kilku różnych danych.

W języku C++ wprowadzanie danych za pomocą polecenia `cin >>` powoduje wczytanie danych do zmiennej, aż do napotkania tzw. białego znaku (np. spacji, tabulatora, znaku nowego wiersza). Jeśli wprowadzimy dwa wyrazy oddzielone spacją, to drugi wyraz nie zostanie wczytany. Podajemy dwa przykładowe sposoby poradzenia sobie z tym problemem:

- Jeśli łańcuch deklarujemy jako tablicę znaków, możemy zastosować instrukcję `cin.get()`, która czeka na naciśnięcie dowolnego klawisza klawiatury i natychmiast zwraca jego kod ASCII. Wykorzystywana jest więc do odczytania „znak po znaku" całego napisu, aż do odczytania kodu znaku nowego wiersza.
- Jeśli zastosujemy klasę standardową `string` (łańcuch zadeklarujemy jako `string`), możemy wykorzystać funkcję `getline()`, umożliwiającą wczytywanie całego wiersza. Naciśnięcie klawisza **Enter** kończy wprowadzanie danych. Na przykład zamiast instrukcji `cin >> napis` należy użyć `getline(cin, napis)`.

W języku Python do zliczania znaków w tekście możemy zastosować metodę `count()`, na przykład dla tekstu zapisanego w zmiennej *tekst*, w którym chcemy zliczyć litery „a", zastosujemy ją w następujący sposób: `tekst.count("a")`.

Ćwiczenie 4. Modyfikujemy program zliczający znaki

1. Zmodyfikuj program zapisany w ćwiczeniu 3.

   C++ Zastosuj funkcję `getline()` do wprowadzania napisów składających się z wyrazów oddzielonych spacjami.

   Python Zastosuj metodę `count()`.
2. Zapisz program w pliku pod nazwą *Zliczanie_znaku_a_zmod*.
3. Przetestuj działanie programu dla różnych danych.

Jeśli funkcja miałaby zliczać dowolny znak tekstu, który wprowadzalibyśmy (tak jak tekst) z klawiatury, należy dodać do definicji funkcji drugi parametr – określający ten znak.

Warto zapamiętać

- Aby zaprogramować algorytm porównywania dwóch tekstów, możemy zdefiniować funkcję logiczną zwracającą wartość z dwoma parametrami, które są tymi tekstami. W funkcji porównujemy kolejne znaki każdego z tekstów, zaczynając od pierwszego znaku dopóki *i*-te znaki będą różne lub osiągniemy koniec jednego z tekstów lub obydwu. Zastosujemy w funkcji instrukcję iteracyjną **`while`**, ponieważ nie znamy z góry liczby kroków iteracji.
- Aby zaprogramować algorytm szukania i zliczania wystąpień znaku, możemy zdefiniować funkcję zwracającą wartość, czyli liczbę znalezionych znaków. W funkcji sprawdzamy każdy znak tekstu, zaczynając od pierwszego znaku do ostatniego. Zastosujemy w funkcji instrukcję iteracyjną **`for`**, ponieważ znamy z góry liczbę kroków iteracji (określa ją długość tekstu).

Pytania i polecenia

1. Wyjaśnij sposób zapisania algorytmu porównywania tekstów w wybranym języku programowania (C++ lub Python).
2. Dlaczego funkcję porównującą teksty *teksty_sa_identyczne()* zdefiniowano tak, aby zwracała wartość logiczną?
3. Dlaczego w programie realizującym algorytm szukania i zliczania wystąpień znaku zastosowano instrukcję iteracyjną **`for`**, a nie **`while`**?
4. Ile razy w programie realizującym algorytm szukania i zliczania wystąpień znaku może być wykonana instrukcja zliczająca znaki? Rozważ różne przypadki dla tekstu o długości dziesięciu znaków.

Zadania

1. Zmodyfikuj program zapisany w ćwiczeniu 3., aby zliczany znak był wprowadzany z klawiatury po uruchomieniu programu. Zapisz plik pod tą samą nazwą.
2. Napisz program zliczający, ile znaków spacji występuje w tekście wprowadzanym z klawiatury (*tekst*). Liczba znaków spacji (*liczba_sp*) ma być wyświetlona na ekranie monitora. Zapisz program w pliku pod nazwą *Zliczanie_spacji*.
   Wskazówka:

   C++
   ```
   if(tekst[i] == ' ')
      liczba_sp++;
   ```

   Python
   ```
   if tekst[i] == ' ':
      liczba_sp = liczba_sp + 1
   ```

3. Napisz program zliczający znaki: „*”, „%” i „x”, wśród *n* znaków wprowadzanych z klawiatury. Zapisz program w pliku pod nazwą *Zliczanie_znakow*.

Dla zainteresowanych

4. Napisz program zliczający wszystkie samogłoski (bez polskich znaków diakrytycznych) w podanym tekście. Zapisz program w pliku pod nazwą *Samogloski*.

Temat 89. Programowanie algorytmu wyszukiwania wzorca w tekście metodą naiwną

1. Funkcja wyszukująca wzorzec w tekście metodą naiwną
2. Wywoływanie w programie głównym funkcji wyszukującej wzorzec w tekście

Warto powtórzyć

1. Na czym polega algorytm wyszukiwania wzorca w tekście metodą naiwną?
2. Kiedy kończymy wyszukiwanie wzorca w tekście?

1. Funkcja wyszukująca wzorzec w tekście metodą naiwną

Chcemy napisać program realizujący algorytm wyszukiwania wzorca w tekście metodą naiwną. W jaki sposób napisać w wybranym języku programowania funkcję znajdującą wzorzec w tekście?

Program realizujący algorytm wyszukiwania wzorca w tekście napiszemy zgodnie z opisem podanym w temacie 79. W wyniku działania programu chcemy otrzymać informację, czy znaleziono wzorzec, a jeśli tak, to na jakiej pozycji (indeks znaku tekstu, od którego znaleziono wzorzec). Ponadto program ma wyszukać wszystkie wzorce w podanym tekście, czyli podać pozycje wszystkich znalezionych wzorców.

Do wyszukiwania wzorca w tekście zdefiniujemy funkcję *znajdz_wzorzec()* z trzema parametrami: *tekst*, *wzorzec*, *pozycja_startowa*. Trzeci parametr określa, od którego znaku *tekstu* ma być wyszukiwany wzorzec.

W funkcji *znajdz_tekst()* zastosujemy pętle zagnieżdżone **`for`**:

- w pętli zewnętrznej przeglądamy kolejne znaki tekstu, począwszy od *pozycji_startowej*,
- w pętli wewnętrznej przeglądamy kolejne znaki wzorca, porównując je ze znakami tekstu. W przypadku rozbieżności, przerywamy pętlę (instrukcja **`break`**).

Funkcja *znajdz_tekst()* zwraca indeks pierwszego znaku wzorca, gdy wzorzec znaleziono lub –1, gdy wzorca nie znaleziono.

Zanim napiszemy cały program, przeanalizujemy działanie przykładowej funkcji znajdującej wzorzec w tekście (rys. 1a i 1b).

C++

```
 5  int znajdz_wzorzec(string tekst, string wzorzec, int pozycja_startowa)
 6  {
 7      int dl1 = tekst.length();
 8      int dl2 = wzorzec.length();
 9
10      for(int i = pozycja_startowa; i < dl1-dl2+1; i++)
11      {
12          int j;
13          for(j = 0; j < dl2; j++)
14              if(tekst[i+j] != wzorzec[j])
15                  break;
16          if(j == dl2)
17              return i;
18      }
19      return -1;
20  }
```

Rys. 1a. Przykład funkcji znajdującej wzorzec w tekście (C++)

Python

```
def znajdz_wzorzec(tekst, wzorzec, pozycja_startowa):
    dl1 = len(tekst)
    dl2 = len(wzorzec)

    for i in range (pozycja_startowa, dl1-dl2+1):
        for j in range(dl2):
            if tekst[i+j] != wzorzec[j]:
                break
        if tekst[i+j] == wzorzec[j]:
            return i
    return -1
```

Rys. 1b. Przykład funkcji znajdującej wzorzec w tekście (Python)

Przykład 1. Analizowanie realizacji algorytmu wyszukiwania wzorca w tekście z wykorzystaniem funkcji *znajdz_wzorzec()*

Analizę funkcji przeprowadzimy dla następujących danych:

`przeszukiwany_tekst = "matematyka"` `dl1 = 10`

`wzorzec = "mat"` `dl2 = 3`

`pozycja startowa = 0`

`dl1 - dl2 + 1 = 10 - 3 + 1 = 8`

W tabeli 1. pokazano analizę jednego wywołania funkcji (dla realizacji w języku C++), w którym znajdujemy pierwsze wystąpienie wzorca „mat".

Wartości				
i	j	tekst[i+j]	wzorzec[j]	
0		m	-	i < 8, wejście do pętli wewnętrznej
0	0	m	m	j < 3, wejście do porównania
0	0	m	m	tekst[i+j] == 'm' wzorzec[j] == 'm' nie ma rozbieżności, kolejna iteracja pętli wewnętrznej, zwiększenie j o 1
0	1	a	a	tekst[i+j] == 'a' wzorzec[j] == 'a' nie ma rozbieżności, kolejna iteracja pętli wewnętrznej, zwiększenie j o 1
0	2	t	t	tekst[i+j] == 't' wzorzec[j] == 't' nie ma rozbieżności, zwiększenie j o 1
0	3	m	-	j nie jest mniejsze od 3, zakończenie pętli wewnętrznej
0	3	m	-	spełniony jest warunek j == dl2, a zatem funkcja kończy się, zwracając aktualną wartość i (0)

Tabela 1. Przykładowa analiza rozwiązania w języku C++ – znajdowanie pierwszego wystąpienia wzorca w tekście



Ćwiczenie 1. Analizujemy realizację algorytmu wyszukiwania wzorca w tekście z wykorzystaniem funkcji *znajdz_wzorzec()*

1. Zapoznaj się z wybraną definicją funkcji (rys. 1a lub 1b). Korzystając z niej, przeanalizuj znajdowanie pierwszego wystąpienia wzorca w tekście. Utwórz tabelę podobną do tabeli 1.
2. Odpowiedz na pytania:
   a. Jakie kolejno wartości przyjmuje zmienna *i*, a jakie zmienna *j*?
   b. Jaką wartość zwraca funkcja?

2. Wywoływanie w programie głównym funkcji wyszukującej wzorzec w tekście

Zaczniemy od napisania programu, który znajdzie pierwsze wystąpienie wzorca w tekście i wyświetli jego pozycję. Funkcję *znajdz_wzorzec()* wywołamy w programie głównym. Sprawdzimy w ten sposób w praktyce realizację algorytmu pokazaną w tabeli 1.

Ćwiczenie 2. Piszemy program znajdujący pierwsze wystąpienie wzorca w tekście

1. W wybranym języku (C++ lub Python) utwórz program, który znajdzie pierwsze wystąpienie wzorca i wyświetli jego pozycję lub komunikat „Wzorca nie znaleziono", gdy nie znajdzie wzorca w podanym tekście.
2. Zdefiniuj funkcję *znajdz_wzorzec()* z trzema parametrami (rys. 1a lub 1b).
3. Wywołaj funkcję *znajdz_wzorzec()* w programie głównym z parametrami aktualnymi (argumentami) *t*, *w*, *p*, określającymi odpowiednio przeszukiwany tekst, szukany tekst i pozycję tekstu *t*, od której zaczyna się przeszukiwanie. Wartości parametrów aktualnych wprowadzaj z klawiatury.
4. Zapisz program w pliku pod nazwą *Wyszukiwanie_pierwszego_wzorca*.
5. Przetestuj działanie programu dla kilku różnych danych.

Wskazówka: Funkcję *znajdz_wzorzec()* można wywołać w następujący sposób:

C++
```
p = znajdz_wzorzec(t, w, p);
```
Python
```
p = znajdz_wzorzec(t, w, p)
```

Po wykonaniu ćwiczenia 2. zauważamy, że jednokrotne wywołanie funkcji znajduje tylko pierwsze wystąpienie wzorca. Dlatego w programie głównym należy jeszcze zastosować odpowiednio instrukcję pętli, aby program powtarzał wyszukiwanie, aż wzorzec przestanie być znajdowany, i podał wszystkie wystąpienia wzorca (rys. 2a i 2b).

C++
```
do
{
    p = znajdz_wzorzec(t, w, p);
    if(p >= 0)
    {
        cout << "Znaleziono: | " << t << " | " << w << " | " << p << " | " << endl;
        p++;
    }
} while(p >= 0);
```

Rys. 2a. Zastosowanie instrukcji iteracyjnej w celu wyszukania wszystkich wystąpień wzorca w tekście – fragment programu głównego (C++)

Python
```
p = 0
while p >= 0:
    p = znajdz_wzorzec(t, w, p)
    if p >= 0:
        print("Znaleziono: | ", t, " | ", w, " | ", p, " | ")
        p = p + 1
```

Rys. 2b. Zastosowanie instrukcji iteracyjnej w celu wyszukania wszystkich wystąpień wzorca w tekście – fragment programu głównego (Python)

Ćwiczenie 3. Piszemy program znajdujący wszystkie wystąpienia wzorca w tekście

1. Zmodyfikuj program zapisany w ćwiczeniu 2., aby wyświetlał pozycje wszystkich znalezionych wzorców. Program powinien również wyświetlać komunikat „Wzorca nie znaleziono", gdy nie znajdzie wzorca w podanym tekście.
2. Wyprowadzaj również odpowiednie komunikaty, np. tak jak pokazano na rysunkach 3a i 3b.
3. Zapisz program w pliku pod nazwą *Wyszukiwanie_wzorca*.
4. Przetestuj działanie programu dla kilku różnych danych.

```
Wprowadz tekst do przeszukania: matematyka
Wprowadz tekst do wyszukania, czyli wzorzec: mat
Znaleziono: | matematyka | mat | 0 |
Znaleziono: | matematyka | mat | 4 |
```

Rys. 3a. Przykładowy wynik działania programu – ćwiczenie 3.

```
Wprowadz tekst do przeszukania: informatyka
Wprowadz tekst do wyszukania, czyli wzorzec: kg
Wzorca nie znaleziono
```

Rys. 3b. Przykładowy wynik działania programu w przypadku nie znalezienia wzorca – ćwiczenie 3.

Warto zapamiętać

- Aby zaprogramować algorytm znajdowania wzorca w tekście, możemy zdefiniować funkcję zwracającą wartość z trzema parametrami, będącymi tekstem, wzorcem i pozycją startową. Funkcja zwraca indeks pierwszego znaku wzorca, gdy wzorzec znaleziono lub –1, gdy wzorca nie znaleziono.
- W funkcji znajdowania wzorca w tekście można zastosować pętle zagnieżdżone **`for`**:
  - pętlę zewnętrzną – do przeglądania kolejnych znaków tekstu, począwszy od pozycji określającej znak tekstu, od którego ma być wyszukiwany wzorzec;
  - pętlę wewnętrzną – do przeglądania kolejnych znaków wzorca, porównując je ze znakami tekstu.

Pytania i polecenia

1. Wyjaśnij, w jaki sposób można zapisać algorytm znajdowania wzorca w tekście w wybranym języku programowania (C++ lub Python)?
2. Dlaczego w funkcji *znajdz_wzorzec()* zastosowano zagnieżdżone instrukcje **`for`**?
3. Czym jest pozycja znalezionego wzorca?
4. W jaki sposób można wywołać funkcję *znajdz_wzorzec()* w programie głównym?

Zadania

1. Napisz program sprawdzający, czy podane przez użytkownika polskie imię jest imieniem męskim czy żeńskim.
   Wskazówka: Typowe polskie imiona żeńskie kończą się na literę „a". Niektóre imiona męskie również kończą się na literę „a", ale twój program nie musi tego uwzględniać.
2. Napisz program wyszukujący i zliczający litery „a" lub „b" lub „c" w tekście wprowadzanym z klawiatury zależnie od podanego z klawiatury znaku opcji. Zapisz program w pliku pod nazwą *Zliczanie_znaków*.
   Wskazówka: Możesz użyć instrukcji wyboru **`switch`** (C++) lub instrukcji **`if`** z klauzulą **`elif`** (Python).
3. Napisz program, który zależnie od wprowadzonego znaku wyprowadza na ekran: dla „i" – twoje imię, dla „n" – twoje nazwisko, a dla znaku innego niż „i" oraz „n" – komunikat „błędny znak".

Dla zainteresowanych

4. Napisz program, który sprawdzi, z ilu wyrazów składa się wprowadzone zdanie i wyświetli na ekranie ich liczbę. Zakładamy, że jest to zdanie zbudowane z wyrazów oddzielonych spacjami oraz zakończone kropką.
5. Zmodyfikuj program z zadania 4., aby wyprowadził każdy wyraz w oddzielnym wierszu na ekranie.
6. Napisz program sprawdzający, czy dany ciąg znaków to palindrom. Zapisz program w pliku pod nazwą *Palindrom*.
   Wskazówki:
   - Palindrom to ciąg znaków, który czyta się tak samo od lewej, jak i od prawej strony. Palindronami mogą być słowa, liczby czy dowolne ciągi znaków, np. *kajak*, *potop*, *zaraz, ala*, *12521*, *565*, *gk@kg*, *kos-sok*.
   - Aby sprawdzić, czy ciąg znaków jest palindromem, możemy porównać pierwszy znak z ostatnim, drugi z przedostatnim itd. Aby nie porównywać tych samych znaków podwójnie, czyli ostatniego z pierwszym, przedostatniego z drugim itd., należy ograniczyć sprawdzanie do połowy długości ciągu.

Temat 90

Programowanie algorytmu szyfrowania tekstu metodą podstawieniową

1. Zamienianie litery na jej pozycję w alfabecie i odwrotnie
2. Definiowanie funkcji szyfrującej
3. Definiowanie funkcji deszyfrującej

Warto powtórzyć

1. Na czym polega algorytm szyfrowania tekstów metodą podstawieniową?
2. Czym charakteryzuje się szyfr Cezara?
3. Która funkcja umożliwia wyznaczanie długości łańcucha w języku C++ lub Python?

1. Zamienianie litery na jej pozycję w alfabecie i odwrotnie

Chcemy napisać program realizujący algorytm szyfrowania metodą Cezara. W jaki sposób zaprogramować ten algorytm?



Aby zaprogramować algorytm, w którym mamy zastąpić daną literę inną literą umiejscowioną w alfabecie o trzy pozycje w prawo względem znaku źródłowego, musimy wiedzieć, jak wyznaczyć tę pozycję litery w alfabecie. Ponadto działania musimy powtórzyć tyle razy, ile wynosi długość tekstu do zaszyfrowania – mamy więc do czynienia z iteracją.

Jak wiemy, każda litera ma swój odpowiednik liczbowy w kodzie ASCII, np. litera A ma kod 65, a Z – 90. W tabeli kodów ASCII litery alfabetu łacińskiego ustawiono po kolei, co możemy wykorzystać do szyfru. Chcemy jednak literom alfabetu przyporządkować kolejne liczby od 0 do 25, czyli zamienić literę na liczbę określającą jej pozycję w alfabecie. Jak to zrobić? Można od kodu ASCII danej wielkiej litery odjąć kod ASCII litery A równy 65. Na przykład:

dla B $66 - 65 = 1$,
dla L $76 - 65 = 11$,
dla Z $90 - 65 = 25$.

Otrzymaliśmy w ten sposób dla każdej litery jej pozycję w alfabecie (przyjęto numerowanie od zera, aby ułatwić wykonywanie operacji dodawania modulo 26, czyli obliczania reszty z dzielenia sumy liczb przez 26).

Litera	A	B	C	D	E	F	G	H	I	J	K	L	M
Wartość	0	1	2	3	4	5	6	7	8	9	10	11	12

Litera	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Wartość	13	14	15	16	17	18	19	20	21	22	23	24	25

Tabela 1. Przyporządkowanie literom kolejnych liczb naturalnych

Aby zamieniać literę na jej pozycję w alfabecie (czyli znak na kod) i odwrotnie (kod na znak), zdefiniujemy dwie funkcje: *znak_na_kod()* i *kod_na_znak()* – rysunki 1a (C++) i 1b (Python).

W języku C++ znak jest tożsamy z jego kodem ASCII, dlatego nie musimy w programie zamieniać liczby na jej kod ASCII – zrobi to automatycznie kompilator. Użyjemy tylko wbudowanej funkcji `toupper()`, która zwraca znak przekazany jako parametr funkcji zamieniony z małej litery na wielką (wielkie litery nie są zmieniane).

Uwaga !

Typ danych **`unsigned int`** określa nieujemne liczby całkowite.

C++

```
unsigned int znak_na_kod(char znak)
{
    return toupper(znak) - 'A';
}

unsigned char kod_na_znak(int kod_znaku)
{
    return kod_znaku + 'A';
}
```

Rys. 1a. Przykładowe funkcje zmieniające literę na jej pozycję w alfabecie i odwrotnie (C++)

W języku Python, inaczej niż w języku C++, znak nie jest tożsamy z liczbą, dlatego do zamiany litery na jej kod Unicode (i odwrotnie) musimy użyć wbudowanych funkcji: `ord()`, która zamienia znak na kod Unicode i `chr()`, która zamienia kod Unicode na odpowiadający mu znak. Użyjemy również metody `upper()`, która zwraca znak zamieniony z małej litery na wielką.

Python

```
def znak_na_kod(znak):
    return ord(znak.upper()) - ord('A')

def kod_na_znak(kod_znaku):
    return chr(kod_znaku + ord('A'))
```

Rys. 1b. Przykładowe funkcje zmieniające literę na jej pozycję w alfabecie i odwrotnie (Python)

W przypadku liter alfabetu łacińskiego, cyfr i znaków interpunkcyjnych dostępnych na klawiaturze, kod Unicode tych znaków jest taki sam, jak ich kod ASCII. Zbiór kodów ASCII jest podzbiorem zbioru kodów Unicode.

Ćwiczenie 1. Testujemy działanie funkcji zmieniającej literę na jej pozycję w alfabecie i odwrotnie

1. Przetestuj działanie funkcji pokazanych na rysunku 1a lub 1b. Napisz program, definiując te funkcje i wywołując je w programie głównym z parametrem aktualnym *litera*. Znak ma być wprowadzany z klawiatury.
2. Zapisz program w pliku pod nazwą *Zamiany*.
3. Sprawdź działanie programu dla liter E oraz e. Co zauważasz?

2. Definiowanie funkcji szyfrującej

Zastanówmy się teraz, jak otrzymać pozycję zakodowanego znaku. Czy wystarczy do liczby określającej pozycję litery dodać wartość klucza, czyli np. 3?

Dla litery L: 11 + 3 = 14 po zakodowaniu otrzymamy literę O, ale dla litery Z: 25 + 3 = 28 po zakodowaniu powinniśmy otrzymać literę C, która w alfabecie ma pozycję 2.

Obliczenie samej sumy nie wystarczy. Aby otrzymywać zawsze poprawną pozycję zaszyfrowanego znaku, musimy obliczać resztę z dzielenia sumy pozycji znaku tekstu jawnego i klucza przez 26.

Zatem dla litery Z poprawne działanie to:

(25 + 3) % 26 = 28 % 26 = 2 (gdzie operator % oznacza resztę z dzielenia).

Dlatego w funkcji *zaszyfruj()* realizującej algorytm szyfrowania metodą Cezara (rys. 2a i 2b) należy umieścić odpowiednio przypisanie:

C++

```
unsigned int kod_znaku_zaszyfrowany = (kod_znaku + 3) % 26;
```

Python

```
kod_znaku_zaszyfrowany = (kod_znaku + 3) % 26
```

Funkcję *zaszyfruj()* zdefiniujemy z jednym parametrem *napis*, określającym tekst jawny. Zastosujemy pętlę **for**, w której przeglądamy kolejne znaki tekstu, począwszy od pozycji 0. Zanim napiszemy cały program, spróbujmy przeanalizować działanie tej funkcji (rys. 2a i 2b).

C++

```
17  string zaszyfruj(string napis)
18  {
19      string wynik = "";
20      for(int i = 0; i < napis.length(); i++)
21      {
22          unsigned int kod_znaku = znak_na_kod(napis[i]);
23          unsigned int kod_znaku_zaszyfrowany =  (kod_znaku + 3) % 26;
24          wynik += kod_na_znak(kod_znaku_zaszyfrowany);
25      }
26      return wynik;
27  }
```

Rys. 2a. Przykładowa funkcja realizująca algorytm szyfrowania metodą Cezara (C++)

Python

```
def zaszyfruj(napis):
    wynik = ""
    for znak in napis:
        kod_znaku = znak_na_kod(znak)
        kod_znaku_zaszyfrowany = (kod_znaku + 3) % 26
        wynik += kod_na_znak(kod_znaku_zaszyfrowany)
    return wynik
```

Rys. 2b. Przykładowa funkcja realizująca algorytm szyfrowania metodą Cezara (Python)

W przypadku zmiennych tekstowych zapis:
`wynik += kod_na_znak(kod_znaku_zaszyfrowany)`
oznacza dołączenie do tekstu w zmiennej wynik znaku po prawej stronie operatora += (rys. 2a i 2b).

Ćwiczenie 2. Piszemy program realizujący algorytm szyfrowania metodą Cezara

1. Napisz w wybranym języku programowania program umożliwiający zaszyfrowanie metodą Cezara tekstu wprowadzonego przez użytkownika. Wykorzystaj w programie funkcje zapisane w pliku *Zamiany* w ćwiczeniu 1. oraz funkcje szyfrujące pokazane na rysunku 2a (C++) lub 2b (Python). Wyjaśnij znaczenie poszczególnych wierszy wybranej funkcji szyfrującej.
2. Wywołaj funkcję w programie głównym z parametrem aktualnym.
3. Zapisz program w pliku pod nazwą *Szyfr_Cezara*.
4. Przetestuj program dla różnych danych.

3. Definiowanie funkcji deszyfrującej

Ćwiczenie 3. Formułujemy algorytm deszyfrowania

1. Napis „SLHV" został zaszyfrowany metodą Cezara. Na przykładzie odszyfrowywania tego słowa uzasadnij, dlaczego kod znaku odszyfrowanego tekstu obliczamy tak, jak pokazano w poniższych instrukcjach przypisania.

C++

```
unsigned int kod_znaku_odszyfrowany = (kod_znaku - 3 + 26) % 26;
```

Python

```
kod_znaku_odszyfrowany = (kod_znaku - 3 + 26) % 26
```

2. Sformułuj algorytm odszyfrowywania tekstu zaszyfrowanego metodą Cezara.

Ćwiczenie 4. Piszemy funkcję deszyfrowania podstawieniowego

1. Uzupełnij program zapisany w ćwiczeniu 2. o funkcję *odszyfruj()* z jednym parametrem (*napis*) deszyfrującą tekst wprowadzony przez użytkownika.
2. Wywołaj funkcję w programie głównym.
3. Zapisz plik pod tą samą nazwą i przetestuj działanie programu dla różnych danych.

Warto zapamiętać

- Aby zapisać algorytm szyfrowania tekstu metodą Cezara w wybranym języku programowania, musimy zamienić każdą literę tekstu na liczbę określającą jej pozycję w alfabecie, a następnie obliczyć pozycję zakodowanego znaku.
- W funkcji szyfrującej stosujemy instrukcję iteracyjną `for`, ponieważ mamy z góry określoną liczbę powtórzeń operacji zamiany liter na liczbę (pozycję litery w alfabecie). Operację zamiany powtarzamy tyle razy, ile wynosi długość tekstu do zaszyfrowania.

Pytania i polecenia

1. W jaki sposób przyporządkować literom alfabetu kolejne liczby naturalne?
2. Jak obliczyć pozycję zaszyfrowanego znaku? Pokaż na przykładzie litery G.
3. Dlaczego w funkcji szyfrującej stosujemy instrukcję iteracyjną `for`?
4. W jaki sposób obliczamy kod znaku odszyfrowanego?

Zadania

1. Do programu zapisanego w ćwiczeniu 4. dodaj możliwość wprowadzania z klawiatury dowolnej wartości klucza. Zapisz program w pliku pod nazwą *Szyfr_Cezara_rozne_klucze*.
   Wskazówka: Do funkcji *zaszyfruj()* i *odszyfruj()* dodaj drugi parametr *pozycja*.

Dla zainteresowanych

2. Do programu zapisanego w zadaniu 1. dodaj możliwość wyboru szyfrowania lub deszyfrowania bądź zakończenia programu. Zapisz program w pliku pod nazwą *Szyfr_Cezara_rozne_klucze_wybor*.

Temat 91. Programowanie algorytmu szyfrowania tekstu metodą przestawieniową

1. Programowanie algorytmu tworzącego anagramy
2. Programowanie algorytmu szyfrowania przestawieniowego

Warto powtórzyć

1. Czym jest anagram?
2. W jaki sposób odwołujemy się do pojedynczego znaku łańcucha?
3. Na czym polega algorytm szyfrowania tekstów metodą przestawieniową?
4. Co oznacza zapis: `suma += a`? Podaj odpowiednik tego zapisu.

1. Programowanie algorytmu tworzącego anagramy

Chcemy napisać w wybranym języku program, który tworzy najprostszy anagram, czyli ustawia litery w odwrotnej kolejności. Jak to zrobić?

Aby napisać program tworzący taki anagram, wystarczy wiedzieć, jak odwołać się do pojedynczego znaku napisu, a także jak użyć instrukcji iteracyjnej do wyświetlania napisu od końca do początku.

Ćwiczenie 1. Piszemy program tworzący anagramy w wybranym języku programowania

1. W wybranym języku programowania napisz program tworzący anagram słowa podanego z klawiatury. Anagram ma być przestawieniem liter w odwrotnej kolejności.
2. Zapisz program w pliku pod nazwą *Miniszyfrowanie*.
3. Przetestuj program dla różnych danych.

Wskazówka: Instrukcja pętli może mieć postać:

C++

```
for(int i = napis.length() - 1; i >= 0; i--)
    cout << napis[i];
```

Python

```
for znak in reversed(napis):
    print(znak)
```

W języku Python można użyć funkcji wbudowanej `reversed()` do iteracji w odwrotnej kolejności po napisie lub liście.

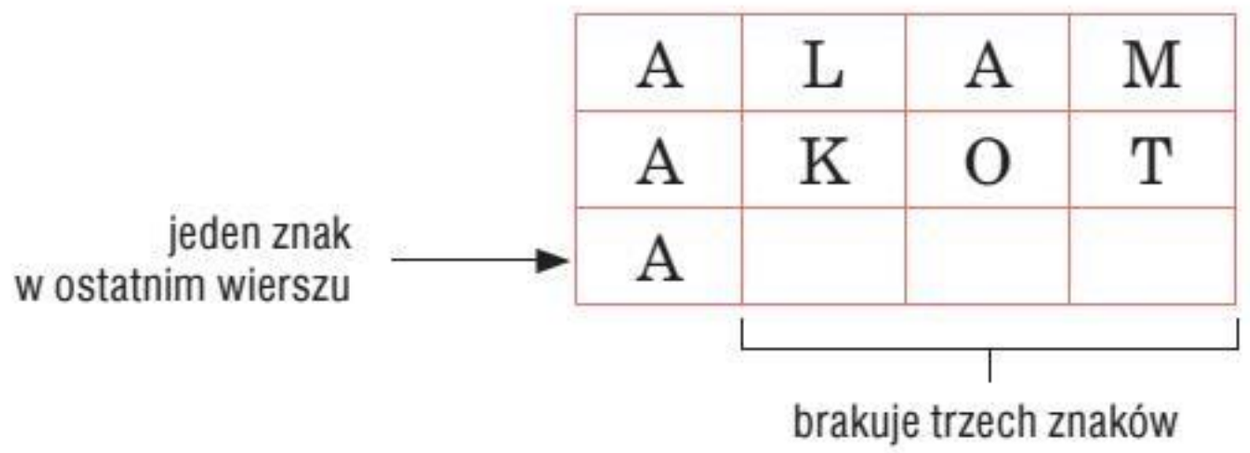
2. Programowanie algorytmu szyfrowania przestawieniowego

Chcemy napisać program realizujący algorytm szyfrowania przestawieniowego. W jaki sposób możemy napisać funkcję szyfrującą w wybranym języku programowania?

Aby zaszyfrować dany tekst metodą przestawieniową z użyciem kolumn, musimy zadbać o to, by w każdej kolumnie było tyle samo znaków. W przypadku tekstu, którego długość nie jest podzielna przez liczbę kolumn, musimy taki tekst dopełnić wybranym znakiem.

Na przykład, aby zaszyfrować tekst jawny ALAMAKOTA z użyciem czterech kolumn, trzeba będzie dopełnić tekst trzema znakami do dwunastu znaków (rys. 1.).

Zdefiniujemy funkcję pomocniczą *dopelnij()* z trzema parametrami: *napis*, *znak* i *liczba_znakow*, która zwraca tekst *napis* dopełniony znakami do długości równej *liczba_znakow*.



Rys. 1. Przykład tekstu jawnego ALAMAKOTA do zaszyfrowania z użyciem czterech kolumn

C++

```
 7  string dopelnij(string napis, char znak, int liczba_znakow)
 8  {
 9      string wynik = napis;
10      for(int i = napis.length(); i < liczba_znakow; i++)
11          wynik += znak;
12      return wynik;
13  }
```

Rys. 2a. Przykładowa pierwsza funkcja pomocnicza do szyfrowania przestawieniowego (C++)

Python

```
def dopelnij(napis, znak, liczba_znakow):
    wynik = napis
    for i in range (len(napis), liczba_znakow):
        wynik += znak
    return wynik
```

Rys. 2b. Przykładowa pierwsza funkcja pomocnicza do szyfrowania przestawieniowego (Python)

Aby skorzystać z funkcji *dopelnij()*, musimy wiedzieć, jaka powinna być długość tekstu z dopełnieniem. W tym celu obliczamy resztę z dzielenia długości tekstu przez liczbę kolumn, która to wartość odpowiadać będzie liczbie znaków w ostatnim wierszu. Jeżeli będzie ona niezerowa – długość tekstu zwiększamy o brakującą liczbę znaków.

W naszym przykładzie tekst jawny ALAMAKOTA ma długość 9, czyli: 9 % 4 = 1, czyli w ostatnim wierszu brakuje: 4 – 1 = 3 znaków.

Do tekstu jawnego dodajemy brakujące znaki, czyli długość tekstu z dopełnieniem wynosi: 9 + 3 = 12 znaków.

Do wykonania tych obliczeń zdefiniujemy funkcję *liczba_znakow_z_dopelnieniem()* z dwoma parametrami *napis* i *liczba_kolumn* (rys. 3a i 3b). Funkcja zwraca długość tekstu z dopełnieniem (wartość zmiennej *dlugosc*).

C++

```
15  int liczba_znakow_z_dopelnieniem(string napis, int liczba_kolumn)
16  {
17      int dlugosc = napis.length();
18      int liczba_znakow_ostatni_wiersz = dlugosc % liczba_kolumn;
19      if(liczba_znakow_ostatni_wiersz != 0)
20          dlugosc += (liczba_kolumn - liczba_znakow_ostatni_wiersz);
21      return dlugosc;
22  }
```

Rys. 3a. Przykładowa druga funkcja pomocnicza do szyfrowania przestawieniowego (C++)

Python

```
def liczba_znakow_z_dopelnieniem(napis, liczba_kolumn):
    dlugosc = len(napis)
    liczba_znakow_ostatni_wiersz = dlugosc % liczba_kolumn
    if liczba_znakow_ostatni_wiersz != 0:
        dlugosc += (liczba_kolumn - liczba_znakow_ostatni_wiersz)
    return dlugosc
```

Rys. 3b. Przykładowa druga funkcja pomocnicza do szyfrowania przestawieniowego (Python)

Zapisując algorytm szyfrowania metodą przestawieniową, możemy zauważyć, że czytanie tekstu kolumnami polega w istocie na odczytywaniu co *n*-tego znaku, gdzie *n* odpowiada kluczowi szyfru określonemu jako liczba kolumn. Dla powyższego przykładu, przyjmując, że znaki numerowane są od zera, powinniśmy zatem odczytać znaki o numerach: 0, 4, 8 (pierwsza kolumna), 1, 5, 9 (druga kolumna) itd. W naszym przykładzie użyjemy znaku podkreślenia „_” do dopełnienia tekstu (rys. 4.). Tego samego znaku użyjemy również w funkcji szyfrującej (rys. 5a i 5b).

Znak	A	L	A	M	A	K	O	T	A	_	_	_
Numer znaku	0	1	2	3	4	5	6	7	8	9	10	11

Rys. 4. Znaki tekstu z dopełnieniem do dwunastu znaków

Ćwiczenie 2. Szyfrujemy tekst

Zanim napiszesz program, zaszyfruj tekst ALAMAKOTA metodą przestawieniową z użyciem czterech kolumn.

Do szyfrowania tekstu utworzymy funkcję *zaszyfruj()*. Aby wybrać z tekstu co *n*-ty znak, wykorzystujemy zmienną *akt_pozycja*, w której przechowywana jest aktualna pozycja znaku tekstu niezaszyfrowanego. Pozycję tę zwiększamy o parametr *liczba_kolumn*. Jeżeli wartość zmiennej *akt_pozycja* będzie większa lub równa długości tekstu, zmniejszamy ją o długość tekstu i zwiększamy o jeden. Proces ten powtarzamy tak długo, aż wykorzystamy wszystkie znaki tekstu niezaszyfrowanego.

C++

```
24  string zaszyfruj(string napis, int liczba_kolumn)
25  {
26      string wynik = "";
27      napis = dopelnij(napis, '_', liczba_znakow_z_dopelnieniem(napis, liczba_kolumn));
28      int dlugosc = napis.length();
29      int akt_pozycja = 0;
30      for(int i = 0; i < dlugosc; i++)
31      {
32          wynik += napis[akt_pozycja];
33          akt_pozycja += liczba_kolumn;
34          if(akt_pozycja >= dlugosc)
35              akt_pozycja -= (dlugosc-1);
36      }
37      return wynik;
38  }
```

Rys. 5a. Przykładowa funkcja realizująca algorytm szyfrowania przestawieniowego (C++)

Python

```
def zaszyfruj(napis, liczba_kolumn):
    wynik = ""
    napis = dopelnij(napis, '_', liczba_znakow_z_dopelnieniem(napis, liczba_kolumn))
    dlugosc = len(napis)
    akt_pozycja = 0
    for i in range(dlugosc):
        wynik += napis[akt_pozycja]
        akt_pozycja += liczba_kolumn
        if akt_pozycja >= dlugosc:
            akt_pozycja -= (dlugosc-1)
    return wynik
```

Rys. 5b. Przykładowa funkcja realizująca algorytm szyfrowania przestawieniowego (Python)

Ćwiczenie 3. Piszemy program realizujący algorytm szyfrowania przestawieniowego

1. W wybranym języku programowania napisz program szyfrujący szyfrem przestawieniowym kolumnowym tekst wprowadzony przez użytkownika. Liczba kolumn powinna być podawana przez użytkownika.
2. Przykładowe funkcje pomocnicze i szyfrujące pokazano na rysunkach 2a, 3a, 5a (C++) i 2b, 3b, 5b (Python). Możesz się wzorować na tych rozwiązaniach. Objaśnij znaczenie poszczególnych wierszy każdej z funkcji.
3. Wywołaj funkcję szyfrującą w programie głównym.
4. Zapisz program w pliku pod nazwą *Szyfrowanie_przestawieniowe*.
5. Przetestuj program dla różnych danych, w tym dla tekstu „ALAMAKOTA".

Deszyfrowanie tekstu wykorzystuje ten sam algorytm, co szyfrowanie. Z tekstu zaszyfrowanego odczytujemy jednak nie co n-ty znak, lecz co m-ty znak, gdzie m odpowiada liczbie wierszy ($m \cdot n = liczba\ znaków\ z\ dopełnieniem$).

Ćwiczenie 4. Piszemy funkcję deszyfrowania przestawieniowego

1. Uzupełnij program zapisany w ćwiczeniu 3. o funkcję deszyfrującą tekst wprowadzony przez użytkownika. Liczba kolumn powinna być podawana przez użytkownika. Wywołaj funkcję w programie głównym.
2. Zapisz plik pod tą samą nazwą.
3. Przetestuj program dla różnych danych.

Warto zapamiętać

- W algorytmie szyfrowania metodą przestawieniową kolumnową w każdej kolumnie musi być tyle samo znaków. Programując ten algorytm w wybranym języku programowania, musimy o to zadbać. Jeśli długość tekstu nie jest podzielna przez liczbę kolumn, dopełniamy tekst wybranym znakiem.
- W programie realizującym algorytm szyfrowania metodą przestawieniową kolumnową możemy zdefiniować dwie funkcje pomocnicze:
  - zwracającą długość tekstu z dopełnieniem,
  - zwracającą tekst dopełniony znakami do odpowiedniej długości.
- Funkcje pomocnicze wykorzystujemy w funkcji szyfrującej, która zwraca zaszyfrowany tekst.

Pytania i polecenia

1. W jaki sposób napisać program, który tworzy najprostszy anagram, czyli ustawia litery w odwrotnej kolejności?

2. Dlaczego, programując algorytm metodą przestawieniową z użyciem kolumn, musimy zadbać o to, by w każdej kolumnie było tyle samo znaków?
3. Jak obliczamy długość tekstu z dopełnieniem?
4. W jakim celu tworzymy funkcje pomocnicze? Jakie wartości zwracają te funkcje?
5. Dlaczego w funkcji szyfrującej można było zastosować instrukcję iteracyjną `for`?

Zadania

1. Wypisz wszystkie anagramy słowa „pora". Wyjaśnij sposób tworzenia tych anagramów. Czy możesz zastosować ten sam sposób do innych czteroliterowych wyrazów? Uzasadnij odpowiedź na przykładzie wyrazu „rano".
2. Do programu zapisanego w ćwiczeniu 4. dodaj możliwość szyfrowania i deszyfrowania *n* tekstów wprowadzanych z klawiatury (wartość zmiennej *n* również wprowadzaj z klawiatury). Zapisz program w pliku pod nazwą *Szyfrowanie_przestawieniowe_n*.

Dla zainteresowanych

3. Napisz program wyświetlający wszystkie anagramy utworzone na podstawie wyrazu „pora". Zapisz program w pliku pod nazwą *Anagramy _pora*. Przetestuj program dla kilku innych czteroliterowych napisów.
4. Zmodyfikuj program zapisany w zadaniu 3., aby czteroliterowy wyraz można było podawać z klawiatury po uruchomieniu programu. Zapisz program w pliku pod nazwą *Anagramy _dowolny _wyraz*.
5. Do programu zapisanego w ćwiczeniu 4. dodaj możliwość wyboru rodzaju operacji (szyfrowania, deszyfrowania bądź zakończenia programu) z wykorzystaniem opcji menu. Zapisz program w pliku pod nazwą *Szyfr_przestawieniowy _wybor*.

Temat 92. Programowanie algorytmów porządkowania

1. Programowanie algorytmu porządkowania metodą bąbelkową
2. Programowanie algorytmu porządkowania przez wstawianie

Warto powtórzyć

1. Na czym polega algorytm porządkowania metodą bąbelkową?
2. Na czym polega algorytm porządkowania przez wstawianie?
3. Jak zamienić elementy tablicy (C++) lub listy (Python) miejscami?
4. W jaki sposób zdefiniować funkcję wprowadzającą elementy do tablicy (C++) lub listy (Python)?
5. Jak zdefiniować funkcję wyprowadzającą dane z tablicy (C++) lub listy (Python) na ekran?
6. Jak wywołujemy w programie głównym funkcję niezwracającą wartości bez parametrów?

1. Programowanie algorytmu porządkowania metodą bąbelkową

Chcemy napisać program porządkujący ceny wybranego artykułu od najwyższej do najniższej, czyli malejąco. Jak to zrobić w wybranym języku programowania?

Niezależnie od wyboru języka programowania, aby napisać program porządkujący liczby (tu: ceny), możemy zastosować jeden z poznanych w temacie 82. algorytmów porządkowania (metodą bąbelkową i przez wstawianie). Sortowanie w porządku malejącym pokażemy, wykorzystując algorytm porządkowania metodą bąbelkową.

Przykład 1. Określenie zadania i jego specyfikacji

Zadanie: Uporządkuj malejąco *N* liczb, stosując algorytm porządkowania metodą bąbelkową.

Dane: stała wartość *N*, tablica (lista) liczb *a*[*N*].

Wynik: tablica (lista) liczb *a*[*N*], w której liczby ułożone są w porządku malejącym.

Program realizujący algorytm porządkowania metodą bąbelkową napiszemy zgodnie z opisem podanym w temacie 82. i podaną specyfikacją.

W trakcie działania programu powinniśmy pamiętać wszystkie wprowadzone dane, dlatego wprowadzimy je do tablicy (C++) lub listy (Python).

W algorytmie mamy do czynienia z sytuacją warunkową (porównywanie elementów) oraz powtarzaniem operacji (porównywanie elementów i ewentualna zamiana). Dlatego w programach będziemy stosować zmienne indeksowane, instrukcję warunkową oraz instrukcję iteracyjną – niezależnie od wyboru języka programowania.

W programie zdefiniujemy dwie funkcje niezwracające wartości bez parametrów:

- *wprowadz_dane()* – do wprowadzenia danych,
- *wyprowadz_dane()* – do wyprowadzenia danych nieuporządkowanych i po uporządkowaniu.

Wynikiem działania algorytmu porządkowania metodą bąbelkową jest uporządkowany ciąg liczb, dlatego zdefiniujemy funkcję *sort_babel()* niezwracającą wartości bez parametrów, która posłuży nam do posortowania danych. W funkcji zastosujemy pętlę zagnieżdżoną **`for`**, a w pętli wewnętrznej instrukcję warunkową, w której, w zależności od wartości warunku, zamieniamy elementy lub pozostawiamy bez zmian.



Ćwiczenie 1. Analizujemy funkcję porządkującą liczby metodą bąbelkową

Objaśnij działanie funkcji pokazanej na rysunku 1a lub 1b, odpowiadając na pytania:

a. Dlaczego nie użyto funkcji zwracającej wartości?
b. Jakie operacje wykonujemy w pętli wewnętrznej, a jakie w zewnętrznej? Od którego elementu zaczynamy przeglądanie zbioru?
c. Wskaż miejsce zamiany elementów tablicy lub listy. W jakim celu użyto dodatkowej zmiennej *t*?

Wskazówka: W analizie funkcji może pomóc rysunek przedstawiający schematycznie porządkowanie metodą bąbelkową (rys. 1. z tematu 82.).

C++

```
23  void sort_babel()
24  {
25      for(int i = N-1; i > 0; i--)
26          for(int j = 0; j < i; j++)
27              if(a[j] < a[j + 1])
28              {
29                  int t = a[j];
30                  a[j] = a[j + 1];
31                  a[j + 1] = t;
32              }
33  }
```

Rys. 1a. Funkcja porządkująca malejąco liczby metodą bąbelkową (C++) – ćwiczenia 1. i 2.

Python

```
def sort_babel():
    for i in range(N - 1, 0, -1):
        for j in range(i):
            if a[j] < a[j+1]:
                t = a[j]
                a[j] = a[j + 1]
                a[j + 1] = t
```

Rys. 1b. Funkcja porządkująca malejąco liczby metodą bąbelkową (Python) – ćwiczenia 1. i 2.

Ćwiczenie 2. Zapisujemy algorytm porządkowania metodą bąbelkową w językach C++ lub Python

1. Napisz program porządkujący malejąco ceny wprowadzone do tablicy (listy) składającej się z pięciu elementów:
   a. zdefiniuj funkcję *wprowadz_dane()* do wprowadzania elementów do tablicy (listy) i *wyprowadz_dane()* – do ich wyprowadzenia na ekran,
   b. zdefiniuj funkcję *sort_babel()* pokazaną na rysunku 1a lub 1b,
   c. wszystkie funkcje wywołaj w programie głównym, przy czym funkcję *wyprowadz_dane()* dwukrotnie – wyświetlaj dane przed uporządkowaniem i po uporządkowaniu,
   d. wyprowadzaj na ekran odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Sortowanie_babelkowe*.
3. Uruchom i przetestuj program dla kilku różnych danych. Wyjaśnij znaczenie poszczególnych wierszy programu.

Wskazówka: Możesz skorzystać ze zdefiniowanych w tematach 63. (C++) i 75. (Python) funkcji wprowadzających dane do tablicy (listy) i wyprowadzających je na ekran.

2. Programowanie algorytmu porządkowania przez wstawianie

Chcemy napisać program porządkujący oceny z informatyki od najniższej do najwyższej, czyli rosnąco. Jaki algorytm zastosować?

Aby napisać program porządkujący liczby (tu: oceny z informatyki) rosnąco, możemy zastosować ten sam algorytm, którego użyliśmy do sortowania malejącego, ale możemy również zastosować metodę przez wstawianie.

Przykład 2. Określenie zadania i jego specyfikacji

Zadanie: Uporządkuj rosnąco ciąg elementów metodą liniową przez wstawianie.

Dane: stała wartość *N*, tablica (lista) liczb *a*[*N*], zwana tablicą (listą) pierwotną.

Wynik: tablica (lista) liczb *b*[*N*], zwana tablicą (listą) docelową, w której liczby ułożone są w porządku rosnącym.

Podobnie jak w przypadku algorytmu porządkownia metodą bąbelkową, chcemy odwoływać się do danych (tu: oceny z informatyki) po ich wprowadzeniu, dlatego również wprowadzimy je do tablicy (C++) lub listy (Python).

W programie zastosujemy funkcje niezwracające wartości bez parametrów do wprowadzania danych do tablicy (listy) oraz do wyprowadzania danych z tablicy (z listy) na ekran – te same, których użyliśmy w programie porządkującym dane metodą bąbelkową (ćwiczenie 2.).

Zdefiniujemy nową funkcję niezwracającą wartości bez parametrów: *sort_wstaw()*, która posłuży do posortowania danych.

W funkcji *sort_wstaw()*:

- definiujemy dodatkową tablicę (listę) $b[N]$, w której będziemy wstawiać na odpowiednich miejscach elementy z tablicy (listy) $a[N]$; tablica (lista) $b[N]$ na początku jest pusta (nie zawiera żadnych elementów),
- przyjęliśmy następujące oznaczenia dla zmiennych:
  - *akt* – aktualnie sprawdzany element tablicy (listy) $a[N]$,
  - *indeks_b* – indeks elementu tablicy (listy) $b[N]$, który określa miejsce wstawienia aktualnego elementu tablicy (listy) $a[N]$,
  - *licz_posort* – liczba posortowanych elementów w tablicy (liście) $b[N]$,
- stosujemy dwie pętle **`for`**, pierwsza z nich zawiera dwie pętle wewnętrzne (opis ich działania pokazano na rysunkach 2a i 2b).

C++

```
23  void sort_wstaw()
24  {
25      int b[N];
26      int licz_posort = 0;
27
28      for(int i = 0; i < N; i++)
29      {
30          int akt = a[i];
31          int indeks_b = licz_posort;
32          for(int j = 0; j < licz_posort; j++)
33              if(b[j] > akt)
34              {
35                  indeks_b = j;
36                  break;
37              }
38          for(int k = licz_posort; k > indeks_b; k--)
39              b[k] = b[k-1];
40          b[indeks_b] = akt;
41          licz_posort++;
42      }
43      for(int i = 0; i < N; i++)
44          a[i] = b[i];
45  }
```

zapamiętanie dla każdego elementu tablicy aktualnie sprawdzanego elementu

znalezienie punktu wstawienia

przesunięcie elementów w tablicy docelowej, od końca

przepisanie tablicy docelowej (posortowanej) do tablicy pierwotnej

Rys. 2a. Funkcja porządkująca rosnąco liczby metodą przez wstawianie (C++) – ćwiczenia 3. i 4.

Python

```
def sort_wstaw():
    b = [0] * N
    licz_posort = 0

    for i in range (N):
        akt = a[i]
        indeks_b = licz_posort
        for j in range(licz_posort):
            if b[j] > akt:
                indeks_b = j
                break
        for k in range(licz_posort,indeks_b, -1):
            b[k] = b[k - 1]
        b[indeks_b] = akt
        licz_posort = licz_posort + 1
    for i in range(N):
        a[i] = b[i]
```

zapamiętanie dla każdego elementu listy aktualnie sprawdzanego elementu

znalezienie punktu wstawienia

przesunięcie elementów w liście docelowej, od końca

przepisanie listy docelowej (posortowanej) do listy pierwotnej

Rys. 2b. Funkcja porządkująca rosnąco liczby metodą przez wstawianie (Python) – ćwiczenia 3. i 4.

Ćwiczenie 3. Analizujemy funkcję porządkującą liczby metodą przez wstawianie

Objaśnij działanie funkcji pokazanej na rysunku 2a lub 2b, odpowiadając na pytania:

a. Dlaczego nie użyto funkcji zwracającej wartości?
b. Jakie operacje wykonujemy w pierwszej pętli `for`? Objaśnij działanie poszczególnych instrukcji, w tym pętli wewnętrznych.

Wskazówka: W analizie funkcji może pomóc rysunek przedstawiający schematycznie porządkowanie przez wstawianie (rys. 2. z tematu 82.).

Ćwiczenie 4. Zapisujemy algorytm porządkowania przez wstawianie w językach C++ lub Python

1. Napisz program porządkujący rosnąco oceny wprowadzone do tablicy (listy) składającej się z pięciu elementów:
   a. do wprowadzania elementów do tablicy (listy) i do ich wyprowadzenia na ekran wykorzystaj funkcje *wprowadz_dane()* i *wyprowadz_dane()* zdefiniowane w ćwiczeniu 3.,
   b. zdefiniuj funkcję *sort_wstaw()* pokazaną na rysunku 2a lub 2b,
   c. wszystkie funkcje wywołaj w programie głównym, przy czym funkcję *wyprowadz_dane()* dwukrotnie – wyświetlaj dane przed uporządkowaniem i po uporządkowaniu,
   d. wyprowadzaj na ekran odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Sortowanie_wstawianie*.
3. Uruchom i przetestuj program dla kilku różnych danych. Wyjaśnij znaczenie poszczególnych wierszy programu.

Warto zapamiętać

- W algorytmach porządkowania liczby, które mają być porządkowane, wprowadzamy do tablicy (C++) lub listy (Python), aby można było się do nich odwoływać.
- W funkcji porządkującej liczby metodą bąbelkową stosujemy zagnieżdżone pętle `for`. W pętli wewnętrznej porównujemy sąsiednie liczby, stosując instrukcję warunkową. W zależności od wartości warunku, zamieniamy je miejscami. Operacje porównywania i zamiany liczb miejscami powtarzamy dla wszystkich liczb danego ciągu.
- W programie realizującym algorytm porządkowania liczb metodą przez wstawianie definiujemy dwie tablice (listy). Pierwsza (pierwotna) zawiera liczby nieuporządkowane. Z tej tablicy (listy) pobieramy kolejno elementy do drugiej (docelowej), które wstawiamy w odpowiednie miejsca tak, aby powstał uporządkowany ciąg liczb.

Pytania i polecenia

1. Jakie struktury danych powinno się zastosować, zapisując algorytmy porządkowania bąbelkowego i przez wstawianie w językach C++ lub Python?
2. Dlaczego w programach realizujących algorytmy porządkowania metodą bąbelkową i przez wstawianie zdefiniowano funkcje niezwracające wartości?
3. Dlaczego w funkcji *sort_babel()* zastosowano zagnieżdżone instrukcje **`for`**?
4. W jakim celu definiujemy drugą tablicę (listę) w funkcji sortującej liczby metodą przez wstawianie?

Zadania

1. Napisz program porządkujący malejąco liczby dostaw telewizorów dostarczanych do magazynu RTV. Zastosuj algorytm porządkowania metodą bąbelkową. Zapisz program w pliku pod nazwą *Dostawy_uporzadkowane_babel*.
2. Napisz program porządkujący rosnąco liczby dostaw telewizorów dostarczanych do magazynu RTV. Zastosuj algorytm porządkowania metodą przez wstawianie. Zapisz program w pliku pod nazwą *Dostawy_uporzadkowane_wstawianie*.

Dla zainteresowanych

3. Korzystając z wybranej metody sortowania, napisz program porządkujący rosnąco liczby rzeczywiste wprowadzone z klawiatury do pięcioelementowej tablicy (C++) lub listy (Python) według ich wartości bezwzględnych. Zapisz program w pliku pod nazwą *Wartosci_bezwgledne*.

   Wskazówki:

   C++ Funkcja `abs()` obliczająca wartość bezwzględną liczby rzeczywistej jest umieszczona w bibliotece `cmath`. Bibliotekę należy dołączyć, umieszczając na początku programu polecenie: **`#include`** `<cmath>`.

   Python Funkcja `abs()` obliczająca wartość bezwzględną liczby rzeczywistej należy do wbudowanych funkcji interpretera Pythona i jest dostępna w dowolnym momencie wykonania programu.

Temat 93.

Programowanie algorytmu wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia

1. **Funkcja wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia**
2. **Wywoływanie funkcji wyszukiwania elementu w zbiorze uporządkowanym w programie głównym**

Warto powtórzyć

1. Na czym polega algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia?
2. Jak działa instrukcja **`do ... while`** w języku C++? W przypadku jakich algorytmów iteracyjnych możemy ją zastosować?
3. W jaki sposób w języku C++ wykonać dzielenie całkowite (z obcięciem części ułamkowej)?
4. Jak działa instrukcja **`while`** w języku Python? W przypadku jakich algorytmów iteracyjnych możemy ją zastosować?
5. Czym w języku Python różni się operator dzielenia / od //?

1. Funkcja wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia

Chcemy napisać program realizujący algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia. W jaki sposób napisać taki program w wybranym języku programowania (C++ lub Python)?

Program realizujący algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia napiszemy zgodnie z opisem podanym w temacie 83. Program ma wyszukiwać w zbiorze uporządkowanym daną podaną przez użytkownika i wyświetlać na ekranie jej pozycję (indeks) w tablicy lub liście.

Zaczynamy od ustalenia, w jaki sposób będziemy pamiętać elementy zbioru. Elementy zbioru wprowadzimy do tablicy (C++) lub listy (Python). Zakładamy, że wprowadzamy dane od najmniejszej do największej (inaczej program nie będzie działał poprawnie). Bez takiego założenia należałoby dodać również funkcję sortowania danych (ustawiania w określonym porządku).

Przykład 1. Określenie zadania i jego specyfikacji

Zadanie: Wyszukaj dany element w uporządkowanym zbiorze *N* liczb, stosując metodę połowienia.

Dane: stała wartość *N*, tablica (lista) liczb *a*[*N*], szukana liczba *wartosc*.

Wynik: pozycja elementu *srodek* lub informacja, że elementu nie znaleziono.

Zdefiniujemy funkcję *znajdz_dana()* z parametrem *wartosc* (rys. 1a i 1b).

- W funkcji wyznaczamy pozycję elementu środkowego:

  C++ `srodek = (poczatek + koniec) / 2;`

  Python `srodek = (poczatek + koniec) // 2`

- Jeśli element środkowy (*a*[*srodek*]) jest szukaną daną (zmienna *wartosc*), to funkcja zwraca wartość zmiennej *srodek*, czyli pozycję danego elementu; w przeciwnym wypadku:
  - jeśli szukana dana jest mniejsza od środkowej, odrzucamy prawą połowę zbioru (liczby większe od elementu środkowego lub mu równe) – ustalamy nową wartość zmiennej *koniec*:
    `koniec = srodek - 1;`
  - w przeciwnym wypadku odrzucamy lewą połowę zbioru (liczby mniejsze od elementu środkowego lub mu równe) i ustalamy nowy początek:
    `poczatek = srodek + 1`.

- Połowienie powtarzamy, dopóki początek zbioru jest mniejszy od końca zbioru lub mu równy (`poczatek <= koniec`) – o ile wcześniej nie znajdziemy szukanego elementu (wtedy komputer wykona instrukcję **`return`** `srodek`).

- Jeśli w przeszukiwanym zbiorze nie ma szukanej liczby, funkcja *znajdz_dana()* zwraca wartość –1.

C++

```
12  int znajdz_dana(int wartosc)
13  {
14      int poczatek, koniec, srodek;
15      poczatek = 0;
16      koniec = N - 1;
17      do
18      {
19          srodek = (poczatek + koniec) / 2;
20          if(a[srodek] == wartosc)
21              return srodek;
22          else
23              if(wartosc < a[srodek])
24                  koniec = srodek - 1;
25              else
26                  poczatek = srodek + 1;
27      }
28      while(poczatek <= koniec);
29      return -1;
30  }
```

Rys. 1a. Definicja funkcji wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia (C++)

Python

```
def znajdz_dana(wartosc):
    poczatek = 0
    koniec = N - 1
    while poczatek <= koniec:
        srodek = (poczatek + koniec) // 2
        if a[srodek] == wartosc:
            return srodek
        else:
            if wartosc < a[srodek]:
                koniec = srodek - 1
            else:
                poczatek = srodek + 1
    return -1
```

Rys. 1b. Definicja funkcji wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia (Python)

Ćwiczenie 1. Analizujemy realizację algorytmu wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia z wykorzystaniem funkcji *znajdz_wzorzec()*

1. Zapoznaj się z wybraną definicją funkcji (rys. 1a lub 1b) i, korzystając z niej, przeprowadź analizę realizacji algorytmu wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia dla następującego zbioru liczb: {11, 23, 35, 44, 51}.
   Liczby pamiętane są w tablicy (liście) $a[N]$, $N = 5$. Szukany element to liczba 23.

a[0]	a[1]	a[2]	a[3]	a[4]
11	23	35	44	51

   Analizę funkcji wykonaj w tabeli (podobnej do tabeli 1. w temacie 83.), pokazując dodatkowo, jak zmieniają się wartości poszczególnych zmiennych (*poczatek*, *koniec*, *srodek*).
2. Odpowiedz na pytania:
   a. Dlaczego w funkcji *znajdz_dana()* zastosowano instrukcję iteracyjną **do** … **while** (C++) lub **while** (Python)?
   b. Jakie wartości zwraca funkcja *znajdz_dana()*?

Wskazówka: W analizie funkcji może pomóc rysunek przedstawiający schematycznie algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia (rys. 2. z tematu 83.). Możesz wykonać podobny lub skorzystać z przygotowanych pomocy dydaktycznych.

2. Wywoływanie funkcji wyszukiwania elementu w zbiorze uporządkowanym w programie głównym

Zgodnie z podaną specyfikacją, wynikiem działania programu realizującego algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia ma być pozycja szukanego elementu (indeks tablicy lub listy) lub komunikat, że elementu nie znaleziono. W programie umieścimy funkcję wprowadzania danych do tablicy (listy) i wyprowadzania ich na ekran oraz funkcję wyszukującą dany element. Następnie wywołamy je w programie głównym, dodając odpowiednie komunikaty.

Ćwiczenie 2. Piszemy program realizujący algorytm wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia w języku C++ lub Python

1. Napisz program realizujący algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia. Dla uproszczenia przyjmij, że w zbiorze jest pięć elementów.
   a. zdefiniuj funkcję *wprowadz_dane()* do wprowadzania elementów do tablicy (listy) i *wyprowadz_dane()* – do ich wyprowadzenia na ekran,
   b. zdefiniuj funkcję *znajdz_dana()* pokazaną na rysunku 1a lub 1b,
   c. wszystkie funkcje wywołaj w programie głównym,
   d. wyprowadzaj na ekran odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Wyszukiwanie_polowienie* i przetestuj działanie programu dla różnych danych (również dla różnej liczby danych). Pamiętaj o wprowadzaniu danych uporządkowanych.

Wskazówka: Możesz skorzystać ze zdefiniowanej w temacie 63. (C++) lub 75. (Python) funkcji wprowadzającej dane do tablicy (listy) i wyprowadzającej je na ekran.

Warto zapamiętać

- W algorytmie wyszukiwania elementu w zbiorze uporządkowanym elementy zbioru wprowadzamy do tablicy (C++) lub listy (Python), aby można było się do nich odwoływać.
- Aby zaprogramować algorytm wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia, możemy zdefiniować funkcję zwracającą wartość z jednym parametrem. Funkcja zwraca pozycję szukanego elementu lub -1, jeśli w przeszukiwanym zbiorze nie ma szukanej liczby.
- W funkcji szukającej danego elementu stosujemy instrukcję iteracyjną **`do ... while`** lub **`while`**, ponieważ nie znamy z góry liczby kroków iteracji. Przeszukiwanie zbioru prowadzimy aż znajdziemy szukany element lub gdy przeszukamy cały zbiór, nie znajdując go.

Pytania i polecenia

1. Jakie struktury danych powinno się zastosować, zapisując algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia w językach C++ lub Python? Uzasadnij odpowiedź.
2. Dlaczego elementy zbioru należy pamiętać w tablicy (C++) lub w liście (Python)?
3. Dlaczego w programie realizującym algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia zdefiniowano funkcję zwracającą wartość?
4. Jaki warunek musi być spełniony, aby funkcja *znajdz_dana()* zwróciła wartość indeksu szukanego elementu?
5. W jaki sposób można wywołać funkcję *znajdz_dana()* w programie głównym?

Zadania

1. Przeprowadź analizę funkcji *znajdz_dana()* dla zbioru liczb {14, 20, 33, 78, 90, 101}, gdzie szukany element to 90.
2. Zaprogramuj grę w zgadywanie liczby. Gra polega na odgadywaniu liczby z określonego zakresu (przyjmij zakres liczb ⟨1, 100⟩). Jedna osoba wybiera liczbę, a druga osoba ją odgaduje. Osoba zgadująca podaje liczbę, a druga osoba udziela odpowiedzi: „za mała" lub „za duża". Wartość szukanej liczby wprowadzamy z klawiatury po uruchomieniu programu. Zapisz program w pliku pod nazwą *Zgadywanie_liczby*.

Dla zainteresowanych

3. Zmodyfikuj program z zadania 2., tak aby komputer wybierał losowo szukaną liczbę z przedziału ⟨1, 100⟩. Zapisz program w pliku pod nazwą *Zgadywanie_liczby _los*.

Wskazówki:

C++ W języku C++ liczby losowe generuje funkcja `rand()`.

Na przykład `rand() % 100 + 1` – wygeneruje losowo liczbę naturalną z przedziału ⟨1, 100⟩.

Na początku programu musimy dołączyć bibliotekę, w której zdefiniowano funkcję `rand()`:

```
#include <stdlib.h>
```

Python W języku Python liczby losowe generuje funkcja `randint()`.

Na przykład `randint(1, 100)` – wygeneruje losowo liczbę naturalną z przedziału ⟨1, 100⟩.

Na początku programu musimy dołączyć bibliotekę, w której zdefiniowano funkcję `randint()`:

```
from random import randint
```

Temat 94.

Programowanie algorytmów zachłannych

1. Przykładowy program realizujący algorytm zachłanny w językach C++ i Python
2. Zapisywanie algorytmu wydawania reszty metodą zachłanną w językach C++ i Python

Warto powtórzyć

1. Na czym polega algorytm zachłanny?
2. W jaki sposób wydajemy resztę, stosując podejście zachłanne? Wyjaśnij na przykładzie.
3. W jaki sposób wykonać dzielenie całkowite (z obcięciem części ułamkowej) w wybranym języku programowania?
4. Czym jest dzielenie z resztą? Jakiego operatora należy użyć do obliczenia reszty z dzielenia dwóch liczb całkowitych w języku C++ lub Python?

1. Przykładowy program realizujący algorytm zachłanny w językach C++ i Python

Chcemy napisać program realizujący algorytm zachłanny przewozu palet z książkami. W jaki sposób napisać taki program w wybranym języku programowania (C++ lub Python)?

Opis problemu transportu książek przedstawiono w temacie 84. Jak pamiętamy, musimy przewieźć określoną liczbę książek. Książki pakuje się w paczki i umieszcza na paletach. W samochodzie można umieszczać pełne palety lub paczki bądź pojedyncze egzemplarze.

Zgodnie z cechą zachłanności, szukamy zachłannego, czyli najkorzystniejszego rozwiązania częściowego. W tym przypadku zaczniemy od sprawdzenia, ile pełnych palet można umieścić w samochodzie, następnie ile paczek i ile pojedynczych książek.



Przyjęto następujące oznaczenia dla zmiennych (wszystkie zmienne są typu całkowitego):

- *l_ksiazek_transport* – liczba książek do przewiezienia,
- *l_paczek_paleta* – liczba paczek mieszczących się na palecie,
- *l_ksiazek_paczka* – liczba książek mieszczących się w paczce,
- *reszta_ksiazek* – pozostała część książek do przewiezienia,
- *l_ksiazek_paleta* – liczba książek mieszczących się na palecie,
- *l_palet* – liczba palet.

We fragmentach programu (rys. 1a i 1b) pokazano wyznaczanie liczby pełnych palet oraz liczby pozostałych książek.

C++

```
l_ksiazek_paleta = l_paczek_paleta * l_ksiazek_paczka;
l_palet = l_ksiazek_transport / l_ksiazek_paleta;
cout << "Liczba palet: " << l_palet << endl;
reszta_ksiazek = l_ksiazek_transport % l_ksiazek_paleta;
```

Rys. 1a. Fragment programu – ćwiczenie 1. (C++)

Python

```
l_ksiazek_paleta = l_paczek_paleta * l_ksiazek_paczka
l_palet = l_ksiazek_transport // l_ksiazek_paleta
print("Liczba palet: ", l_palet)
reszta_ksiazek = l_ksiazek_transport % l_ksiazek_paleta
```

Rys. 1b. Fragment programu – ćwiczenie 1. (Python)

Ćwiczenie 1. Programujemy przykładowy algorytm zachłanny

1. W wybranym języku programowania napisz program obliczający, ile pełnych palet, paczek i pojedynczych książek umieścić w samochodzie, aby książki zapakować zgodnie z algorytmem zachłannym. Objaśnij działanie kolejnych instrukcji pokazanych na rysunkach 1a lub 1b i wykorzystaj je w programie.
2. Liczbę książek do przewiezienia, liczbę paczek mieszczących się na palecie i liczbę książek mieszczących się w paczce wprowadzaj z klawiatury.
3. Zapisz program w pliku pod nazwą *Transport*. Przetestuj program dla danych podanych w ćwiczeniu 1. z tematu 84.

2. Zapisywanie algorytmu wydawania reszty metodą zachłanną w językach C++ i Python

Chcemy napisać program realizujący zachłanny algorytm wydawania reszty. W jaki sposób napisać taki program w wybranym języku programowania (C++ lub Python)?

Program realizujący algorytm wydawania reszty napiszemy zgodnie z opisem podanym w temacie 84.

Listę dostępnych nominałów umieścimy w tablicy (C++) lub w liście (Python), które należy zdefiniować następująco:

C++

```
const int nominaly[] = {100, 50, 20, 10, 5, 2, 1};
const int l_nominalow = 7;
```

Python

```
nominaly = [100, 50, 20, 10, 5, 2, 1]
```

Elementy tablicy lub listy powinny być posortowane malejąco.

Program powinien wyświetlić dla obliczonej reszty wypłacone nominały, podobnie jak pokazano na rysunku 2.

```
Cena: 7, wplata: 100
Reszta do wyplaty: 93
Wyplacone nominaly:
1 x 50 zl
2 x 20 zl
1 x 2 zl
1 x 1 zl
```



Rys. 2. Przykład wyświetlania reszty w postaci wypłaconych nominałów

W zachłannym algorytmie wydawania reszty wykorzystujemy iterację. Szukanie nominału powtarzamy dla pomniejszonej wartości reszty wśród kolejnych nominałów, aż wypłacimy całą sumę – wykorzystamy instrukcję **`while`**.

Na rysunkach 3a i 3b pokazano definicję funkcji *znajdz_wypisz_reszte()* w językach C++ i Python. Funkcja ma za zadanie wypisać na ekranie liczbę poszczególnych nominałów, niezbędnych do wydania reszty.

Przyjęto następujące oznaczenia dla zmiennych:

- *aktualny_nominal* – aktualnie sprawdzany nominał, czyli indeks nominału w tablicy (liście),
- *reszt*a – reszta do wydania,
- *l_nominalow* – liczba wydawanych nominałów o wartości *nominaly*[*aktualny_nominal*].

Do obliczenia liczby banknotów należy zastosować:

- w języku C++ operator dzielenia /, którego działanie zależy od typu danych (w przypadku liczb całkowitych wynik również będzie całkowity),
- w języku Python – operator dzielenia całkowitego //.

C++

```cpp
void znajdz_wypisz_reszte(int reszta)
{
    if(reszta < 0)
    {
        cout << "Reszta ujemna!" << endl;
        return;
    }

    cout << "Wyplacone nominaly: " << endl;
    int aktualny_nominal = 0;

    while(reszta > 0 && aktualny_nominal < l_nominalow)
    {
        while(aktualny_nominal < l_nominalow && reszta < nominaly[aktualny_nominal])
            aktualny_nominal++;

        if(aktualny_nominal < l_nominalow && reszta >= nominaly[aktualny_nominal])
        {
            int l_nominalow = reszta / nominaly[aktualny_nominal];
            cout << l_nominalow << " x " << nominaly[aktualny_nominal] << " zl" << endl;
            reszta = reszta - l_nominalow * nominaly[aktualny_nominal];
        }
    }

    if(reszta > 0)
        cout << "Brak nominalow do wydania pozostalej kwoty: " << reszta << " zl" << endl;
}
```

Rys. 3a. Funkcja wydawania reszty metodą zachłanną (C++)

Python

```python
def znajdz_wypisz_reszte(reszta):
    if reszta < 0:
        print("Reszta ujemna")
        return

    aktualny_nominal = 0

    while reszta > 0 and aktualny_nominal < len(nominaly):
        while aktualny_nominal < len(nominaly) and reszta < nominaly[aktualny_nominal]:
            aktualny_nominal = aktualny_nominal + 1
        if aktualny_nominal < len(nominaly) and reszta >= nominaly[aktualny_nominal]:
            l_nominalow = reszta // nominaly[aktualny_nominal]
            print(nominaly[aktualny_nominal], "x", l_nominalow)
            reszta = reszta - l_nominalow * nominaly[aktualny_nominal]

    if reszta > 0:
        print("Brak nominałów do wydania pozostałej kwoty", reszta)
```

Rys. 3b. Funkcja wydawania reszty metodą zachłanną (Python)

Ćwiczenie 2. Analizujemy realizację algorytmu wydawania reszty metodą zachłanną z wykorzystaniem funkcji *znajdz_wypisz_reszte()*

1. Zapoznaj się z wybraną definicją funkcji (rys. 3a lub 3b) i, korzystając z niej, przeprowadź analizę realizacji algorytmu wydawania reszty metodą zachłanną dla reszty równej 97.
   Analizę funkcji wykonaj w tabeli (podobnej do tabeli 1. z tematu 84.), pokazując dodatkowo, jak zmieniają się wartości poszczególnych zmiennych (m.in. *reszta*, *l_banknotow*, *nominaly*[*aktualny _nominal*]).
2. Objaśnij znaczenie poszczególnych instrukcji, odpowiadając na pytania:
   a. Dlaczego w funkcji *znajdz_wypisz_reszte()* zastosowano instrukcję iteracyjną **`while`**?
   b. Jaki warunek musi być spełniony, aby nastąpiło wyjście z pętli zewnętrznej **`while`**, a jaki z pętli wewnętrznej **`while`**?
   c. Jakie instrukcje umieszczono w bloku instrukcji warunkowej **`if`**? Kiedy zostaną one wykonane?

Ćwiczenie 3. Piszemy program realizujący algorytm zachłanny wydawania reszty w języku C++ lub Python

1. W wybranym języku programowania napisz program realizujący algorytm zachłanny wydawania reszty. Przyjmij następujące nominały: `{100, 50, 20, 10, 5, 2, 1}`, które zapamiętaj w tablicy (liście), jak pokazano w tym temacie.
   a. Zdefiniuj funkcję niezwracającą wartości *wypisz_nominaly()*, wyprowadzającą na ekran dostępne nominały.
   b. Zdefiniuj funkcję *znajdz_wypisz_reszte()* z parametrem *reszta* pokazaną na rysunku 3a lub 3b.
   c. W programie głównym wywołaj funkcję *wypisz_nominaly()*, wprowadź cenę (*cena*) i wpłaconą kwotę (*wplata*), oblicz wartość reszty (*r*) i wywołaj funkcję *znajdz_wypisz_reszte()* z parametrem aktualnym *r*. Wyprowadzaj na ekran odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Wydawanie_reszty*, a następnie przetestuj program dla kilku różnych danych.

Warto zapamiętać

- W programie realizującym algorytm wydawania reszty metodą zachłanną, nominały zapamiętujemy w tablicy (C++) lub liście (Python), aby można było się do nich odwoływać.
- Aby zaprogramować algorytm wydawania reszty metodą zachłanną, definiujemy funkcję wypisującą na ekranie liczbę poszczególnych nominałów, niezbędnych do wydania reszty. W funkcji powtarzamy szukanie nominału wśród kolejnych nominałów dla pomniejszonej wartości reszty tak długo, aż wypłacimy całą sumę. Stosujemy instrukcję iteracyjną **`while`**, ponieważ nie znamy z góry liczby kroków iteracji.

Pytania i polecenia

1. Jakie struktury danych powinno się zastosować do pamiętania nominałów, zapisując algorytm wydawania reszty metodą zachłanną w językach C++ lub Python? Uzasadnij odpowiedź.
2. Dlaczego nominały powinny być uporządkowane malejąco?
3. Dlaczego w programie realizującym algorytm wydawania reszty metodą zachłanną zdefiniowano funkcję niezwracającą wartości? Jakie jest zadanie tej funkcji?

Zadania

1. Musisz przewieźć określoną liczbę palet ładunku, którego nie można rozdzielać. Dysponujesz pojazdami o trzech różnych ładownościach. W wybranym języku programowania napisz program obliczający, ile i jakich samochodów należy użyć, aby jak najlepiej wykorzystać przestrzeń. Liczbę palet do przewiezienia, liczbę samochodów i ich ładowność wprowadzaj z klawiatury. Zapisz program w pliku pod nazwą *Ladunek*. Przetestuj program dla danych podanych w zadaniu 2. z tematu 84.

Dla zainteresowanych

2. Sformułuj samodzielnie problem, do którego rozwiązania można zastosować podejście zachłanne. Napisz program rozwiązujący ten problem. Zdefiniuj odpowiednie funkcje. Program zapisz w pliku pod nazwą *Problem_zachlanny*.

Temat 95. Programowanie algorytmów rekurencyjnych i iteracyjnych

1. **Zapisywanie algorytmu obliczania silni w językach C++ i Python**
   1.1. Obliczanie silni liczby naturalnej metodą iteracyjną
   1.2. Obliczanie silni liczby naturalnej metodą rekurencyjną
2. **Zapisywanie algorytmu obliczania liczb Fibonacciego w językach C++ i Python**
   2.1. Obliczanie liczb Fibonacciego metodą rekurencyjną
   2.2. Obliczanie liczb Fibonacciego metodą iteracyjną

Warto powtórzyć

1. Kiedy mamy do czynienia z iteracją, a kiedy z rekurencją? Czym różni się rekurencja od iteracji?
2. Czym jest stos w informatyce?
3. Czym jest silnia liczby naturalnej?
4. Jak obliczamy silnię liczby naturalnej metodą iteracyjną, a jak – rekurencyjną?
5. Czym jest liczba Fibonaccego, a czym ciąg Fibonaccego?
6. Jak obliczamy liczby Fibonacciego metodą rekurencyjną, a jak – iteracyjną?

1. Zapisywanie algorytmu obliczania silni w językach C++ i Python

Chcemy zaprogramować algorytm obliczania silni liczby naturalnej metodą iteracyjną i rekurencyjną. W jaki sposób napisać takie programy w wybranym języku programowania (C++ lub Python)?

1.1. Obliczanie silni liczby naturalnej metodą iteracyjną

W iteracyjnej definicji silni występuje powtarzanie poleceń, mamy więc do czynienia z iteracją, w której określono z góry liczbę kroków. Dlatego w iteracyjnej realizacji algorytmu obliczania silni zastosujemy instrukcję `for`.

Uwaga

Instrukcję:

```
wynik = wynik * i;
```

można zapisać krócej:

```
wynik *= i;
```

C++

```
 7  int silnia_iter(int n)
 8  {
 9      int wynik = 1;
10
11      if(n > 1)
12          for(int i = 2; i <= n; i++)
13              wynik = wynik * i;
14      return wynik;
15  }
```

Rys. 1a. Iteracyjna definicja funkcji obliczającej silnię liczby naturalnej (C++)

Uwaga

Instrukcję:

```
wynik = wynik * i
```

można zapisać krócej:

```
wynik *= i
```

Python

```
def silnia_iter(n):
    wynik = 1

    if n > 1:
        for i in range(2, n + 1):
            wynik = wynik * i
    return wynik
```

Rys. 1b. Iteracyjna definicja funkcji obliczającej silnię liczby naturalnej (Python)

Ćwiczenie 1. Zapisujemy w postaci programu iteracyjną realizację algorytmu obliczającego silnię liczby naturalnej *n*

1. Napisz specyfikację zadania obliczania silni liczby naturalnej.
2. W wybranym języku programowania (C++ lub Python) napisz program iteracyjnej realizacji algorytmu obliczania silni liczby naturalnej. Wykorzystaj funkcję *silnia_iter()* z rysunku 1a lub 1b. Omów jej działanie – wyjaśnij znaczenie poszczególnych instrukcji.
3. Funkcję *silnia_iter()* wywołaj w programie głównym z parametrem aktualnym *liczba*.
4. Zapisz program w pliku pod nazwą *Silnia_iteracyjnie* i przetestuj program dla kilku różnych danych.

1.2. Obliczanie silni liczby naturalnej metodą rekurencyjną

Dana funkcja jest **rekurencyjna**, jeśli zawiera odwołanie do samej siebie. Aby zapisać rekurencyjną realizację wybranego algorytmu, musimy zapisać algorytm w postaci funkcji, która może wywołać samą siebie.

Definiujemy funkcję z jednym parametrem *n*. W definicji funkcji *silnia_rek()* następuje jej wywołanie wewnątrz niej samej z parametrem *n* – 1.

C++

```
 6  int silnia_rek(int n)
 7  {
 8      if(n == 0)
 9          return 1;
10      else
11          return n*silnia_rek(n - 1);
12  }
```

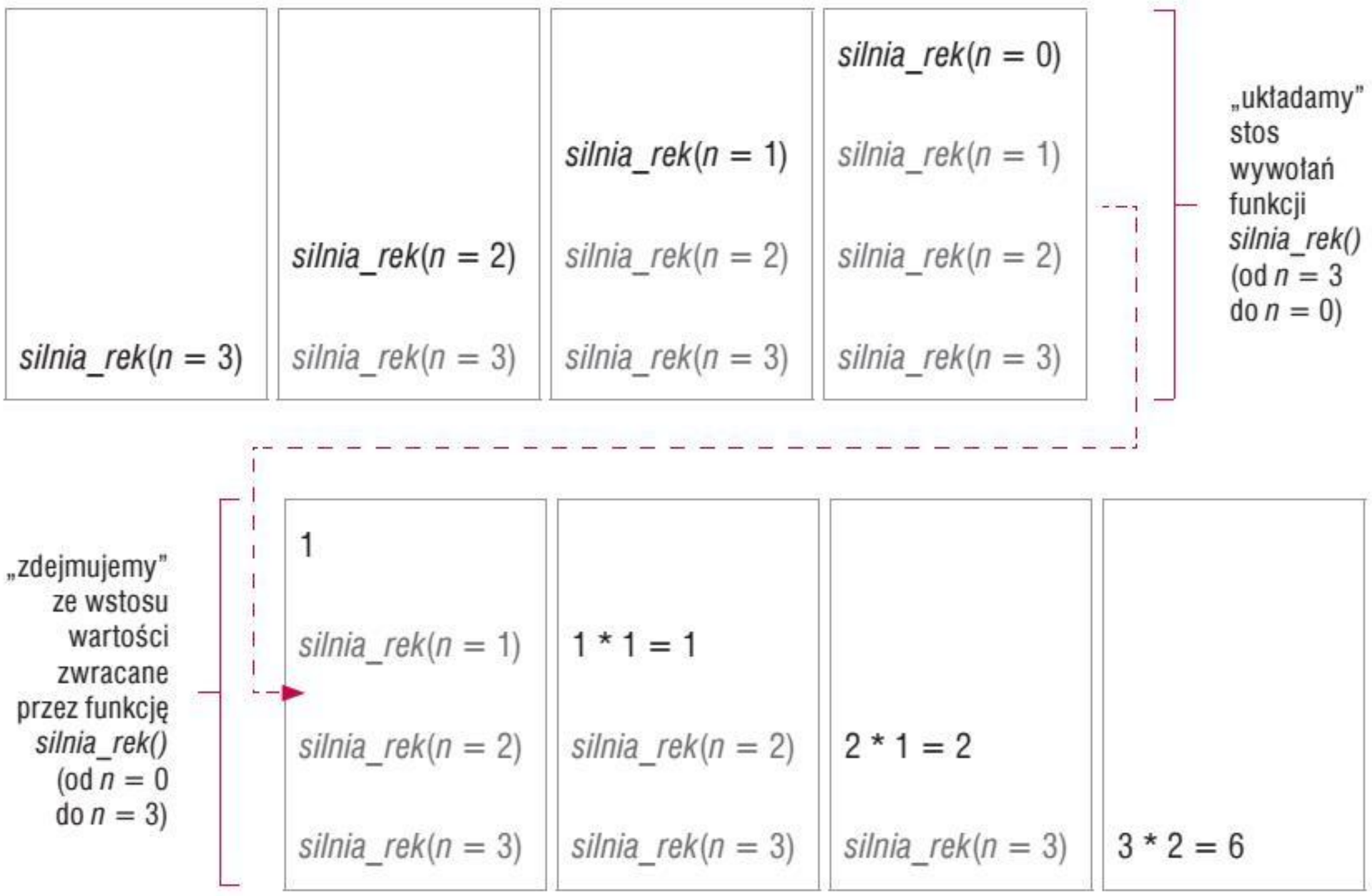
Rys. 2a. Rekurencyjna definicja funkcji obliczającej silnię liczby naturalnej (C++)

Python

```
def silnia_rek(n):
    if n == 0:
        return 1
    else:
        return n*silnia_rek(n - 1)
```

Rys. 2b. Rekurencyjna definicja funkcji obliczającej silnię liczby naturalnej (Python)

Na rysunku 3. zobrazowano schematycznie, w jaki sposób zmieniają się wartości funkcji *silnia_rek()* podczas tworzenia tzw. **stosu wywołań rekurencyjnych** funkcji *silnia_rek()*, a potem podczas ich „zdejmowania" (począwszy od ostatnio wywołanej funkcji).



Rys. 3. Schemat wywołań rekurencyjnych funkcji *silnia_rek()* dla $n = 3$

Ćwiczenie 2. Zapisujemy w postaci programu rekurencyjną realizację algorytmu obliczającego silnię liczby naturalnej *n*

1. Przeanalizuj schemat wywołań rekurencyjnych funkcji *silnia_rek()*. Porównaj go z rysunkiem 2. z tematu 85.
2. W wybranym języku programowania napisz program rekurencyjnej realizacji algorytmu obliczania silni liczby naturalnej. Wykorzystaj funkcję *silnia_rek()* z rysunku 2a lub 2b. Omów jej działanie – wyjaśnij zapis `n*silnia_rek(n - 1)`.
3. Funkcję *silnia_rek()* wywołaj w programie głównym z parametrem aktualnym *liczba*, którego wartość wprowadzaj z klawiatury.
4. Zapisz program w pliku pod nazwą *Silnia_rekurencyjnie* i przetestuj go dla kilku różnych danych.

Aby obliczyć i wyświetlić na ekranie dwanaście kolejnych wartości silni, można wywołanie funkcji *silnia_rek()* umieścić w pętli z parametrem aktualnym *i*, gdzie *i* jest zmienną globalną.

C++

```
for(i = 0; i <= 12; i++)
    cout << i << "!=" << silnia_rek(i) << endl;
```

Python

```
for i in range(13):
    print(i, '!=', silnia_rek(i))
```

Ćwiczenie 3. Modyfikujemy program obliczający silnię

1. Zmodyfikuj program zapisany w ćwiczeniu 2., aby obliczał i wyświetlał w jednej kolumnie wartości silni dla liczb od 1 do 12.
2. Zapisz program w pliku pod nazwą *Silnia_rekurencyjnie_12*.

2. Zapisywanie algorytmu obliczania liczb Fibonacciego w językach C++ i Python

Chcemy zaprogramować algorytm obliczania liczb Fibonacciego metodą iteracyjną i rekurencyjną. W jaki sposób napisać takie programy w wybranym języku programowania (C++ lub Python)?

2.1. Obliczanie liczb Fibonacciego metodą rekurencyjną

Definicja ciągu Fibonacciego jest rekurencyjna, gdyż odwołuje się do samej siebie.

Funkcja realizująca rekurencyjny algorytm obliczania liczb Fibonacciego $n \geq 3$ powinna być wywoływana dwukrotnie wewnątrz siebie samej: raz z parametrem $n - 1$, drugi raz z parametrem $n - 2$. Na rysunkach 4a i 4b pokazano przykładowe definicje funkcji *fib_rek()* w wybranych językach programowania.



C++

```
 6  long fib_rek(int n)
 7  {
 8      if(n < 3)
 9          return 1;
10      else
11          return fib_rek(n - 2) + fib_rek(n - 1);
12  }
```

Rys. 4a. Rekurencyjna definicja funkcji obliczającej liczby Fibonacciego (C++)

Python

```
def fib_rek(n):
    if n < 3:
        return 1
    else:
        return fib_rek(n - 2) + fib_rek(n - 1)
```

Rys. 4b. Rekurencyjna definicja funkcji obliczającej liczby Fibonacciego (Python)

Ćwiczenie 4. Zapisujemy w postaci programu rekurencyjną realizację algorytmu obliczającego liczby Fibonacciego

1. Napisz specyfikację zadania obliczania liczb Fibonacciego.
2. Korzystając z wybranej definicji funkcji (rys. 4a lub 4b), napisz program w wersji rekurencyjnej, obliczający *k*-tą liczbę Fibonacciego. Omów działanie funkcji – wyjaśnij zapis `fib_rek(n - 2) + fib_rek(n - 1)`.
3. Funkcję *fib_rek()* wywołaj w programie głównym z parametrem aktualnym *k*.
4. Zapisz program w pliku pod nazwą *Fibonacci_rekurencyjnie* i przetestuj dla różnych danych.

2.2. Obliczanie liczb Fibonacciego metodą iteracyjną

Do obliczenia poszczególnych liczb Fibonacciego użyliśmy algorytmu rekurencyjnego, ale możemy również zastosować algorytm iteracyjny, co pokazaliśmy w temacie 85.

W programie zdefiniujemy funkcję obliczającą iteracyjnie liczby Fibonacciego *fib_iter()* – z jednym parametrem formalnym *n*. Funkcja będzie zwracać do programu głównego wartość *n*-tej liczby Fibonacciego (rys. 5a i 5b). W funkcji zastosujemy instrukcję iteracyjną **for**. Liczba kroków iteracji jest z góry określona.

W algorytmie iteracyjnym wartości dwóch poprzednich liczb Fibonacciego musimy pamiętać w zmiennych (tu są to zmienne: *poprzedni1* i *poprzedni2*).

C++

```
 6  int fib_iter(int n)
 7  {
 8      int wynik;
 9
10      if  (n < 3)
11          wynik = 1;
12      else
13      {
14          int poprzedni1 = 1;
15          int poprzedni2 = 1;
16          for(int i = 3; i <= n; ++i)
17          {
18              wynik = poprzedni1 + poprzedni2;
19              poprzedni2 = poprzedni1;
20              poprzedni1 = wynik;
21          }
22      }
23      return wynik;
24  }
```

dwie początkowe liczby są równe jeden

przypisanie zmiennej *wynik* sumy dwóch poprzednich liczb

przypisanie zmiennej *poprzedni2* wartości zmiennej *poprzedni1*

przypisanie zmiennej *poprzedni1* – wartości zmiennej *wynik*

Rys. 5a. Iteracyjna definicja funkcji obliczającej liczby Fibonacciego (C++)

Python

```
def fib_iter(n):
    if n < 3:
        wynik = 1
    else:
        poprzedni1 = 1
        poprzedni2 = 1
        for i in range(3, n + 1):
            wynik = poprzedni1 + poprzedni2
            poprzedni2 = poprzedni1
            poprzedni1 = wynik
    return wynik
```

dwie początkowe liczby są równe jeden

przypisanie zmiennej *wynik* sumy dwóch poprzednich liczb

przypisanie zmiennej *poprzedni2* wartości zmiennej *poprzedni1*

przypisanie zmiennej *poprzedni1* – wartości zmiennej *wynik*

Rys. 5b. Iteracyjna definicja funkcji obliczającej liczby Fibonacciego (Python)

Ćwiczenie 5. Zapisujemy w postaci programu iteracyjną realizację algorytmu obliczającego liczby Fibonacciego

1. W wybranym języku programowania napisz program realizujący iteracyjny algorytm obliczania liczb Fibonacciego. Skorzystaj z wybranej definicji funkcji (rys. 5a lub 5b). Objaśnij działanie funkcji.
2. Funkcję wywołaj *k* razy w programie głównym z parametrem aktualnym *i* oznaczającym numer kolejnej liczby Fibonacciego (*i* powinna przyjmować wartości od 1 do *k*).
3. Zapisz program w pliku pod nazwą *Fibonacci_iteracyjnie* i przetestuj dla różnych danych.

Warto zapamiętać

- W iteracyjnych realizacjach algorytmów obliczania silni liczby naturalnej oraz liczb Fibonacciego powtarzamy w pętli obliczenia dla z góry określonej liczby danych – stosujemy instrukcję iteracyjną **`for`**.
- W rekurencyjnej realizacji algorytmu obliczania silni liczby naturalnej *n* w definicji funkcji wywołujemy ją wewnątrz niej samej z parametrem *n* – 1.
- W rekurencyjnej realizacji obliczania *n*-tej liczby Fibonacciego w definicji funkcji wywołujemy ją dwukrotnie wewnątrz niej samej: raz z parametrem *n* – 1, drugi raz z parametrem *n* – 2.

Pytania i polecenia

1. W jaki sposób programujemy algorytm obliczania silni liczby naturalnej metodą iteracyjną?
2. Dlaczego w funkcjach realizujących algorytmy obliczania silni liczby naturalnej i liczb Fibonacciego metodą rekurencyjną nie ma potrzeby stosowania instrukcji iteracyjnej? Uzasadnij odpowiedź.
3. Dlaczego w algorytmie iteracyjnym obliczania liczb Fibonacciego wartości dwóch poprzednich liczb musimy pamiętać w oddzielnych zmiennych?
4. Wyjaśnij, dlaczego przy liczbach Fibonacciego umieszczono zdjęcie królików.

Zadania

1. Stosując rekurencyjną definicję obliczania silni, napisz program obliczający wartość symbolu Newtona: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, gdzie *n* i *k* – liczby naturalne.
   Zapisz program w pliku pod nazwą *Symbol_Newtona*.
2. Wykonaj, podobnie jak pokazano na rysunku 3., schemat rekurencyjnego wyznaczania sześciu początkowych liczb Fibonacciego.
3. Zmodyfikuj program zapisany w ćwiczeniu 4., aby obliczał i wyświetlał w jednej kolumnie czterdzieści początkowych liczb Fibonacciego. Zapisz program w pliku pod nazwą *Fibonacci_rekurencyjnie_40*.

Dla zainteresowanych

4. Korzystając z rozwiązania zadania 1., napisz program obliczający, na ile sposobów można wybrać *k*-osobową reprezentację spośród *s* uczniów jednej klasy (*k* i *s* wprowadzane z klawiatury). Umieść w programie odpowiednie opisy, wyświetlające się przy wprowadzaniu danych i wyprowadzaniu wyniku. Zapisz program w pliku pod nazwą *Reprezentacja_uczniowie*.
5. Wykorzystując program zapisany w ćwiczeniu 1., oblicz wartość 16! i 17!. Porównaj otrzymane wartości z wartościami wyznaczonymi inną metodą (np. za pomocą kalkulatora naukowego lub arkusza kalkulacyjnego). Co zauważasz?

Temat 96.

Programowanie zamiany reprezentacji liczb między pozycyjnymi systemami liczbowymi

1. **Programowanie algorytmu obliczania wartości dziesiętnej liczby dwójkowej**
2. **Programowanie algorytmu wyznaczania rozwinięcia dwójkowego liczby dziesiętnej**

Warto powtórzyć

1. W jaki sposób wyznaczamy rozwinięcie dwójkowe liczby dziesiętnej? Pokaż na przykładzie.
2. Jak oblicza się wartość dziesiętną liczby binarnej? Wyjaśnij na przykładzie.
3. Jak możemy odwoływać się do pojedynczego znaku łańcucha w języku C++ lub Python?
4. Jaka funkcja zwraca długość łańcucha w języku C++ lub Python?
5. W jaki sposób możemy połączyć dwa łańcuchy w języku C++ lub Python?

1. Programowanie algorytmu obliczania wartości dziesiętnej liczby dwójkowej

Chcemy zaprogramować algorytm zamiany liczby dwójkowej na dziesiętną. W jaki sposób napisać taki program w wybranym języku programowania (C++ lub Python)?

W temacie 50. omówiliśmy system dwójkowy oraz algorytmy zamiany liczb pomiędzy systemami dziesiętnym i dwójkowym. W tym temacie zaprogramujemy te algorytmy w językach C++ i Python.

Aby obliczyć wartość dziesiętną liczby zapisanej w systemie dwójkowym, dokonujemy rozwinięcia liczby według wzoru podanego w temacie 50., przyjmując jako podstawę systemu liczbę 2.

Na przykład wartość dziesiętną liczby 1001_2 możemy obliczyć następująco:

3 2 1 0

$1\,0\,0\,1_2 = 2^3 + 2^0 = 8 + 1 = 9.$

Pominęliśmy zera w liczbie binarnej, ponieważ $0 \cdot 2^n = 0$. Wystarczy zsumować tylko te potęgi dwójki, które znajdują się przy cyfrach 1. Takie rozwiązanie przyjęto również w funkcji *dwojkowy_na_10()* – rys. 1a i 1b.

Pisząc program, należy zastanowić się, jakiego typu powinna być zmienna, w której będziemy pamiętać liczbę dwójkową (binarną). Ponieważ musimy odwołać się do każdej cyfry liczby dwójkowej, zastosujemy typ tekstowy.

Na rysunkach 1a i 1b pokazano funkcje zamieniające liczbę dwójkową na dziesiętną. Korzystamy ze wzoru na wartość dziesiętną liczby dwójkowej (podanego w temacie 50.). W algorytmie mamy więc do czynienia z iteracją, dlatego w funkcji zastosujemy instrukcję pętli **`for`**, w której sprawdzamy, czy kolejna cyfra binarna jest równa '1' – jeśli tak, to sumujemy potęgi dwójki.

Aby uniknąć operacji potęgowania (dla obliczenia 2^n), kolejne potęgi dwójki otrzymujemy, podwajając wcześniej obliczoną potęgę (`potega = potega * 2`, co możemy zapisać też krócej `potega *= 2`).

C++

```
 6  int dwojkowy_na_10(string bin)
 7  {
 8      int dzies = 0;
 9      int potega = 1;
10      for(int i = bin.length() - 1; i >= 0; i--)
11      {
12          if (bin[i] == '1')
13              dzies = dzies + potega;
14          potega = potega * 2;
15      }
16      return dzies;
17  }
```

Rys. 1a. Funkcja obliczająca wartość dziesiętną liczby dwójkowej (C++)

Python

```
def dwojkowy_na_10(bin):
    dzies = 0
    potega = 1
    for znak in reversed(bin):
        if znak == '1':
            dzies = dzies + potega
        potega = potega * 2
    return dzies
```

Rys. 1b. Funkcja obliczająca wartość dziesiętną liczby dwójkowej (Python)

! Uwaga

W języku Python do iteracji po napisie w odwrotnej kolejności zastosowano funkcję `reversed()`.

Ćwiczenie 1. Analizujemy realizację algorytmu zamiany liczby dwójkowej na dziesiętną z wykorzystaniem funkcji *dwojkowy _na_10()*

1. Napisz specyfikację zadania zamiany liczby dwójkowej na dziesiętną.
2. Zapoznaj się z wybraną definicją funkcji (rys. 1a lub 1b) i objaśnij znaczenie poszczególnych instrukcji.
3. Korzystając z wybranej definicji funkcji (rys. 1a lub 1b), przeprowadź analizę realizacji algorytmu dla liczby 1001_2, pokazując dodatkowo, jak zmieniają się wartości zmiennych *i*, *dzies*, *bin*[*i*] oraz *potega*.

Ćwiczenie 2. Piszemy program realizujący algorytm zamiany liczby dwójkowej na dziesiętną w językach C++ lub Python

1. W wybranym języku programowania utwórz program obliczający wartość dziesiętną liczby zapisanej w systemie dwójkowym (binarnym).
   a. Zdefiniuj funkcję zwracającą wartość *dwojkowy_na_10()* z parametrem *bin*.
   b. W programie głównym wywołaj funkcję *dwojkowy_na_10()* z parametrem aktualnym *b* i wyprowadzaj na ekran odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Dwa_na_10*.
3. Przetestuj program dla kilku różnych danych.

2. Programowanie algorytmu wyznaczania rozwinięcia dwójkowego liczby dziesiętnej

Chcemy zaprogramować algorytm wyznaczania rozwinięcia dwójkowego liczby dziesiętnej. W jaki sposób napisać taki program w wybranym języku programowania (C++ lub Python)?

Aby wyznaczyć postać dwójkową liczby dziesiętnej, powtarzamy operację dzielenia z resztą tej liczby przez dwa i zapisujemy wszystkie reszty z dzielenia, aż do otrzymania ilorazu równego zero. Rozwinięcie dwójkowe liczby to zapisane kolejno (począwszy od ostatniej) reszty z dzieleń.

Algorytm zamiany liczby dziesiętnej na dwójkową omówiliśmy w temacie 50. Na przykład w tabeli 1. pokazujemy zamianę liczby 13 na liczbę dwójkową.

Liczba	Reszta	Obliczenia
13	1	$13 = 2 \cdot 6 + 1$
6	0	$6 = 2 \cdot 3 + 0$
3	1	$3 = 2 \cdot 1 + 1$
1	1	$1 = 2 \cdot 0 + 1$
0		Iloraz równy 0 – koniec
		Wynik: $13 = 1101_2$

Tabela 1. Przykład wyznaczania rozwinięcia dwójkowego liczby dziesiętnej

Na rysunkach 2a i 2b pokazano definicje funkcji wyznaczającej rozwinięcie dwójkowe liczby dziesiętnej. W algorytmie zamiany liczby dziesiętnej na dwójkową występuje powtarzanie poleceń, dlatego w funkcjach użyto instrukcji iteracyjnej. W pętli obliczamy reszty z dzieleń liczby dziesiętnej przez 2 i zapisujemy je (w zmiennej *wynik*) tak, aby utworzyły rozwinięcie dwójkowe liczby (kolejna cyfra jest dodawana „z przodu").

C++ `wynik = (char)(dzies % 2 + '0') + wynik;`

W języku C++ zapis: `(char)(dziesietna % 2 + '0')` oznacza zamianę wyrażenia występującego po `(char)` na typ zapisany w nawiasie, czyli w tym przypadku `char`.

Python `wynik = chr(dzies % 2 + ord('0')) + wynik`

W języku Python funkcja `ord()` zamienia znak na kod Unicode, a `chr()` zamienia kod Unicode na odpowiadający mu znak.

C++

```
 6  string dziesietny_na_2(int dzies)
 7  {
 8      string wynik = "";
 9      do
10      {
11          wynik = (char)(dzies % 2 + '0') + wynik;
12          dzies = dzies / 2;
13      } while(dzies > 0);
14      return wynik;
15  }
```

Rys. 2a. Funkcja wyznaczająca rozwinięcie dwójkowe liczby dziesiętnej (C++)

Python

```
def dziesietny_na_2(dzies):
    wynik = ''
    if dzies == 0:
        wynik = '0'
    else:
        while dzies > 0:
            wynik = chr(dzies % 2 + ord('0')) + wynik
            dzies = dzies // 2
        return wynik
```

Rys. 2b. Funkcja wyznaczająca rozwinięcie dwójkowe liczby dziesiętnej (Python)

Ćwiczenie 3. Analizujemy realizację algorytmu wyznaczania rozwinięcia dwójkowego liczby dziesiętnej z wykorzystaniem funkcji *dziesietny_na_2()*

1. Napisz specyfikację zadania wyznaczania rozwinięcia dwójkowego liczby dziesiętnej.
2. Zapoznaj się z wybraną definicją funkcji (rys. 2a lub 2b) i, korzystając z niej, przeprowadź analizę realizacji algorytmu dla liczby 13 pokazując dodatkowo, jak zmieniają się wartości zmiennych *dzies* i *wynik*.
3. Objaśnij znaczenie poszczególnych instrukcji, odpowiadając na pytania:
   a. Którą cyfrę liczby binarnej wyznaczamy w pierwszym przebiegu pętli (rys. 3.)?
   b. Dlaczego w funkcji nie zastosowano instrukcji iteracyjnej **`for`**?

```
Podaj liczbe dziesietna: 13
wynik = 1
wynik = 01
wynik = 101
wynik = 1101
13 = 1101(2)
```

Rys. 3. Utworzenie rozwinięcia dwójkowego liczby 13 – na ekranie wyświetlono kolejne wartości zmiennej *wynik*

Ćwiczenie 4. Piszemy program wyznaczający rozwinięcie dwójkowe liczby dziesiętnej w językach C++ lub Python

1. W wybranym języku programowania utwórz program obliczający wartość dziesiętną liczby zapisanej w systemie dwójkowym (binarnym):
   a. Zdefiniuj logiczną funkcję zwracającą wartość *dziesietny_na_2()* z parametrem *dzies*.
   b. W programie głównym wywołaj funkcję *dziesietny_na_2()* z parametrem aktualnym *d* i wyprowadzaj na ekran odpowiednie komunikaty.
2. Zapisz program w pliku pod nazwą *Dziesiec_na_2*.
3. Przetestuj program dla kilku różnych danych.

Warto zapamiętać

- Aby zaprogramować algorytm obliczania wartości dziesiętnej liczby dwójkowej, definiujemy funkcję, w której w pętli `for` sumujemy tylko te potęgi dwójki, które znajdują się przy cyfrach 1.
- Aby zaprogramować algorytm wyznaczania rozwinięcia dwójkowego liczby dziesiętnej, definiujemy funkcję, w której w pętli obliczamy reszty z dzielenia liczby dziesiętnej przez 2 i zapisujemy je (w zmiennej *wynik*) tak, aby utworzyły rozwinięcie dwójkowe liczby (kolejna cyfra jest dodawana „z przodu").

Pytania i polecenia

1. W jaki sposób zapisujemy algorytm obliczania wartości dziesiętnej liczby dwójkowej w wybranym języku programowania?
2. Wyjaśnij sposób realizacji algorytmu wyznaczania rozwinięcia dwójkowego liczby dziesiętnej w wybranym języku programowania.
3. Jaki typ danych należy zastosować do zmiennej reprezentującej liczbę binarną w języku C++ lub Python? Uzasadnij odpowiedź.

Zadania

1. Zmodyfikuj program zapisany w ćwiczeniu 2., aby można było obliczyć wartości dziesiętne siedmiu liczb dwójkowych. Zapisz program w pliku pod nazwą *Dwa_na_10_7*.
2. Zmodyfikuj program zapisany w ćwiczeniu 4., aby można było wyznaczyć rozwinięcia dwójkowe dwunastu liczb dziesiętnych. Zapisz program w pliku pod nazwą *Dziesiec_na_2_12*.

Dla zainteresowanych

3. Napisz program realizujący rekurencyjny algorytm zamiany liczby dziesiętnej na postać binarną. Zapisz program w pliku pod nazwą *Dziesiec_na_2_rek*.

4. Napisz w wybranym języku programowania program umożliwiający zamianę liczb z systemu szesnastkowego na dziesiętny i odwrotnie. Zdefiniuj funkcje *szesnastkowy_na_10()* i *dziesietny_na_16()*. Zadbaj o właściwą postać parametrów i wyników funkcji. Zapisz program w pliku pod nazwą *Dziesiec_na_16_i_odwrotnie*.
Wskazówka:

C++ W języku C++ wartości typu **`char`** mają wartości liczbowe odpowiadające kodowi ASCII znaku. Przyjmując, że *c* jest znakiem cyfry (np. szesnastkowej), wartość tej cyfry reprezentują wyrażenia:

`c - '0'`	dla cyfr 0...9,
`c - 'A' + 10`	dla „cyfr" A...Z,
`c - 'a' + 10`	dla „cyfr" a...z.

Python W języku Python znaki nie mają bezpośredniej wartości liczbowej i aby jej użyć wykorzystujemy funkcję `ord()`. Przyjmując, że *c* jest znakiem cyfry (np. szesnastkowej), wartość tej cyfry reprezentują wyrażenia:

`ord(c) - ord('0')`	dla cyfr 0...9,
`ord(c) - ord('A') + 10`	dla „cyfr" A...Z,
`ord(c) - ord('a') + 10`	dla „cyfr" a...z.

Zamieniając znak na odpowiadającą mu wartość, należy najpierw sprawdzić, do którego z podanych wyżej przedziałów należy cyfra, i użyć odpowiedniego wyrażenia. W tym celu warto utworzyć osobną funkcję.

Temat 97. Programowanie wybranych algorytmów na liczbach

1. **Zapisywanie algorytmu badania pierwszości liczby w językach C++ i Python**
2. **Programowanie algorytmu Euklidesa**
   - 2.1. Zapisywanie algorytmu Euklidesa w wersji z odejmowaniem w językach C++ i Python
   - 2.2. Zapisywanie algorytmu Euklidesa w wersji z dzieleniem w językach C++ i Python

Warto powtórzyć

1. Jak sprawdzić, czy dana liczba jest pierwsza? Pokaż na przykładzie.
2. Czym jest NWD?
3. W jaki sposób obliczamy NWD, stosując algorytm Euklidesa?
4. Czym różni się wersja algorytmu Euklidesa z odejmowaniem od wersji z dzieleniem?
5. Jakiego operatora należy użyć do obliczenia reszty z dzielenia dwóch liczb całkowitych w języku C++ lub Python?

1. Zapisywanie algorytmu badania pierwszości liczby w językach C++ i Python

Chcemy zaprogramować algorytm badania pierwszości liczby. W jaki sposób napisać taki program w wybranym języku programowania (C++ lub Python)?

Program realizujący algorytm badania pierwszości liczby napiszemy zgodnie z opisem podanym w temacie 86. Jeśli liczba n nie ma dzielników w zbiorze $\langle 2; \sqrt{n} \rangle$, to jest liczbą pierwszą.

Przykład 1. Określenie zadania i jego specyfikacji

Zadanie: Napisz program w wybranym języku programowania sprawdzający, czy liczba naturalna n (większa od 1) jest liczbą pierwszą.

Dane: liczba naturalna n większa od 1.

Wynik: komunikat „Podana liczba jest liczbą pierwszą” lub „Podana liczba nie jest liczbą pierwszą”.

W programie zdefiniujemy funkcję *jest_pierwsza()* z jednym parametrem formalnym *n* sprawdzającą, czy liczba *n* jest liczbą pierwszą. Dla liczb pierwszych funkcja zwraca wartość logiczną PRAWDA, w przeciwnym przypadku – wartość logiczną FAŁSZ.

W funkcji *jest_pierwsza()* zastosowano funkcje standardowe :

- `floor()` – zaokrągla liczbę z częścią ułamkową w dół (obcina część ułamkową), np. wywołanie `floor(18.7777)` zwróci 18,
- `sqrt()` – oblicza pierwiastek drugiego stopnia, np. `sqrt(9)` zwróci liczbę 3.

W funkcji obliczamy resztę z dzielenia, dlatego stosujemy operator `%`.

C++ W języku C++ w instrukcji `m = floor(sqrt(n))` stosujemy operator (**`int`**) powodujący zamianę wartości rzeczywistej na całkowitą.

Python W języku Python interpreter określa typ zmiennej na podstawie rodzaju danych, jaki jej przypiszemy, dlatego w instrukcji `m = floor(sqrt(n))` po przypisaniu zmiennej *m* części całkowitej pierwiastka kwadratowego z *n*, zmienna *m* będzie typu całkowitego.

C++

```
1  #include <iostream>
2  #include <cmath>
3
4  using namespace std;
5
6  bool jest_pierwsza(int n)
7  {
8      int m, i;
9
10     m = (int) floor(sqrt(n));
11     for(i = 2; i <= m; i++)
12         if(n % i == 0)
13             return false;
14     return true;
15 }
```

dołączenie biblioteki funkcji matematycznych

przypisanie zmiennej *m* części całkowitej pierwiastka kwadratowego z *n*

sprawdzenie, czy *i* dzieli *n*

Rys. 1a. Definicja funkcji sprawdzającej, czy liczba jest liczbą pierwszą (C++)

Python

```
from math import floor, sqrt

def jest_pierwsza(n):
    m = floor(sqrt(n))
    for i in range (2, m + 1):
        if n % i == 0:
            return False
    return True
```

dołączenie biblioteki funkcji matematycznych

przypisanie zmiennej *m* części całkowitej pierwiastka kwadratowego z *n*

sprawdzenie, czy *i* dzieli *n*

Rys. 1b. Definicja funkcji sprawdzającej, czy liczba jest liczbą pierwszą (Python)

Aby funkcję *jest_pierwsza()* wywołać w programie głównym, należy wywołanie funkcji umieścić jako warunek w instrukcji **`if`**, ponieważ funkcja zwraca wartość logiczną.

Ćwiczenie 1. Zapisujemy algorytm badania pierwszości liczby w języku programowania

1. Zapoznaj się z wybraną definicją funkcji (rys. 1a lub 1b) i, korzystając z niej, przeprowadź analizę realizacji algorytmu badania pierwszości liczby dla liczb 11 i 16. Objaśnij działanie funkcji na tych przykładach.
2. W wybranym języku programowania napisz program realizujący algorytm badania pierwszości liczby:
   a. Zdefiniuj logiczną funkcję zwracającą wartość *jest_pierwsza()* z parametrem *n*.
   b. W programie głównym wywołaj funkcję *jest_pierwsza()* z parametrem aktualnym *liczba* i wyprowadzaj na ekran odpowiednie komunikaty.
3. Zapisz program w pliku pod nazwą *Pierwsza* i przetestuj dla kilku różnych danych.

2. Programowanie algorytmu Euklidesa

Chcemy napisać funkcje realizujące algorytm Euklidesa znajdowania największego wspólnego dzielnika (NWD), które przydadzą się w obliczeniach na ułamkach zwykłych. W jaki sposób zdefiniować takie funkcje w językach programowania C++ i Python?

Program realizujący algorytm Euklidesa napiszemy zgodnie z opisem podanym w temacie 86. Przedstawimy jego realizację w językach programowania C++ i Python w dwóch wersjach (z odejmowaniem i z dzieleniem). Program ma obliczać i wyświetlać na ekranie największy wspólny dzielnik dwóch liczb naturalnych.

Przykład 2. Określenie zadania i jego specyfikacji

Zadanie: Znajdź największy wspólny dzielnik dwóch niezerowych liczb naturalnych.

Dane: Liczby naturalne: a, b ($a \neq 0$, $b \neq 0$).

Wynik: Wartość największego wspólnego dzielnika liczb a i b: *nwd*.

Aby zapisać algorytm Euklidesa w wybranym języku programowania (niezależnie od wersji), zdefiniujemy funkcję zwracającą wartość *nwd()* z dwoma parametrami a i b.

2.1. Zapisywanie algorytmu Euklidesa w wersji z odejmowaniem w językach C++ i Python

W algorytmie Euklidesa w wersji z odejmowaniem powtarzamy operacje odejmowania liczby mniejszej od większej i zastępowania większej liczby otrzymaną różnicą, dopóki większa liczba i otrzymana różnica nie będą równe. Dlatego w funkcji *nwd()* wykorzystamy instrukcję iteracyjną `while`.

Wewnątrz pętli **while** zastosujemy dodatkowo instrukcję warunkową **if**, w której będziemy porównywać liczby *a* i *b*. Jeśli $a > b$, to zmiennej *a* przypisujemy wartość wyrażenia $a - b$; w przeciwnym przypadku zmiennej *b* przypisujemy wartość wyrażenia $b - a$.

Jeśli *a* jest równe *b*, następuje wyjście z pętli **while** i funkcja zwraca wartość NWD równą *a*.

C++

```
int nwd(int a, int b)
{
    while(a != b)
    {
        if(a > b)
            a = a - b;
        else
            b = b - a;
    }
    return a;
}
```

Rys. 2a. Definicja funkcji realizującej algorytm Euklidesa w wersji z odejmowaniem (C++)

Python

```
def nwd(a, b):
    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

Rys. 2b. Definicja funkcji realizującej algorytm Euklidesa w wersji z odejmowaniem (Python)

Ćwiczenie 2. Zapisujemy w języku C++ lub Python algorytm Euklidesa w wersji z odejmowaniem

1. Zapoznaj się z wybraną definicją funkcji (rys. 2a lub 2b) i, korzystając z niej, przeprowadź analizę realizacji algorytmu Euklidesa w wersji z odejmowaniem dla liczb 24 i 8. Objaśnij działanie funkcji na tym przykładzie.
2. W wybranym języku programowania napisz program realizujący algorytm Euklidesa w wersji z odejmowaniem:
   a. Zdefiniuj funkcję zwracającą wartość *nwd()* z dwoma parametrami *a* i *b*.
   b. W programie głównym wywołaj funkcję *nwd()* z parametrami aktualnymi *liczba1* i *liczba2* wprowadzanymi z klawiatury; na ekran wyprowadzaj też odpowiednie komunikaty.
3. Zapisz program w pliku pod nazwą *Euklides_odejmowanie* i przetestuj dla kilku różnych danych.

2.2. Zapisywanie algorytmu Euklidesa w wersji z dzieleniem w językach C++ i Python

W algorytmie Euklidesa w wersji z dzieleniem powtarzamy dzielenie liczby *a* przez liczbę *b*, dopóki liczba *b* jest różna od zera, dlatego w funkcji *nwd()* wykorzystamy, podobnie jak w wersji z odejmowaniem, instrukcję iteracyjną **`while`**.

Wewnątrz pętli **`while`** wykonujemy kolejno przypisania: zmiennej *dzielnik* przypisujemy wartość zmiennej *b*, zmiennej *b* resztę z dzielenia *a* przez *b*, zmiennej *a* wartość zmiennej *dzielnik*. W trakcie wykonywania obliczeń w zmiennej *dzielnik* przechowujemy aktualny dzielnik.

Jeśli *b* jest równe zero, następuje wyjście z pętli **`while`** i funkcja zwraca wartość NWD równą *a*.

C++

```
 5  int nwd(int a, int b)
 6  {
 7      while(b != 0)
 8      {
 9          int dzielnik = b;
10          b = a % b;
11          a = dzielnik;
12      }
13      return a;
14  }
```

Rys. 3a. Definicja funkcji realizującej algorytm Euklidesa w wersji z dzieleniem (C++)

Python

```
def nwd(a, b):
    while b != 0:
        dzielnik = b
        b = a % b
        a = dzielnik
    return a
```

Rys. 3b. Definicja funkcji realizującej algorytm Euklidesa w wersji z dzieleniem (Python)

Ćwiczenie 3. Zapisujemy w języku C++ lub Python algorytm Euklidesa w wersji z dzieleniem

1. Zapoznaj się z wybraną definicją funkcji (rys. 3a lub 3b) i, korzystając z niej, przeprowadź analizę realizacji algorytmu Euklidesa w wersji z dzieleniem dla liczb 24 i 8. Objaśnij działanie funkcji na tym przykładzie.
2. W wybranym języku programowania napisz program realizujący algorytm Euklidesa w wersji z dzieleniem:
   a. Zdefiniuj funkcję zwracającą wartość *nwd()* z dwoma parametrami *a* i *b*,
   b. W programie głównym wywołaj funkcję *nwd()* z parametrami aktualnymi *liczba1* i *liczba2* wprowadzanymi z klawiatury; na ekran wyprowadzaj też odpowiednie komunikaty.
3. Zapisz program w pliku pod nazwą *Euklides_dzielenie* i przetestuj dla kilku różnych danych.

Warto zapamiętać

- W programie realizującym algorytm badania pierwszości liczby sprawdzamy, czy dana liczba *n* ma dzielniki w zbiorze $\langle 2; \sqrt{n} \rangle$. Definiujemy funkcję logiczną, która zwraca wartość PRAWDA, gdy liczba nie ma tych dzielników, czyli jest liczbą pierwszą.
- W funkcji realizującej algorytm Euklidesa w wersji z odejmowaniem stosujemy instrukcję iteracyjną **`while`**, ponieważ nie znamy z góry liczby kroków iteracji. Iterację kończymy, gdy liczby *a* i *b* są równe.
- W funkcji realizującej algorytm Euklidesa w wersji z dzieleniem stosujemy instrukcję iteracyjną **`while`**, ponieważ nie znamy z góry liczby kroków iteracji. Iterację kończymy, gdy liczba *b* jest równa zero.

Pytania i polecenia

1. Dlaczego do zrealizowania algorytmu badania pierwszości liczby zdefiniowano funkcję logiczną.
2. Dlaczego w funkcji realizującej algorytm Euklidesa w wersji z odejmowaniem w wersji z dzieleniem zastosowano instrukcję iteracyjną **`while`**, a nie **`for`**?
3. Kiedy następuje wyjście z pętli **`while`** w funkcji realizującej algorytm Euklidesa w wersji odejmowaniem?
4. W jakim celu użyto dodatkowej zmiennej *dzielnik* w funkcji realizującej algorytm Euklidesa w wersji dzieleniem?

Zadania

1. Zmodyfikuj program zapisany w ćwiczeniu 1., aby można było sprawdzić pierwszość pięciu liczb. Zapisz program w pliku pod nazwą *Pierwsza_5*.
2. Zmodyfikuj program zapisany w ćwiczeniu 2., aby szukał NWD dla dziesięciu par liczb *a* i *b* wprowadzanych kolejno z klawiatury i wyświetlał wyniki na ekranie. Zapisz program w pliku pod nazwą *Euklides_odejmowanie_10*.
3. Zmodyfikuj program zapisany w ćwiczeniu 3., aby szukał NWD dla siedmiu par liczb *a* i *b* wprowadzanych kolejno z klawiatury i wyświetlał wyniki na ekranie. Zapisz program w pliku pod nazwą *Euklides_dzielenie_7*.

Dla zainteresowanych

4. Zapisz w wybranym języku programowania rekurencyjną realizację algorytmu Euklidesa w wersji z odejmowaniem. Zapisz program w pliku pod nazwą *Euklides_rekurencyjnie*.

Temat 98.

Programowanie działań na ułamkach zwykłych z wykorzystaniem NWD i NWW

1. **Funkcja obliczająca NWW**
2. **Zapisywanie dodawania i odejmowania ułamków zwykłych w językach C++ i Python**
3. **Zapisywanie mnożenia i dzielenia ułamków zwykłych w językach C++ i Python**

Warto powtórzyć

1. Czym jest NWD, a czym NWW?
2. Jaka jest zależność między NWD i NWW?
3. Jak skracamy ułamki?
4. W jaki sposób dodajemy i odejmujemy ułamki zwykłe?
5. Jak mnożymy i dzielimy ułamki zwykłe?

1. Funkcja obliczająca NWW

W obliczeniach, w których posługujemy się ułamkami zwykłymi, ma zastosowanie znajdowanie największego wspólnego dzielnika i najmniejszej wspólnej wielokrotności. Do zapisania działań na ułamkach zwykłych w języku programowania będziemy potrzebowali funkcji obliczającej największy wspólny dzielnik (NWD) i najmniejszą wspólną wielokrotność (NWW).

Zdefiniujemy funkcję *nww()* zwracającą wartość wspólnej wielokrotności dwóch liczb z dwoma parametrami *a* i *b* będącymi tymi liczbami. W definicji funkcji *nww()* użyjemy funkcji *nwd()*, którą zdefiniowaliśmy w temacie 97.

C++

```
17  int nww(int a, int b)
18  {
19      return (a / nwd(a, b)) * b;
20  }
```

Rys. 1a. Funkcja obliczająca NWW (C++)

Python

```
def nww(a, b):
    return (a / nwd(a, b)) * b
```

Rys. 1b. Funkcja obliczająca NWW (Python)

Ćwiczenie 1. Piszemy funkcję wyznaczającą NWW

1. Zdefiniuj funkcję *nww()* wyznaczającą i zwracającą NWW dwóch nieujemnych liczb całkowitych *a* i *b*, będących parametrami tej funkcji.
2. Wywołaj funkcję w programie głównym z parametrami aktualnymi *a* i *b*. Wyprowadzaj wynik na ekran wraz z odpowiednimi komunikatami. Możesz również wyprowadzać wartość NWD.
3. Zapisz program w pliku pod nazwą *NWW* i przetestuj jego działanie dla różnych danych.

Wskazówka: Skorzystaj z definicji funkcji *nwd()* z tematu 97. (ćwiczenie 2.).

2. Zapisywanie dodawania i odejmowania ułamków zwykłych w językach C++ i Python

Ułamek zwykły opisują dwie liczby całkowite: licznik i mianownik przedzielone kreską ułamkową.

W programach komputerowych, w których będziemy definiowali funkcje wykonujące działania na ułamkach zwykłych, ułamek zwykły będziemy reprezentowali w postaci pary liczb całkowitych.

Do wszystkich obliczeń wykorzystamy wzory przedstawione w temacie 87.

W funkcjach przyjmiemy następujące oznaczenia dla zmiennych:

l1 – licznik pierwszego ułamka,
m1 – mianownik pierwszego ułamka,
l2 – licznik drugiego ułamka,
m2 – mianownik drugiego ułamka,
lw – licznik wyniku,
mw – mianownik wyniku.

Uwaga

W ułamku mianownik musi być zawsze różny od zera. Dla potrzeb programów przyjęliśmy, że licznik również jest różny od zera.

Przykład 1. Określenie zadania i jego specyfikacji

Zadanie: Napisz program w wybranym języku programowania obliczający sumę dwóch ułamków zwykłych

Dane: cztery liczby całkowite różne od zera: *l1* i *m1* (licznik i mianownik pierwszego ułamka), *l2* i *m2* (licznik i mianownik drugiego ułamka).

Wynik: dwie liczby całkowite licznik *lw* i mianownik *mw* sumy ułamków (w postaci nieskracalnej).

W programie zdefiniujemy funkcję *suma_ulamki()* niezwracającą wartości z czterema parametrami: *l1*, *m1*, *l2*, *m2*. W funkcji obliczamy licznik i mianownik sumy i wyprowadzamy wyniki na ekran.

W funkcji *suma_ulamki()* wywołamy funkcję *nwd()*, potrzebną do skrócenia otrzymanego licznika i mianownika sumy.

Python W języku Python dodano w instrukcji wyświetlającej wyniki określenie typu wyświetlanych danych **int**, aby wartości licznika i mianownika wyświetlały się jako liczby całkowite.

C++

```
21  void suma_ulamki(int l1, int l2, int m1, int m2)
22  {
23      int mw = nww(m1, m2);
24      int lw = l1 * mw / m1 + l2 * mw / m2;
25      int nwd_w = nwd(lw, mw);
26      cout << "Suma = " << lw / nwd_w << "/" << mw / nwd_w << endl;
27  }
```

Rys. 2a. Funkcja dodająca ułamki zwykłe (C++)

Python

```
def suma_ulamki(l1, l2, m1, m2):
    mw = nww(m1, m2);
    lw = l1 * mw / m1 + l2 * mw / m2;
    nwd_w = nwd(lw, mw);
    print("Suma =", int(lw / nwd_w), "/", int(mw / nwd_w))
```

Rys. 2b. Funkcja dodająca ułamki zwykłe (Python)

Ćwiczenie 2. Piszemy program dodający ułamki zwykłe

1. Zapoznaj się z wybraną definicją funkcji sumującej ułamki (rys. 2a lub 2b). Który ze wzorów na sumę ułamków podanych w temacie 87. wykorzystano w tej funkcji? Objaśnij działanie poszczególnych instrukcji.
2. W wybranym języku programowania napisz program sumujący ułamki zwykłe:
   a. Zdefiniuj funkcję *suma_ulamki()* z czterema parametrami: *l1*, *m1*, *l2*, *m2*.
   b. W programie głównym wywołaj funkcję *suma_ulamki()* z parametrami aktualnymi: *licz1*, *mian1*, *licz2*, *mian2*.
3. Zapisz program w pliku pod nazwą *Dzialania_ulamki* i przetestuj jego działanie dla kilku różnych danych.

Wskazówka: Skorzystaj z definicji funkcji *nww()* z ćwiczenia 1.

Podobnie obliczymy różnicę ułamków, w powyższych wzorach zamieniając dodawanie na odejmowanie.

Ćwiczenie 3. Piszemy program odejmujący ułamki zwykłe

1. Napisz specyfikację zadania odejmowania ułamków zwykłych.
2. Otwórz plik *Dzialania_ulamki* zapisany w ćwiczeniu 2.
   a. Zdefiniuj funkcję *roznica_ulamki()* z czterema parametrami (*l1*, *m1*, *l2*, *m2*), odejmującą ułamki zwykłe. Pamiętaj o skróceniu otrzymanego wyniku.
   b. Wywołaj funkcję *roznica_ulamki()* w programie głównym z parametrami aktualnymi: *licz1*, *mian1*, *licz2*, *mian2*.
3. Zapisz plik pod tą samą nazwą i przetestuj jego działanie dla różnych danych.

W praktyce, o ile nie wykonujemy działań na dużych liczbach, przekraczających zakres dopuszczalnych wartości danego typu zmiennej, wygodniej jest korzystać z metody bez użycia NWW, ponieważ wyznaczenie NWW jest stosunkowo czasochłonne (wymaga uprzedniego iteracyjnego lub rekurencyjnego wyznaczenia NWD).

3. Zapisywanie mnożenia i dzielenia ułamków zwykłych w językach C++ i Python

W programie zdefiniujemy funkcję *iloczyn_ulamki()* niezwracającą wartości z czterema parametrami *l1*, *m1*, *l2*, *m2*. W funkcji obliczamy licznik i mianownik iloczynu i wyprowadzamy wyniki na ekran.

W funkcji *iloczyn_ulamki()* wywołamy funkcję *nwd()*, potrzebną do skrócenia otrzymanego licznika i mianownika iloczynu.

C++

```
36  void iloczyn_ulamki(int l1, int l2, int m1, int m2)
37  {
38      int li = l1 * l2;
39      int mi = m1 * m2;
40      int nwd_w = nwd(li, mi);
41      cout << "Iloczyn = " << li / nwd_w << "/" << mi / nwd_w << endl;
42  }
```

Rys. 3a. Funkcja mnożąca ułamki zwykłe (C++)

Python

```
def iloczyn_ulamki(l1, l2, m1, m2):
    li = l1 * l2;
    mi = m1 * m2;
    nwd_w = nwd(li, mi);
    print("Iloczyn =", int(li / nwd_w), "/", int(mi / nwd_w))
```

Rys. 3b. Funkcja mnożąca ułamki zwykłe (Python)

Ćwiczenie 4. Piszemy program mnożący ułamki zwykłe

1. Napisz specyfikację zadania mnożenia ułamków zwykłych.
2. Zapoznaj się z wybraną definicją funkcji mnożącej ułamki zwykłe (rys. 3a lub 3b). Objaśnij działanie poszczególnych instrukcji.
3. Otwórz plik *Dzialania_ulamki* zapisany w ćwiczeniu 3.
   a. Zdefiniuj funkcję *iloczyn_ulamki()* z czterema parametrami: *l1*, *m1*, *l2*, *m2*.
   b. W programie głównym wywołaj funkcję *iloczyn_ulamki* z parametrami aktualnymi: *licz1*, *mian1*, *licz2*, *mian2* .
4. Zapisz program w pliku pod tą samą nazwą i przetestuj jego działanie dla kilku różnych danych.

Ćwiczenie 5. Piszemy program dzielący ułamki zwykłe

1. Napisz specyfikację zadania dzielenia ułamków zwykłych.
2. Otwórz plik *Dzialania_ulamki* zapisany w ćwiczeniu 4.
   a. Zdefiniuj funkcję *iloraz_ulamki()*, z czterema parametrami (*l1*, *m1*, *l2*, *m2*), dzielącą ułamki zwykłe. Pamiętaj o skróceniu otrzymanego wyniku.
   b. Wywołaj funkcję *iloraz_ulamki()* w programie głównym z parametrami aktualnymi: *licz1*, *mian1*, *licz2*, *mian2*.
3. Zapisz program w pliku pod tą samą nazwą i przetestuj jego działanie dla kilku różnych danych.

Warto zapamiętać

- Ułamek zwykły opisujemy przez dwie wartości całkowite: licznik i mianownik.
- Do zapisania działań na ułamkach w wybranym języku programowania wykorzystujemy funkcje obliczające największy wspólny dzielnik NWD i najmniejszą wspólną wielokrotność NWW.
- Aby zaprogramować funkcje wykonujące działania na ułamkach zwykłych: dodawanie, odejmowanie, mnożenie i dzielenie, można zdefiniować funkcje niezwracające wartości z czterema parametrami *l1*, *m1*, *l2*, *m2*, będącymi odpowiednio wartościami liczników i mianowników dwóch ułamków. W każdej funkcji obliczamy i wyprowadzamy na ekran licznik i mianownik wyniku.

Pytania i polecenia

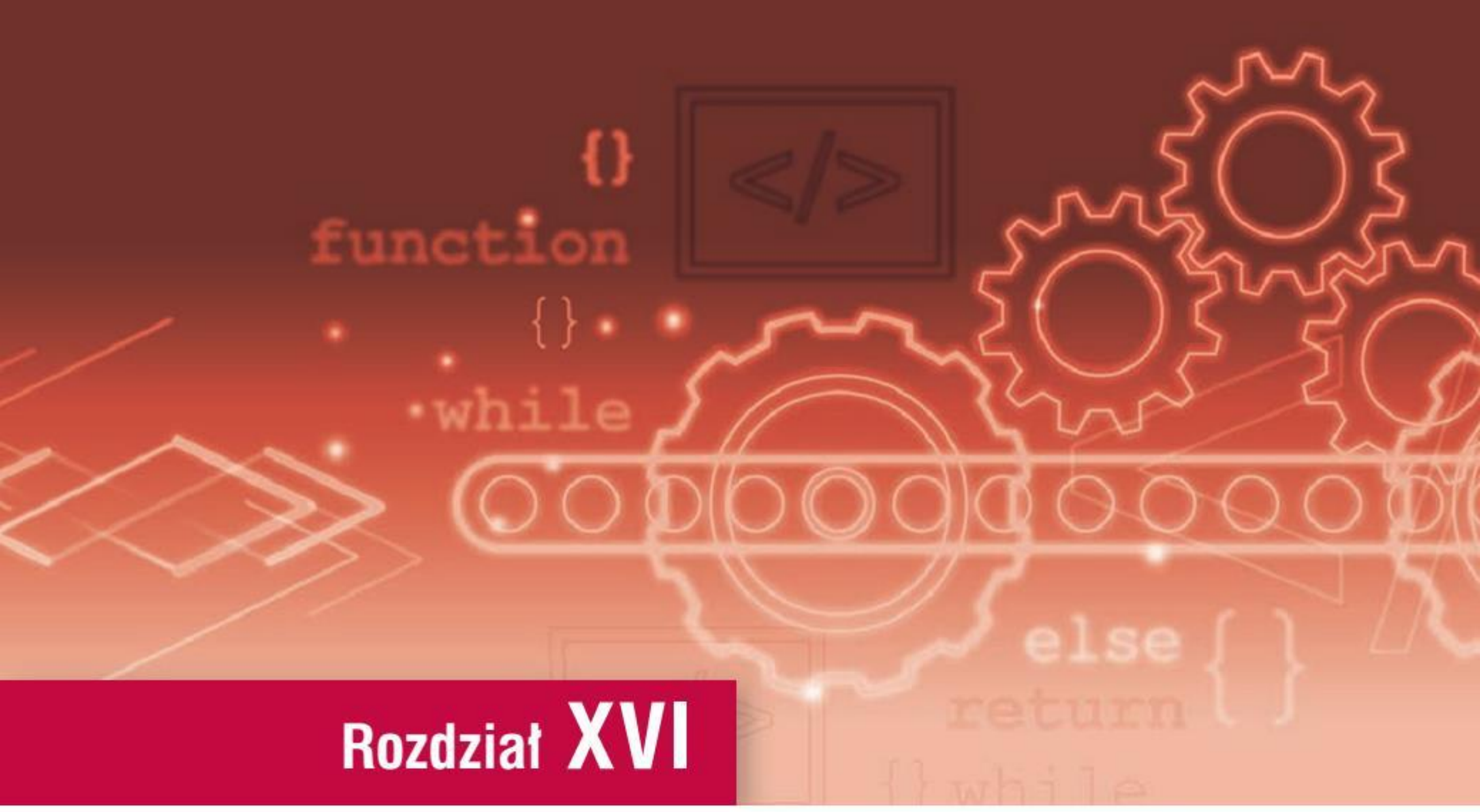
1. Czym jest ułamek zwykły? Jak można zaprezentować go w programie komputerowym?
2. Które algorytmy możemy wykorzystać do zapisania w wybranym języku programowania działań na ułamkach zwykłych?
3. W jaki sposób możemy zaprogramować działania na ułamkach zwykłych w wybranym języku programowania?

Zadania

1. Napisz program obliczający sumę ułamków, wykorzystując inny wzór niż zastosowany na rysunkach 2a i 2b. Zaproponuj inny sposób wyświetlania wyniku niż pokazany na rysunkach 2a i 2b. Zapisz program w pliku pod nazwą *Dodawanie_ulamkow*.
2. Zmodyfikuj program *Dzialania_ulamki* zapisany w ćwiczeniu 5., dodając menu programowe z pytaniem o działanie, jakie użytkownik chce wykonać na ułamkach. Zapisz program w pliku pod nazwą *Dzialania_ulamki_z_menu*.

Dla zainteresowanych

3. Zmodyfikuj program *Dzialania_ulamki* zapisany w ćwiczeniu 5., dodając sprawdzanie, czy wprowadzane dane są poprawne (mianownik nie może być zerem). Zapisz program w pliku pod nazwą *Dzialania_ulamki_popr*.
4. Napisz program obliczający wartość wyrażenia $(u_1 + u_2)^2$, gdzie u_1 i u_2 są ułamkami zwykłymi. Skorzystaj ze wzoru skróconego mnożenia. Zapisz program w pliku pod nazwą *Kwadrat_sumy_ulamkow*.



Rozdział XVI

Projekty – elementy robotyki z wykorzystaniem Arduino

Temat 99. Sterowanie nawadnianiem ogrodu – projekt

Temat 100. Projekt „Zamknij okno”

Temat 101. Projekt „Wycisz radio”

Temat 102. Budowanie robota – projekt

Temat 99.

Sterowanie nawadnianiem ogrodu – projekt

1. **Wprowadzenie do robotyki**
2. **Czym jest Arduino?**
3. **Projekt sterowania nawadnianiem ogrodu**
   - 3.1 Zadanie projektowe
   - 3.2 Zadania szczegółowe

1. Wprowadzenie do robotyki

Robotyka łączy zagadnienia z takich dziedzin, jak: inżynieria mechaniczna, elektrotechnika, **automatyka** i informatyka. Celem robotyki jest tworzenie robotów, w tym sterowanie ich ruchem. Robotyka pozwala automatyzować różne czynności i procesy w przemyśle czy – coraz częściej – w życiu codziennym.

Zadania projektowe w tym rozdziale będziemy realizować z wykorzystaniem układu **Arduino**.

Zastosujemy również poznane dotychczas zagadnienia związane z technikami algorytmicznymi (sytuacja warunkowa, iteracja) oraz ich realizacjami w językach programowania.

2. Czym jest Arduino?

Arduino to rodzina układów zawierających **mikrokontrolery**, które umożliwiają budowę robotów i prostych systemów automatyki.

Uwaga !

Programy dla Arduino możemy tworzyć w środowisku Arduino IDE (dostępnym bezpłatnie na stronie *www.arduino.cc*)

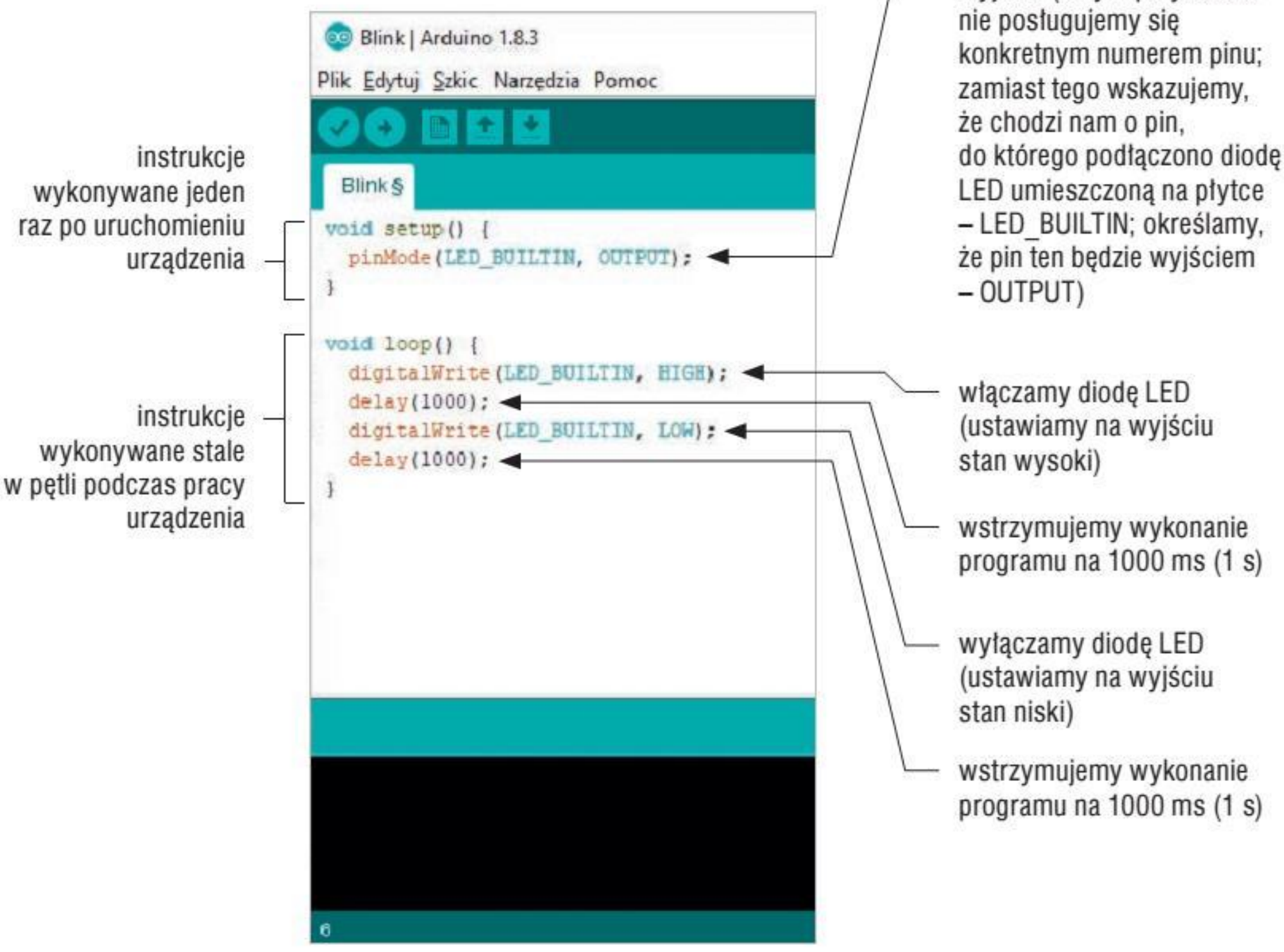
Mikrokontroler to układ elektroniczny, zawierający mikroprocesor, pamięci RAM i ROM oraz posiadający wejścia i wyjścia, do których można podłączyć różne urządzenia (np. diodę LED, przycisk, wyświetlacz, czujnik temperatury). Układ ten programuje się, aby ustalić, jak mają się zachowywać wyjścia urządzenia na podstawie parametrów wejściowych, w tym związanych z czasem. Najprostszy przykład to włączenie światła, gdy robi się ciemno albo gdy jest ciemno i wykryto ruch w pomieszczeniu.

W układach opartych o Arduino często wykorzystuje się gotowe moduły elektroniczne, które pozwalają na budowę złożonych urządzeń z wykorzystaniem jedynie elementarnej wiedzy z zakresu elektroniki (używamy tylko przewodów do łączenia modułów). Początkującym użytkownikom pomogą liczne przykłady rozwiązań dostępne w Internecie.

Typowa płytka Arduino zawiera mikrokontroler, cyfrowe i analogowe linie wejścia/wyjścia oraz interfejs USB do połączenia z komputerem. Komputer służy do programowania kontrolera. Można go też wykorzystać do interakcji z Arduino w czasie działania, choć Arduino po zaprogramowaniu działa jako samodzielny układ.

Programowanie Arduino opiera się o język programowania zbliżony do języka C. W każdym programie dla Arduino powinny znajdować się dwie funkcje (rys. 1.):

- `setup()` – funkcja zawierająca polecenia wykonywane na początku programu (po uruchomieniu urządzenia),
- `loop()` – funkcja wykonywana w pętli po wykonaniu funkcji `setup()`.

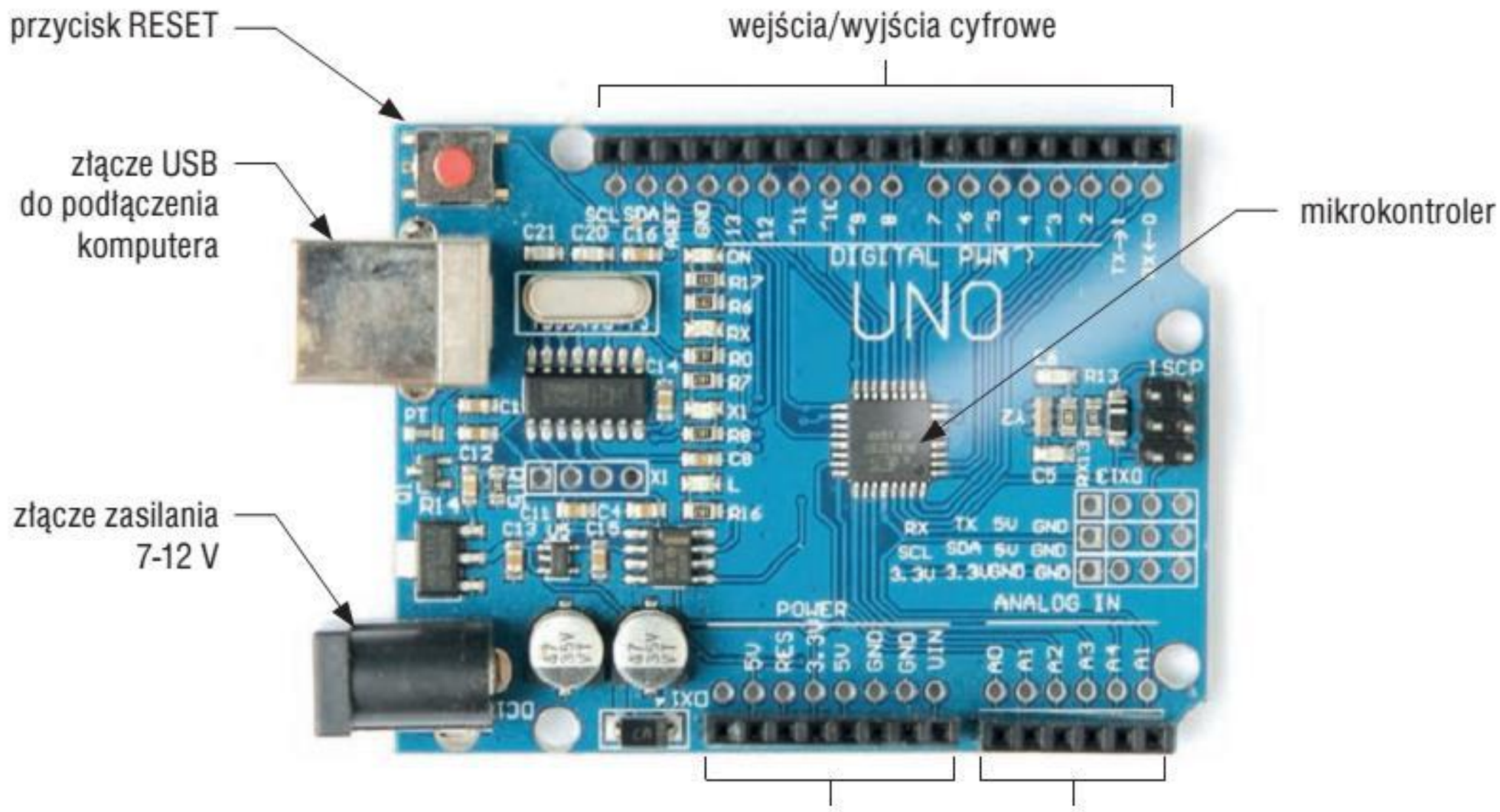


Rys. 1. Przykładowy program dla Arduino – mruganie diody umieszczonej na płytce

W programie pokazanym na rysunku 1. w funkcji `setup()` wywołujemy funkcję `pinMode()` z dwoma parametrami: pierwszy określa numer **pinu**, drugi – sposób działania pinu (wejście lub wyjście układu).

P **Pin w elektronice**

Styk, przez który wprowadzamy sygnały do urządzenia lub z niego wyprowadzamy.



Rys. 2. Płytka zgodna z Arduino Uno

3. Projekt sterowania nawadnianiem ogrodu

3.1. Zadanie projektowe

Celem projektu „Podlewanie ogrodu" jest zaprojektowanie prostego systemu sterowania nawadnianiem ogrodu. Zakładamy, że podlewamy, gdy przez dłuższy czas nie pada deszcz (czyli ziemia jest sucha). Czynności te oczywiście **powtarzamy**, gdy spada wilgotność ziemi (rys. 3.). Mamy tu więc do czynienia z **sytuacją warunkową** i **iteracją** (powtarzaniem czynności podlewania).





Rys. 3. Schemat blokowy algorytmu podlewania ogrodu

W przypadku nawadniania ogrodu jedyną akcją, którą musimy podjąć, jest włączenie lub wyłączenie nawadniania (przez włączenie pompy albo otwarcie elektrozaworu). Gdybyśmy nawadniali szklarnię, moglibyśmy jeszcze sterować wentylatorem lub otwieraniem okien w przypadku zbyt wysokiej temperatury.

Na rysunku 3. pokazano uproszczony schemat podlewania ogrodu. Zakładamy w nim, że podlewamy ogród tylko wtedy, gdy ziemia jest sucha.

W prawdziwym systemie nie byłby to jednak jedyny warunek. Jeśli jest silne słońce, powstrzymujemy się od podlewania. Podobnie, gdy właśnie zaczął padać deszcz – wtedy nie podlewamy (nawet gdy ziemia jest sucha), tylko czekamy, aż deszcz wykona pracę za nas. Dlatego w docelowym projekcie rozszerzymy system o czujnik światła oraz czujnik deszczu.

Aby móc wykonywać rzeczywiste pomiary (m.in. temperatury czy wilgotności gleby), musimy wybrać odpowiednie moduły i elementy, których użyjemy do naszego projektu (pokazano je w tabeli 1.).

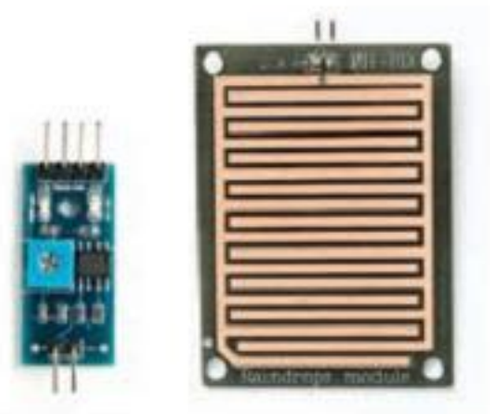
Czujnik deszczu oraz jego sonda	
Czujnik wilgotności gleby oraz jego sonda	
Czujnik światła	
Przekaźnik do sterowania urządzeniami zasilanymi prądem o dużym napięciu i natężeniu – np. lampą, grzałką czy pompą	
Pompa	

Tabela 1. Moduły i elementy użyte w projekcie sterowania nawadnianiem ogrodu

Uwaga

Dla początkujących użytkowników, chcących spróbować różnych możliwości, powstały gotowe układy zawierające mikrokontroler Arduino i zestaw czujników oraz urządzeń wyjściowych. Można w ten sposób skupić się na programowaniu, nie martwiąc się o połączenia elektryczne.

Jeśli chcielibyśmy włączać nawadnianie o określonych godzinach, potrzebny będzie nam także moduł zegara czasu rzeczywistego (**RTC** – z ang. ***R**eal **T**ime **C**lock*). W naszym projekcie możemy jednak bazować na czujniku oświetlenia – będziemy podlewać, kiedy zrobi się ciemno.

Do łączenia prototypowych układów służą płytki stykowe (rys. 4.) i przewody zakończone specjalnymi twardymi końcówkami. W płytkach styki połączono kolumnami, a dwie skrajne linie z każdej strony – wierszami, które służą do zasilania. Przewody podłączamy poprzez wbicie ich w odpowiednie pola.

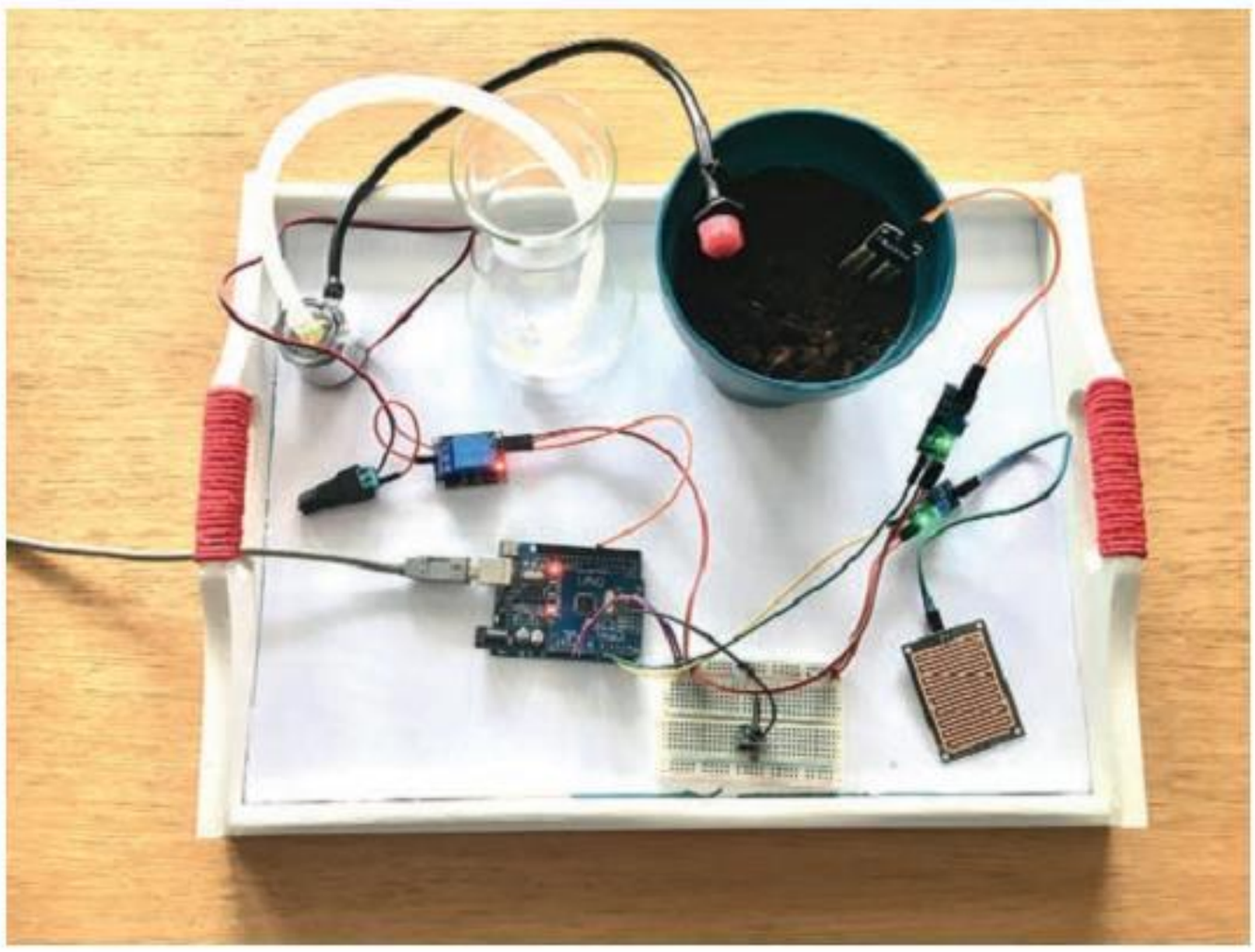


Rys. 4. Płytka stykowa – brązowymi liniami zaznaczono wewnętrzne połączenia poszczególnych pól

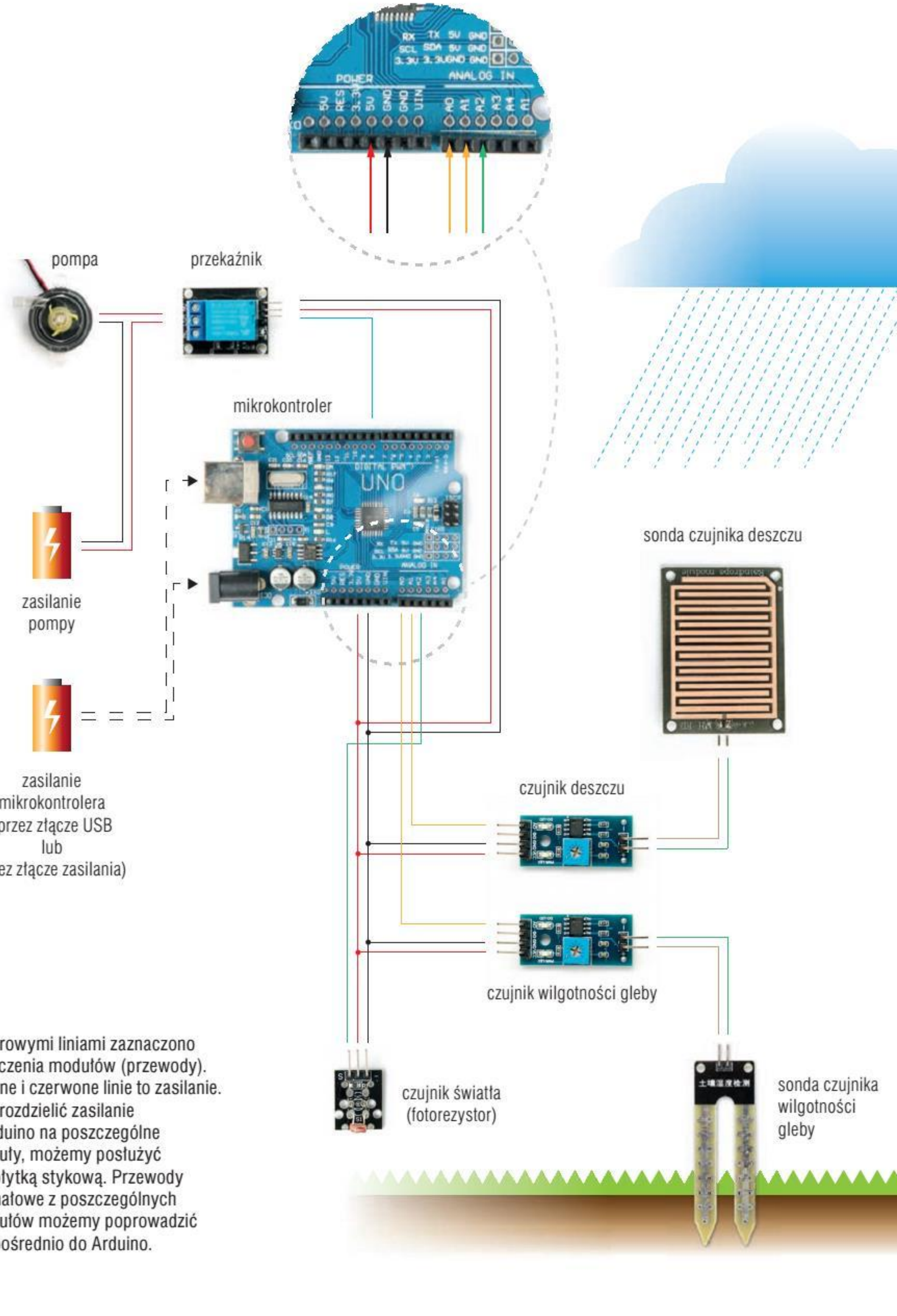
Na rysunku 5. pokazano zmontowany układ nawadniania ogrodu połączony za pomocą płytki stykowej. Zastosowano czujnik wilgotności gleby, czujnik opadów oraz czujnik natężenia oświetlenia (umieszczony na płytce stykowej). Podłączono też przekaźnik (urządzenie wyjściowe), który będzie włączał pompę. Zanim jednak zmontujemy ten układ, warto przygotować jego schemat (rys. 6.).

Uwaga

Do celów testowych można zamiast przekaźnika i pompy zastosować diodę LED, sygnalizującą, że należy włączyć pompę. W tym czasie można wodą z konewki nawadniać doniczkę, w której umieszczono czujnik wilgotności gleby (rys. 5.) lub wkładać czujnik do szklanki z wodą.



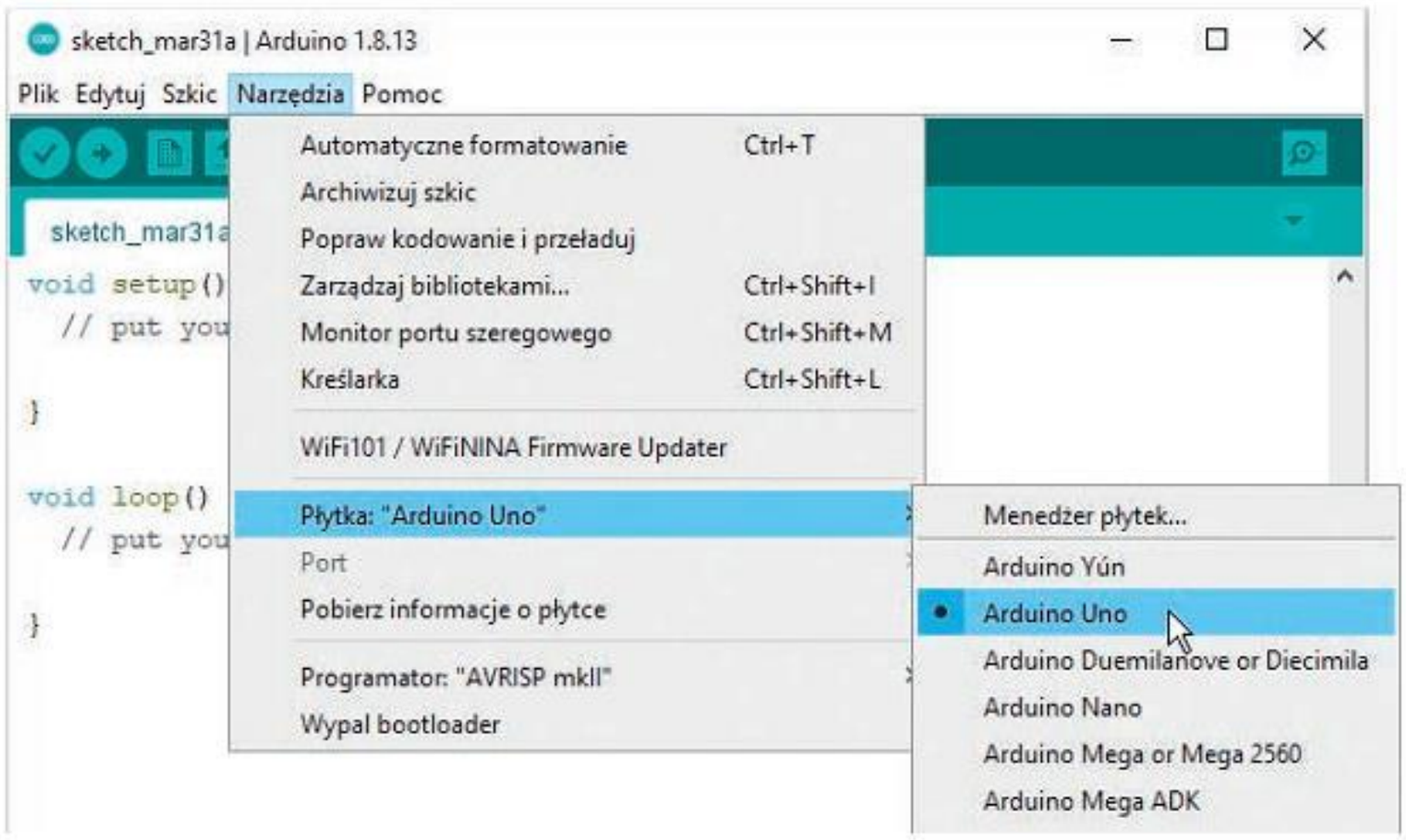
Rys. 5. Prototyp układu nawadniania ogrodu



Kolorowymi liniami zaznaczono połączenia modułów (przewody). Czarne i czerwone linie to zasilanie. Aby rozdzielić zasilanie z Arduino na poszczególne moduły, możemy posłużyć się płytką stykową. Przewody sygnałowe z poszczególnych modułów możemy poprowadzić bezpośrednio do Arduino.

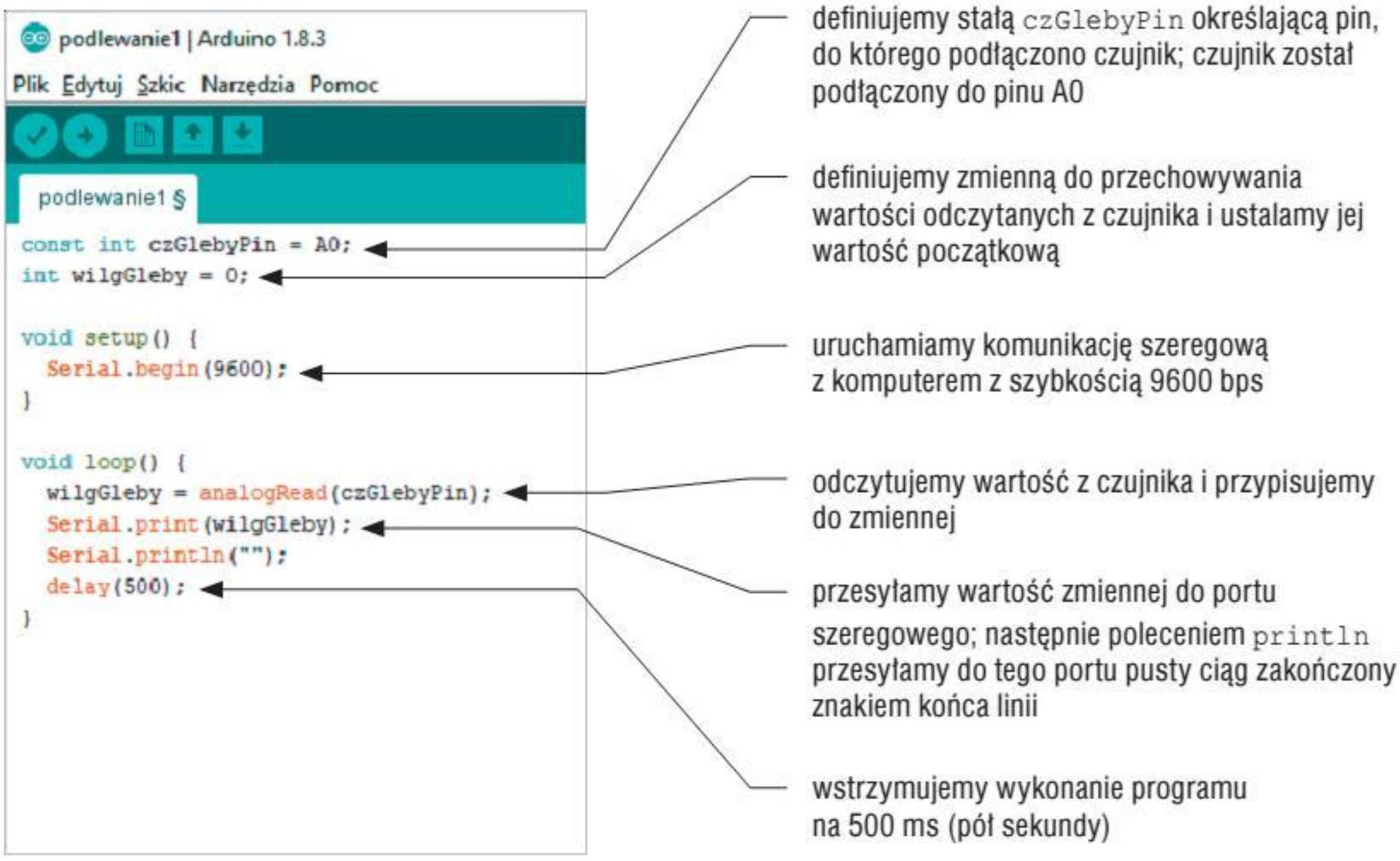
Rys. 6. Schemat połączeń modułów składających się na system sterowania nawadnianiem ogrodu

W środowisku Arduino IDE musimy wybrać wykorzystywaną płytkę, w naszym przypadku – Arduino Uno (rys. 7.). W tym samym menu należy również wybrać port, do którego jest podłączona wybrana płytka.



Rys. 7. Środowisko Arduino IDE

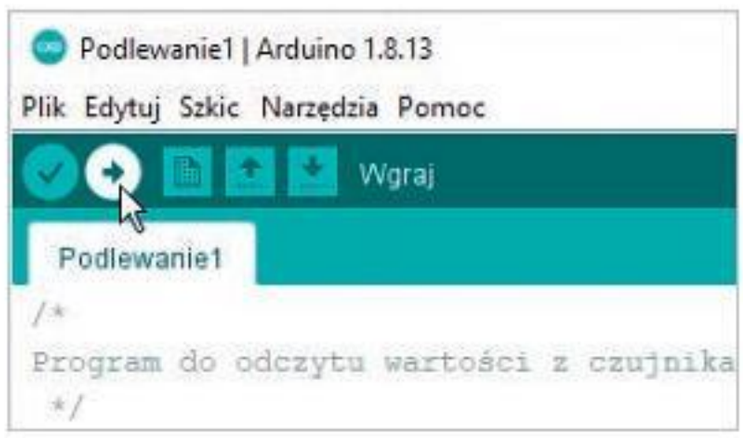
Napiszemy program, który odczyta wartości z czujnika wilgotności gleby. Przykładowy program łącznie z komentarzami pokazano na rysunku 8.



Rys. 8. Program odczytujący wartości z czujnika wilgotności gleby

Kolejnym krokiem jest przesłanie programu do płytki z mikrokontrolerem.

Aby przesłać program do płytki, należy kliknąć strzałkę w lewym górnym rogu lub wybrać z menu **Szkic** opcję **Wgraj** (rys. 9.). W dolnej części okna programu powinien po chwili pojawić się komunikat „Ładowanie zakończone".



Rys. 9. Menu programu z ikoną przesyłania

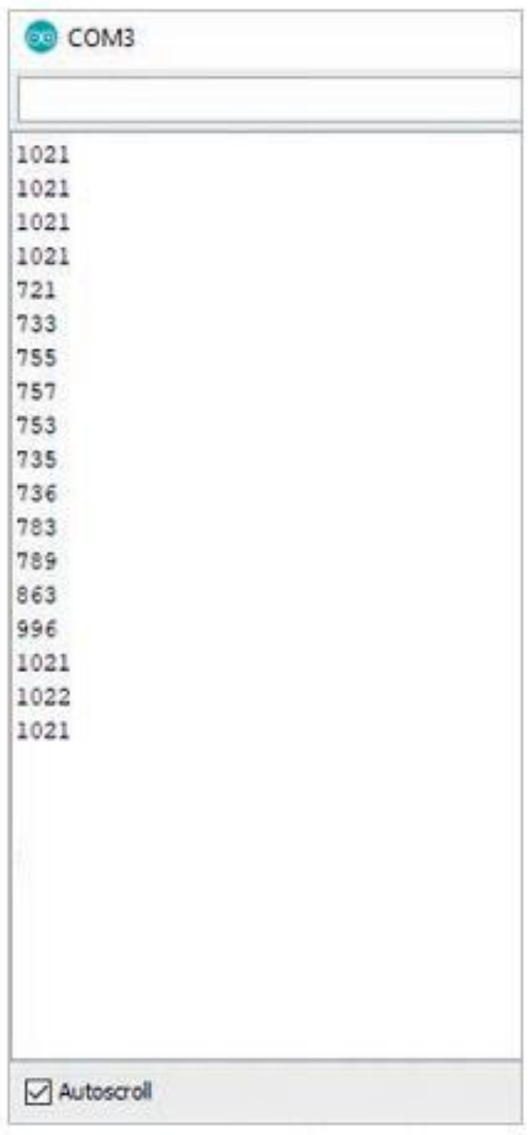
W programie umieściliśmy polecenia odpowiedzialne za komunikację z komputerem i wyświetlanie wyników pomiarów. Aby je zobaczyć, należy wybrać z menu **Narzędzia** opcję **Monitor portu szeregowego**.

Należy pamiętać, aby w prawym dolnym rogu nowo otwartego okna (rys. 10b) wybrać szybkość, którą określiliśmy w naszym programie (9600 bps).

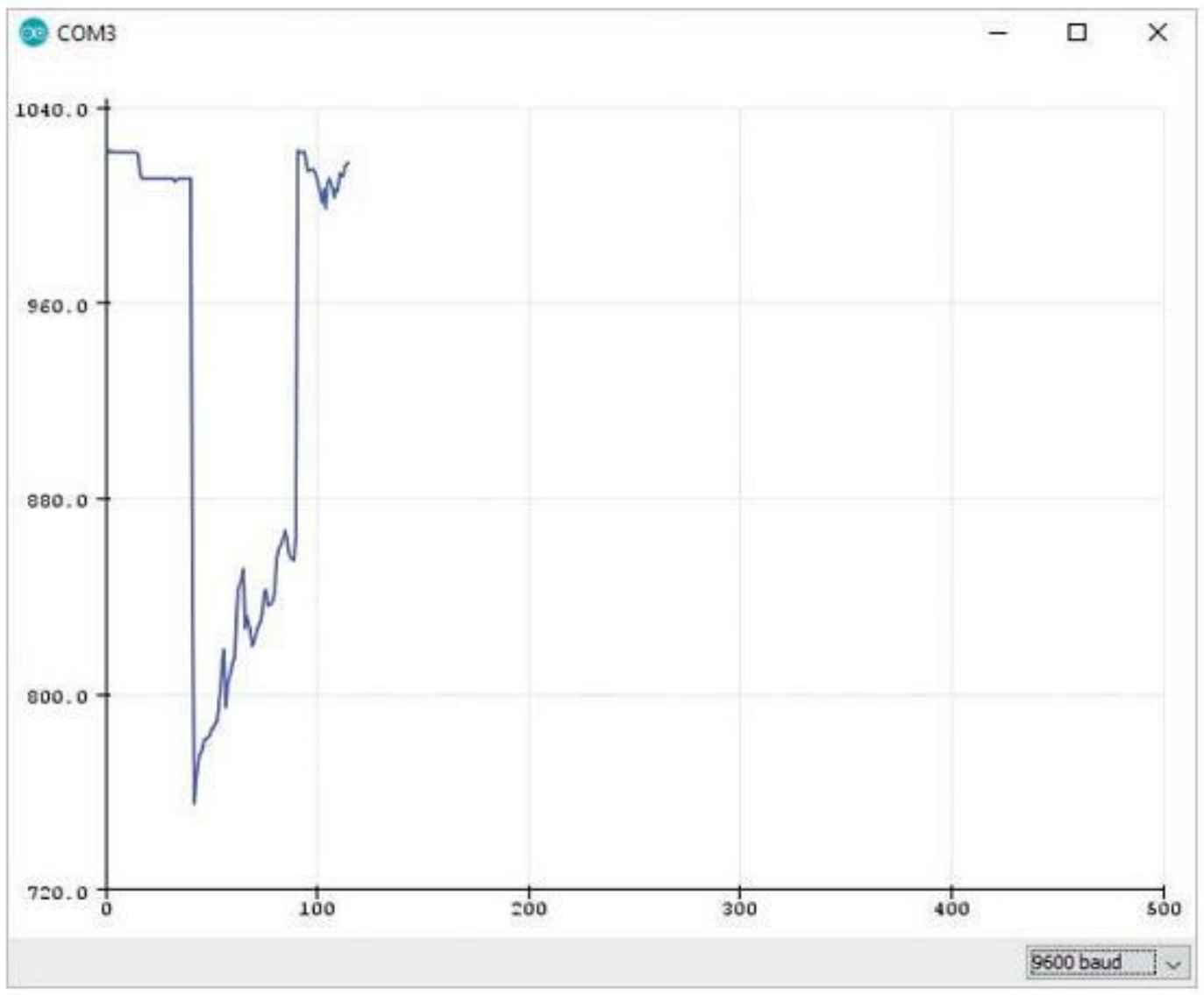
Zgodnie z oczekiwaniami, odczytane wartości pojawiają się w kolejnych wierszach (rys. 10a). Możemy te dane zobrazować na wykresie, aby łatwiej obserwować zmiany (rys. 10b). Należy w tym celu zamknąć monitor portu szeregowego i z menu **Narzędzia** wybrać opcję **Kreślarka**.

! Uwaga

Aby obserwować zmiany wartości z czujnika wilgotności gleby, możemy zanurzać stopniowo czujnik w wodzie lub dotykać jednocześnie obu styków zwilżonymi palcami.



Rys. 10a. Monitor portu szeregowego



Rys. 10b. Kreślarka – obrazowanie danych z monitora portu szeregowego na wykresie; zapis „9600 baud" oznacza to samo, co „9600 bps"

Zauważmy, że przy wzroście wilgotności odczytane wartości maleją – odczytujemy, jak bardzo ziemia jest sucha, a nie jak bardzo jest mokra – należy o tym pamiętać, tworząc warunek uruchamiający nawadnianie.

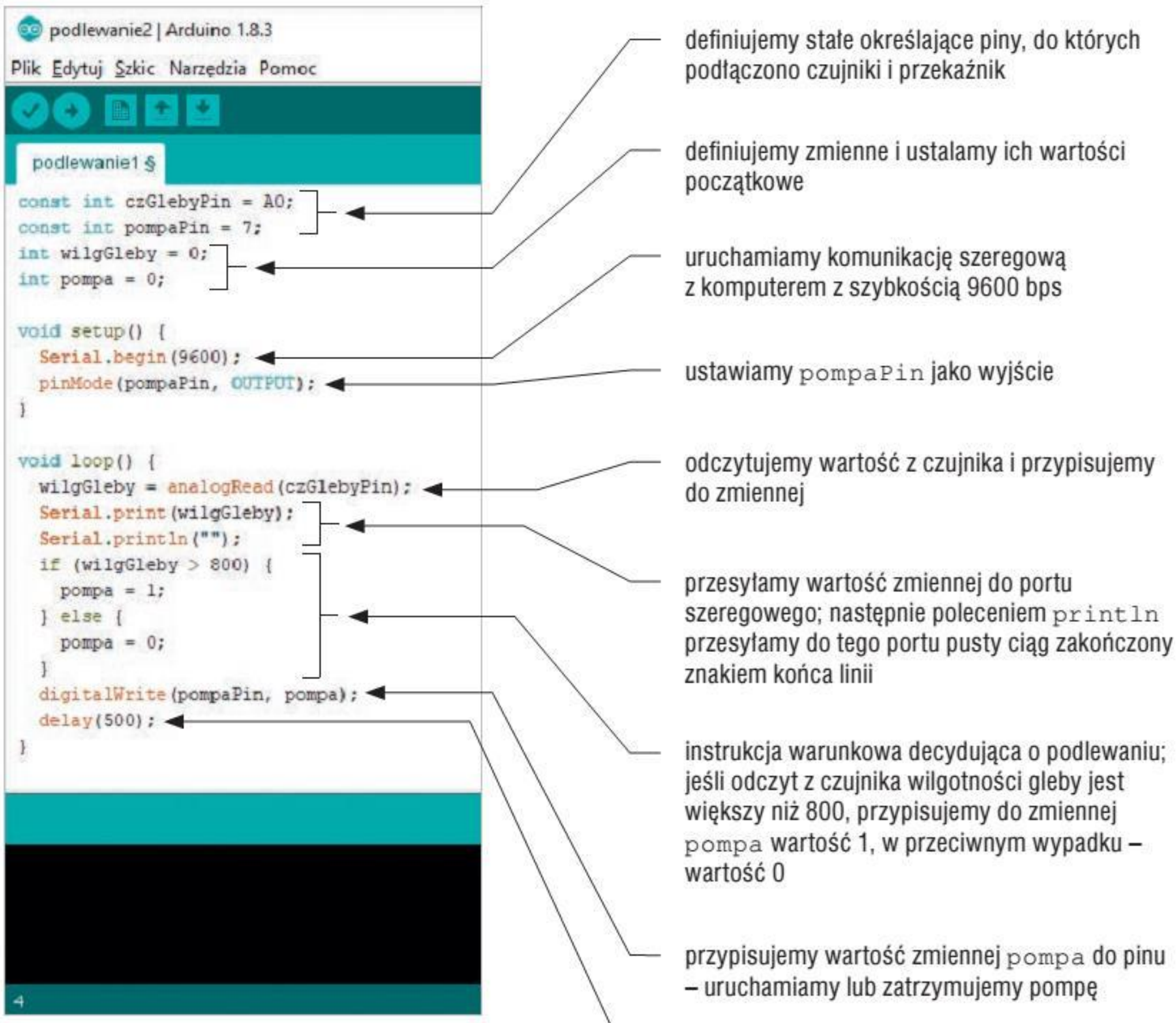
Możemy teraz dodać do naszego programu instrukcje odpowiedzialne za sterowanie wyjściem, czyli uruchamianie pompy.
Musimy dodać następujące linie programu:

```
const int pompaPin = 7;
int pompa = 0;
```

i w funkcji `setup()`:

```
pinMode(pompaPin, OUTPUT);
```

Możemy teraz dodać do programu instrukcję warunkową sterującą pompą (zgodnie ze schematem blokowym pokazanym na rysunku 3.) – podlewanie powinno zacząć działać.



Rys. 11. Program sterowania nawadnianiem na podstawie wilgotności gleby

3.2. Zadania szczegółowe

Zadanie 1. Wykonujemy system „Podlewanie ogrodu"

1. Zapoznaj się z opisem systemu i przygotuj odpowiednie pomoce.
2. Wykonaj przykładowy program – mruganie wbudowanej diody świecącej (rys 1.).
3. Wykonaj układ według wskazówek podanych w opisie projektu.
4. Przetestuj system (testowanie możesz przeprowadzić w podobny sposób, jak pokazano na rysunku 5. i opisano w uwadze w ramce).

Zaprogramowany układ można odłączyć od komputera i zasilać z powerbanku przez wejście USB lub z baterii albo zasilacza przez złącze zasilania.



Pamiętaj, aby korzystać tylko z bezpiecznych napięć!
Jeśli system będzie działał prawidłowo i zechcesz podłączyć go do pompy ogrodowej, pamiętaj, że może to zrobić wyłącznie elektryk z odpowiednimi uprawnieniami.

Zadanie 2. Rozszerzamy system „Podlewanie ogrodu"

1. W systemie „Podlewanie ogrodu" czujnik wilgotności gleby jest zasilany cały czas. Zmień połączenia, aby zasilić go z jednego z wyjść mikrokontrolera. Zaprogramuj to wyjście w taki sposób, aby czujnik był zasilany tylko na czas pomiaru – pomoże to uniknąć szybkiej korozji czujnika.
2. Dodaj do projektu czujnik nasłonecznienia (fotorezystor) oraz czujnik deszczu. Sprawdź odczyty z tych modułów i dostosuj algorytm – podlewamy tylko w nocy i jeśli nie pada.
3. Dodaj dodatkowy czujnik wilgotności gleby. Podlewanie powinno być uruchamiane, gdy którykolwiek z czujników wykryje zbyt niską wilgotność.
4. Dodaj do projektu drugi obwód pompy. Rozdziel algorytm tak, aby każdy z czujników wilgotności gleby sterował osobną pompą (czujnik światła i deszczu jest wspólny).
5. Przetestuj system.

Temat 100. Projekt „Zamknij okno”

Zadanie projektowe

Doświadczenia nabyte podczas przygotowania projektu systemu sterowania nawadnianiem ogrodu z zastosowaniem układu Arduino możemy wykorzystać do zautomatyzowania niektórych czynności w swoim pokoju.

Celem projektu jest utworzenie systemu „Zamknij okno”, sygnalizującego potrzebę zamknięcia okna w przypadku zmiany temperatury powietrza.

Potrzebę zamknięcia okna sygnalizuj:

a. przy temperaturze wewnętrznej poniżej 20 °C i temperaturze zewnętrznej niższej niż wewnętrzna (ochrona przed wychłodzeniem pomieszczenia),
b. przy temperaturze wewnętrznej powyżej 24 °C i temperaturze zewnętrznej wyższej niż wewnętrzna (ochrona przed przegrzaniem pomieszczenia).

W projekcie „Podlewanie ogrodu” poznaliśmy już metodę odczytu wartości sygnałów analogowych. Teraz wykorzystamy te umiejętności do pomiaru temperatury w pokoju.

Zadania szczegółowe

Zadanie 1. Przygotowujemy odpowiednie pomoce

Do wykonania zadania zastosujemy czujniki analogowe LM35, które na wyjściu podają napięcie zależne od temperatury. Do sygnalizacji potrzeby zamknięcia okna można wykorzystać diodę LED.

Zadanie 2. Odczytujemy temperaturę powietrza

W celu stworzenia systemu sygnalizacji „Zamknij okno” zastosuj w projekcie dwa czujniki temperatury – jeden wewnątrz pomieszczenia, drugi za oknem. W zależności od temperatury sygnalizuj, czy okno powinno zostać zamknięte.

Aby odczytać temperaturę, musimy najpierw wyliczyć napięcie otrzymane na wejściu Arduino:

```
float napiecie = odczyt * (5000 / 1024.0);
```

gdzie 5000 to napięcie odniesienia w miliwoltach (maksymalne, jakie można podać na wejście), a 1024 to rozdzielczość (tyle poziomów rozpoznaje Arduino – odczytana z wejścia wartość 0 odpowiada napięciu 0 V, a wartość 1023 – napięciu 5 V).

Następnie napięcie musimy przeliczyć na temperaturę:

```
float temperatura = napiecie / 10;
```

```
int odczyt = analogRead(sensorPin);
float napiecie = odczyt * (5000 / 1024.0);
float temperatura = napiecie / 10;
```

Zadanie 3. Porównujemy temperatury i decydujemy o stanie okna

Będziemy odczytywać dwie temperatury (wewnętrzną i zewnętrzną), w związku z czym musimy podłączyć dwa czujniki temperatury do dwóch oddzielnych wejść analogowych mikrokontrolera. Wejścia możemy nazwać *sensorPinWew* i *sensorPinZew*, a odczytane temperatury – *tempWew* i *tempZew*.

Jako sygnalizację możemy użyć diody LED znajdującej się na płytce mikrokontrolera. Przy podłączaniu zewnętrznej diody LED należy pamiętać o włączeniu w szereg rezystora ograniczającego prąd.

Zgodnie z przyjętymi wcześniej założeniami tworzymy instrukcję warunkową (rys. 1.).

```
if((tempWew < 20 && tempZew < tempWew) || (tempWew > 24 && tempZew > tempWew))
{
  // sygnalizuj potrzebę zamknięcia
  digitalWrite(LED_BUILTIN, HIGH);
}
else
{
  // wyłącz sygnalizację
  digitalWrite(LED_BUILTIN, LOW);
}
```

Rys. 1. Instrukcja warunkowa odpowiedzialna za sygnalizację, czy okno powinno zostać zamknięte – fragment programu



Rys. 2. Wodoodporna sonda z analogowym czujnikiem temperatury LM35



Rys. 3. Diody LED



Temat 101. Projekt „Wycisz radio”

Zadanie projektowe – dla zainteresowanych

Celem projektu jest zbudowanie urządzenia wyciszającego radio po otwarciu drzwi do pokoju. Radioodbiornik musi być wyposażony w pilota zdalnego sterowania. Na pilocie powinien znajdować się przycisk „mute” do szybkiego wyciszenia dźwięku.

Zadania szczegółowe

Zadanie 1. Przygotowujemy odpowiednie pomoce

Do zbudowania urządzenia wykorzystaj:

- płytkę Arduino,
- moduł odbiornika podczerwieni (np. VS1838B) – rys. 1.,
- diodę LED emitującą światło podczerwone (IR) – rys. 2.,
- czujnik otwarcia drzwi (np. kontaktron) – rys. 3.

Przy tworzeniu programu wykorzystamy bibliotekę `IRremote`. Aby z niej skorzystać, należy w Arduino IDE wybrać opcję **Szkic/Dołącz bibliotekę**.



Rys. 1. Moduł odbiornika podczerwieni



Rys. 2. Dioda LED emitująca światło podczerwone (IR)



Rys. 3. Czujnik magnetyczny – kontaktron

Zadanie 2. Odbieramy sygnały z pilota

Napisz program, który wyświetla w **Monitorze portu szeregowego** kody odebrane przez odbiornik podczerwieni. Znajdź kod, który jest przypisany do przycisku „mute" na pilocie od twojego radioodbiornika.

Skorzystaj z przykładowych programów w bibliotece `IRremote` (**Plik/Przykłady/IRremote**), aby przetestować odbieranie i wysyłanie sygnałów.

Zadanie 3. Tworzymy system

1. Napisz program, który po otwarciu drzwi (przy zastosowaniu kontaktronu będzie to rozwarcie styków czujnika) wyemituje za pomocą diody LED IR kod przypisany do przycisku „mute".
2. Sygnał IR wysyłamy do radia tylko w momencie otwarcia drzwi, a nie zawsze, gdy są otwarte. Musimy więc w zmiennej przechowywać poprzedni stan drzwi i sygnał wysyłać tylko w momencie, gdy stan drzwi zmienia się z zamkniętych na otwarte.

Do celów testowych czujnik otwierania drzwi można symulować, zwierając i rozwierając styki na płytce stykowej.



Temat 102. Budowanie robota – projekt

Zadanie projektowe

Celem zadania projektowego jest zbudowanie robota-zabawki, poruszającego się na kołach i omijającego przeszkody. Do wykonania zadania wykorzystamy układ Arduino.

Zadania szczegółowe

Zadanie 1. Przygotowujemy odpowiednie pomoce

W celu wykonania zadania będziesz potrzebować: płytki zgodnej z Arduino UNO, podwozia z dwoma silnikami, sterownika silników Motor Shield L293D, ultradźwiękowego czujnika odległości. Do zmontowania podwozia i podłączenia silników mogą być potrzebne wkrętak i lutownica.

Zadanie 2. Montujemy układ

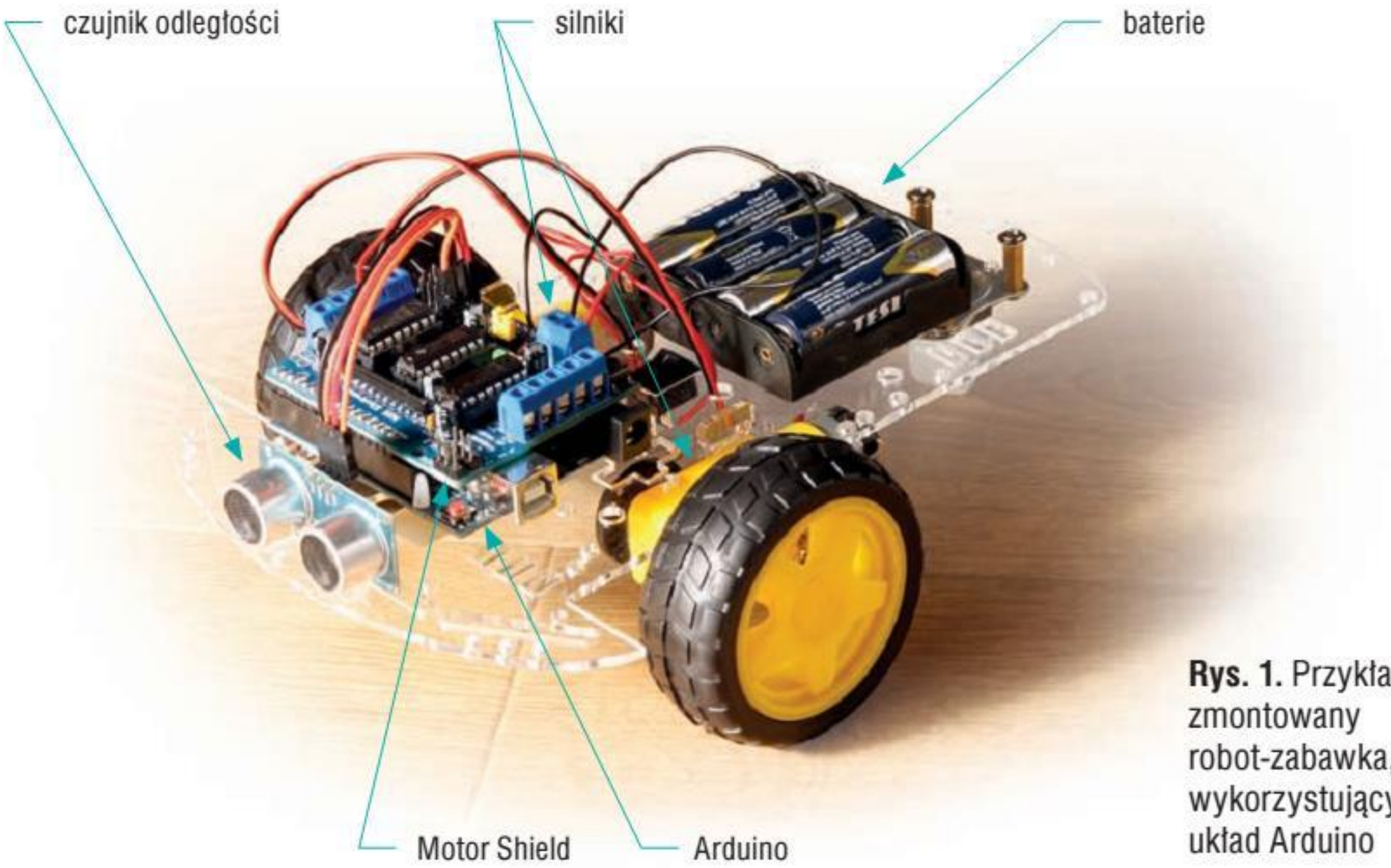
Pracę rozpocznij od zamontowania silników na podwoziu. Czujnik odległości, płytki z elektroniką oraz koszyk na baterie możesz zamontować później, tak aby nie było problemów z długością przewodów.

Silniki pobierają stosunkowo duży prąd, w związku z czym nie można ich zasilić bezpośrednio z Arduino. Ponieważ dodatkowo potrzebujemy odwracać polaryzację zasilania (zamieniać + i -), wymagany będzie specjalny układ zasilający silniki. Pomoże nam tutaj nakładka Motor Shield. Nakładkę umieszczamy (wciskamy) bezpośrednio w płytce z mikrokontrolerem.

Podłącz silniki do wybranych wyjść nakładki, np. oznaczonych **M3** i **M4**. Korzystając z wyprowadzeń wejść analogowych na nakładce, podłącz czujnik odległości. Pin czujnika odległości oznaczony **Trig** podłącz do wejścia **A0**, natomiast pin oznaczony **Echo** – do wejścia **A1**. Nie zapomnij o podłączeniu zasilania czujnika.

Pomiar odległości polega na wysłaniu impulsu ultradźwiękowego i zmierzeniu czasu, po jakim czujnik usłyszy jego odbicie (echo) – dlatego musimy użyć dwóch wyprowadzeń mikrokontrolera (jednego użyjemy jako wyjścia, drugiego jako wejścia).

Napisz program mierzący odległość przedmiotu od czujnika (rys. 2.).



Rys. 1. Przykładowy zmontowany robot-zabawka, wykorzystujący układ Arduino

PomiarOdleglosci | Arduino 1.8.3

Plik Edytuj Szkic Narzędzia Pomoc

PomiarOdleglosci

```
int Trig = A0;
int Echo = A1;
int DYSTANS;
long CZAS;

void setup()
{
  Serial.begin(9600);
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);
  Serial.println("Testujemy czujnik odleglosci");
}

void loop()
{
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);
  digitalWrite(Echo, HIGH);
  CZAS = pulseIn(Echo, HIGH);
  DYSTANS = CZAS / 58;

  Serial.print("Odleglosc: ");
  Serial.print(DYSTANS);
  Serial.println(" cm");
  delay(200);
}
```

seria impulsów uruchamiających pomiar odległości (są one opisane w dokumentacji czujnika)

Rys. 2. Przykładowy program mierzący odległość za pomocą czujnika ultradźwiękowego

Zadanie 3. Instalujemy bibliotekę AFMotor

Aby w prosty sposób sterować nakładką Motor Shield, a co za tym idzie silnikami, skorzystamy z gotowej biblioteki *AFMotor*. Aby zainstalować bibliotekę `AFMotor` (z ang. ***Adafruit Motor Shield Library***), wybierz opcję **Szkic/Dołącz bibliotekę/Zarządzaj bibliotekami**. Razem z biblioteką zainstalują się przykładowe programy.

Otwórz przykład *MotorTest* i w instrukcji `AF_DCMotor motor(4);` wstaw numer swojego silnika (numer wyjścia na nakładce), aby przetestować jego działanie. Kluczowymi poleceniami są:

`motor.run(FORWARD);` ruch silnika naprzód
`motor.run(BACKWARD);` ruch silnika wstecz
`motor.run(RELEASE);` odłączenie zasilania silnika

Aby sterować dwoma silnikami, musisz zadeklarować osobne zmienne, oznaczając je *silnik_lewy*, *silnik_prawy*:

```
AF_DCMotor silnik_lewy(3);
AF_DCMotor silnik_prawy(4);
```

Jeśli nie masz zamiaru zmieniać prędkości silników, możesz ją określić tylko raz – na początku programu w funkcji `setup()`.

Prędkość silnika określamy poleceniem `motor.setSpeed(x);`
gdzie zmienna *x* to wartość w przedziale od 0 do 255.

Przy mniejszych wartościach silnik może w ogóle się nie obracać, trzeba ustalić doświadczalnie minimalną wartość.

Dla zmiennych *silnik_lewy* i *silnik_prawy* polecenia określające prędkość silników będą miały postać:

```
silnik_lewy.setSpeed(x);
silnik_prawy.setSpeed(x);
```

Zadanie 4. Piszemy program, dzięki któremu robot omija przeszkody

Korzystając ze zdobytych doświadczeń, napisz program, w którym w przypadku przeszkód przed robotem oba silniki kół będą się kręcić do przodu. W przypadku przeszkody znajdującej się w bliskiej odległości jedno koło powinno się zatrzymać, a drugie – kręcić do tyłu. Efektem tego będzie cofnięcie i obrócenie się robota oraz ominięcie przeszkody. Możesz dodać instrukcję `delay(500);`, aby ruch robota do tyłu połączony z obracaniem trwał przynajmniej pół sekundy. Na koniec przetestuj system.

Zadanie 5. Dodajemy czujniki (dla zainteresowanych)

Dodaj do robota odbiciowy czujnik podczerwieni skierowany na podłogę i zmień program, aby przy braku odbicia (podniesienie robota lub niebezpieczeństwo upadku ze schodów) robot się zatrzymywał.

Dodaj czujniki odległości skierowane lekko na boki, aby bardziej precyzyjnie identyfikować położenie przeszkody i lepiej podejmować decyzje o skręcaniu. Możesz też pozostać przy jednym czujniku, ale obracać go za pomocą serwomechanizmu.

Słownik pojęć

A

Algorytm
Uporządkowany i uściślony sposób rozwiązywania danego problemu, zawierający szczegółowy opis wykonywanych czynności w skończonej liczbie kroków (str. 17).

D

Deszyfrowanie
Proces odtwarzania tekstu oryginalnego na podstawie treści szyfrogramu (str. 192).

I

Implementacja
Zapisanie algorytmu w postaci kodu źródłowego (str. 24).

Interpretacja
Tłumaczenie programu utworzonego w języku programowania instrukcja po instrukcji, tak aby komputer mógł wykonać każdą z nich. Tłumaczenie następuje każdorazowo przy uruchomieniu programu (str. 25).

J

Język programowania
Specjalny język służący do pisania programów komputerowych. Jest on zbiorem określonych instrukcji i zasad składni (str. 18).

K

Klucz
Informacja służąca do szyfrowania lub deszyfrowania tekstu (str. 193).

Kod maszynowy
(program maszynowy, kod wynikowy, program wynikowy)
Program napisany w języku wewnętrznym (maszynowym), wykonywany przez procesor (str. 24).

Kompilacja
Przetłumaczenie programu w całości na język zrozumiały dla procesora, tak by mógł go wykonać komputer. Jeśli kompilacja przebiegnie poprawnie, można uruchomić program. Raz skompilowany program nie wymaga już powtórnej operacji tłumaczenia (str. 25).

Komórka pamięci
Część pamięci operacyjnej komputera (pamięci RAM). Pamięć operacyjna jest podzielona logicznie na komórki pamięci. Każda komórka pamięci ma swój unikatowy adres. W komórkach pamięci mogą być przechowywane dane reprezentujące instrukcje procesora lub dane dla tych instrukcji (str. 40).

L

Liczba pierwsza
Liczba naturalna większa od 1, która ma dokładnie dwa dzielniki naturalne: jedynkę i siebie samą, np.: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Liczba 1 nie jest liczbą pierwszą (str. 220).

Lista kroków
Przedstawienie algorytmu w kolejnych punktach (krokach). Każdy punkt takiej listy zawiera opis wykonywanej czynności. Kolejność punktów nie jest przypadkowa – musi być zgodna z działaniem algorytmu (str. 27).

N

NWD
(ang. GCD – *Greatest Common Divisor*)
Największy wspólny dzielnik dwóch lub więcej liczb naturalnych (różnych od zera) to największa liczba naturalna, przez którą dzieli bez reszty każda z tych liczb (str. 221).

NWW
Najmniejsza wspólna wielokrotność dwóch lub więcej liczb całkowitych jest to najmniejsza liczba różna od zera, która jest jednocześnie wielokrotnością obu liczb (str. 226).

P

Parametr
Wartość przekazywana do funkcji (str. 80 i str. 154).

Pin w elektronice
Styk, przez który wprowadzamy sygnały do urządzenia lub z niego wyprowadzamy (s. 295).

Podprogram
(procedura lub funkcja)
Wyodrębniona część programu, mająca unikatową nazwę i ustalony sposób wymiany danych z innymi częściami programu (str. 79 i 153).

Preprocesor
Program, który przetwarza wstępnie kod źródłowy przed rozpoczęciem zasadniczej kompilacji. Polecenia preprocesora (zwane dyrektywami) rozpoczynają się od znaku #. Np. dyrektywa `#include` nakazuje dołączenie do kodu źródłowego kodu z innego pliku źródłowego (str. 35).

Program komputerowy
(program źródłowy,
kod źródłowy)
Ciąg instrukcji języka programowania, realizujący algorytm (str. 17).

S

Schemat blokowy
Przedstawienie poszczególnych operacji algorytmu za pomocą odpowiednio połączonych figur (bloków). Połączenia określają, w jakiej kolejności i w jaki sposób będą wykonywane operacje realizujące dany algorytm (str. 27) .

Słowo kluczowe
Słowo mające szczególne znaczenie w danym języku programowania, np. oznaczające określony rozkaz, instrukcję lub polecenie (str. 23).

Specyfikacja zadania
(problemu)
Szczegółowy opis zadania, w którym określa się dane wejściowe i oczekiwane wyniki oraz związek między danymi a wynikami (warunki, jakie muszą spełniać wyniki) (str. 19).

Stos w informatyce
Struktura przechowywania danych, w której dostępny jest tylko element znajdujący się na wierzchołku stosu (tak jak w stosie książek na biurku – dostępna jest książka położona na wierzchu jako ostatnia) (str. 215).

Szyfr
Algorytm szyfrowania i deszyfrowania (str. 191).

Szyfrowanie
Proces przetwarzania wiadomości na szyfrogram, pozwalający na ukrycie jej tekstu (str. 192).

System liczbowy
Sposób zapisywania i odczytywania liczb (str. 6).

T

Typ zmiennej
Określa rodzaj danych przechowywanych w zmiennej – mogą to być np. liczby, znaki, wartości logiczne (typy proste) lub teksty, tablice, rekordy i obiekty (typy złożone) (str. 40).

W

Wartość zwracana
Wartość przekazywana z funkcji do miejsca wywołania (np. programu głównego) (str. 80 i str. 154).

Wzorzec
Spójny ciąg znaków (str. 187).

Z

Zintegrowane środowisko programistyczne
(IDE – z ang. *Integrated Development Environment*)
Aplikacja lub zespół aplikacji do tworzenia, modyfikowania, testowania i utrzymywania oprogramowania (str. 35 i str. 109).

#2

Algorytmika i programowanie w praktyce

Wiedzę i umiejętności zdobyte w części 2. możesz wykorzystać w wielu sytuacjach.



■ Stosowanie szyfru do przesyłania wiadomości

- podstawowe metody szyfrowania (tematy 80. i 81.)
- programowanie algorytmu szyfrowania tekstu (tematy 90. i 91.)

* Wiadomość zaszyfrowana szyfrem Cezara. Spróbuj ją odszyfrować.



■ Wykonanie systemu magazynowego sklepu internetowego

- przyjmowanie dostaw – funkcje zwracające wartość (tematy 61. i 73.)
- zarządzanie produktami w magazynie – wykonywanie operacji na elementach tablicy lub listy (tematy 64. i 76.)
- pakowanie paczek do wysyłki – metoda zachłanna (temat 84.)



■ Pisanie prostego programu do nauki języka obcego

- poznanie metod porównywania tekstów – wybrane algorytmy na tekstach i wyszukiwanie wzorca w tekście (tematy 78. i 79.)
- napisanie programu sprawdzającego pisownię słówek – programowanie algorytmów na tekstach w językach C++ i Python (tematy 88. i 89.)



■ Budowanie prostego robota

- wprowadzenie do robotyki (temat 99.)
- poznanie układów Arduino (temat 99.)
- zaprojektowanie robota omijającego przeszkody (temat 102.)

ISBN 978-83-964794-1-9