Temat C5. Stosowanie instrukcji iteracyjnych w językach C++ i Python

Warto powtórzyć

1. Czym charakteryzuje się algorytm z warunkami?
2. Jak działa instrukcja warunkowa w wybranym języku programowania?
3. Kiedy program jest poprawny?
4. Czym jest blok instrukcji (blok kodu) i jak go wyróżniamy?

1. Iteracja – podstawowa technika algorytmiczna

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.

2. 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 1.

Przykład 1. 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 1. 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 1. dodaliśmy siedem liczb. Zazwyczaj jednak zapisujemy algorytm w uogólnionej postaci – dla dowolnej liczby danych. Taki algorytm jest uniwersalny.

Przykład 2. Zapisujemy algorytm sumowania *n* liczb w postaci listy kroków

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: *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 2. wartość sumy liczb dla kolejno wprowadzanych danych jest zapamiętywana w zmiennej `suma`, której na początku algorytmu przypisujemy wartość zero.

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 `suma` zwiększoną o wartość kolejnej liczby `a`". 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 podstawienia:

C++

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

Python

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

w zmiennej `suma` będzie pamiętana wartość 27.

Przykład 3. 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 1. Analizujemy działanie algorytmu dodawania liczb

1. Korzystając z przykładów 2. i 3., 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.

3. Algorytmy iteracyjne w językach C++ i 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 wybranym języku programowania?

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

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

W języku C++ są dostępne trzy instrukcje iteracyjne. Pokażemy zastosowanie jednej z nich – instrukcji **`for`**.

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

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 `{}`.

W języku C++ do określania kroków iteracji (powtórzeń) w instrukcji **`for`** stosujemy zwykle tzw. **zmienną sterującą**.

Przykład 4. Stosowanie instrukcji for w języku C++

W języku C++ 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`.

```
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 zostanie wyświetlonych 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 `n + 1` razy w kolejnych wierszach 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 będzie czekał na wprowadzenie wartości zmiennej `liczba`. Te czynności zostaną powtórzone `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`.

Ćwiczenie 2. Stosujemy instrukcję **for** w języku C++

Korzystając z przykładu 4., napisz cztery programy:

a. Program wyświetlający piętnaście razy znak „@” w tym samym wierszu.
b. Program wyświetlający na ekranie w kolejnych wierszach *n* napisów „Lubię informatykę” (wartość zmiennej *n* wprowadzaj z klawiatury).
c. Program wprowadzający *k* liczb rzeczywistych z klawiatury i zapamiętujący je kolejno w zmiennej *liczba*.
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.

3.2. Instrukcja iteracyjna `for` w języku Python

W języku Python są dostępne dwie instrukcje iteracyjne. Pokażemy zastosowanie jednej z nich – instrukcji **for**.

Instrukcja iteracyjna **for**	**for** *zmienna* **in** *lista_wartości:* *lista_instrukcji*	Python

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 5. 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.

Przykład 6. Stosowanie funkcji `range()` do określania *listy_wartości* w instrukcji **`for`**

Argumentami funkcji `range()` mogą być konkretne wartości lub zmienne, np. `range(10)`, `range(n)`. Wartość zmiennej `n` możemy wprowadzać z klawiatury.

- 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 3. Stosujemy instrukcję `for` w języku Python

Korzystając z przykładów 5. i 6., 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.

3.3. Zapisywanie rozwiązania problemu iteracyjnego w językach C++ i Python

Przykład 7. Zapisywanie algorytmu iteracyjnego w językach C++ i Python

Algorytm sumowania *n* liczb całkowitych wprowadzanych z klawiatury na podstawie specyfikacji i listy kroków podanej w przykładzie 2.

Uwaga: W językach C++ i Python instrukcję przypisania `suma = suma + a` można zapisać krócej: `suma += a` (tabela 3., temat C4).

C++

[*] 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;
}
```

Python

Suma_n.py - C:/Python/Suma_n.py (3.7.2)

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 = suma + a
print("Suma wynosi:", suma)


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

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

Tabela 1. *Operatory inkrementacji i dekrementacji w językach C++ i Python*

Ćwiczenie 4. Zapisujemy algorytm sumowania *n* liczb w języku programowania.

1. Przepisz wybrany program z przykładu 7. Zapisz program w pliku pod nazwą *Suma_n*.
2. Uruchom i przetestuj program dla kilku różnych wartości zmiennych. Wyjaśnij, co się dzieje w poszczególnych wierszach programu.
3. Dodaj do programu sprawdzanie poprawności wprowadzanej wartości zmiennej *n*, zgodnie ze specyfikacją zadania (podaną w przykładzie 2.). Sumowanie wprowadzanych liczb powinno być wykonywane dla poprawnej wartości *n*. Jeśli użytkownik wprowadzi błędne dane, powinien wyświetlić się komunikat „Niepoprawne dane”. Zapisz plik pod tą samą nazwą i uruchom program dla różnych wartości zmiennej *n*.

Ćwiczenie 5. Stosujemy instrukcję iteracyjną **`for`**

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

4. Instrukcje iteracyjne zagnieżdżone

Instrukcje iteracyjne **for** mogą być zagnieżdżone, czyli instrukcją powtarzaną w pętli może być kolejna instrukcja pętli (przykład 8.). Liczba kroków tej iteracji jest określona przez iloczyn: $n \cdot m$, gdzie n i m to liczby powtórzeń odpowiednio w każdej pętli.

Przykład 8. Pętla w pętli w językach C++ i Python

Zadanie: Zapisz w wybranym języku programowania algorytm, który umożliwi wyświetlenie na ekranie monitora „prostokąta” utworzonego ze znaków „x” o bokach *n*, *m* (*m* – liczba znaków „x” w poziomie, *n* – liczba znaków „x” w pionie). Wnętrze prostokąta ma być również wypełnione znakami „x”.

Python 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 = ""`).

C++

[*] Prostokat.cpp

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

int main()
{
    int i, j, m, n;
    cout << "Podaj wymiary prostokata: ";
    cin >> m >> n;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
            cout << "x";
        cout << endl;
    }
    return 0;
}
```

Python

Prastokat.py - C:\Python\Prastokat.py (3.7.2)

File Edit Format Run Options Window Help

```python
m = int(input("Podaj pierwszy wymiar prostokata: "))
n = int(input("Podaj pierwszy drugi prostokata: "))

for i in range(n):
    for j in range(m):
        print("x", end = "")
    print()
```

Ćwiczenie 6. Stosujemy zagnieżdżone instrukcje iteracyjne `for`

1. Napisz specyfikację zadania z przykładu 8.
2. Przepisz wybrany program z przykładu 8. Zapisz program w pliku pod nazwą *Prostokat*, uruchom i przetestuj go dla kilku różnych wartości zmiennych *m* i *n*. Objaśnij, co się dzieje w poszczególnych wierszach programu, m.in. wskaż w programie miejsce (instrukcje), gdzie powstaje „wiersz prostokąta”.

W języku C++ należy stosować odrębną nazwę zmiennej sterującej określającą liczbę iteracji dla pętli zewnętrznej i wewnętrznej.

Ćwiczenie 7. Sprawdzamy działanie pętli zagnieżdżonych dla takich samych zmiennych określających liczbę iteracji

1. Otwórz plik *Prostokat* zapisany w ćwiczeniu 6.
2. Sprawdź, co się stanie, jeśli zamiast zmiennej *j* w wewnętrznej pętli użyjesz również zmiennej *i*. Zmodyfikuj program i uruchom go. Co zauważasz?

Warto zapamiętać

- Jeśli w programie w instrukcji przypisania nadamy zmiennej nową wartość, to poprzednia zostanie usunięta.
- 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 pewne kroki algorytmu zostaną wykonane wiele razy.
- 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).
- W języku Python liczbę iteracji określa długość *listy_wartości* po słowie **`in`** w instrukcji **`for`**.
- Zapętlenie programu nastąpi wtedy, gdy nie określimy prawidłowo warunku zakończenia algorytmu.
- 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. Na czym polega iteracja?
2. Kiedy mamy do czynienia z pętlą?
3. Omów zastosowanie iteracji na przykładzie sumowania ośmiu liczb.
4. Wyjaśnij na przykładach stosowanie instrukcji iteracyjnej **`for`** w wybranym języku programowania.
5. Do czego służy zmienna sterująca w instrukcji **`for`** w języku C++?
6. Co określa liczbę iteracji w instrukcji **`for`** w języku Python?
7. Przedstaw, na czym polega zagnieżdżanie się instrukcji pętli.
8. Na podstawie następujących fragmentów programów napisz, jakie są wyniki ich działania:

a. C++
```
for (i = 0; i <= 9; i++ )
    cout << "$";
```
Python
```
for i in range(10):
    print("$", end = "")
```

b. C++
```
for (i = 0; i <= 9; i++ )
    cout << "$" << endl;
```
Python
```
for i in range(10):
    print("$")
```

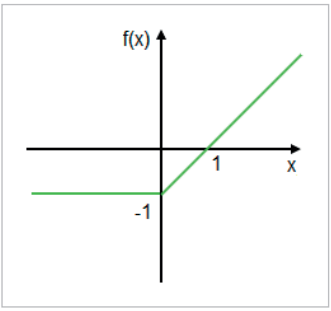
9. C++: Ile razy będzie wykona instrukcja `cout << i`? Dla jakich wartości zmiennej `i` zostanie wykonana instrukcja `cout << i`?
```
for (i = 0; i <= 20; i++)
    cout << i;
```
Python: Ile razy będzie wykona instrukcja `print(i)`? Dla jakich wartości zmiennej `i` zostanie wykonana instrukcja `print(i)`?
```
for i in range(9, 51):
    print(i)
```

Zadania

Uwagi:
- Staraj się dodawać do programów odpowiednie komunikaty dla użytkownika.
- Pisz przejrzyste programy oraz dodawaj w odpowiednich miejscach komentarze.
- Każdy program uruchom i przetestuj dla różnych danych, nawet jeśli w zadaniu nie ma takiego polecenia (w przypadku języka C++ najpierw skompiluj program).

1. Zmodyfikuj program *Przepis* zapisany w zadaniu 9. z tematu C4 tak, aby obliczenia wykonywane były dla każdego z *n* składników sałatki, gdzie *n* jest wprowadzane z klawiatury. Zapisz plik pod tą samą nazwą.
2. Zmodyfikuj program utworzony w zadaniu 2. z tematu C4. Program powinien dodatkowo zliczać, ile wprowadzono liczb parzystych i na koniec wyprowadzać ich liczbę. Zapisz plik pod tą samą nazwą.
3. Napisz specyfikację zadania, utwórz listę kroków oraz napisz w wybranym języku programowania 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 iloczynu 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 i program do obliczania osobno sumy liczb dodatnich i liczb ujemnych dla *n* liczb całkowitych wprowadzanych z klawiatury. Zapisz program w pliku pod nazwą *Dodatnie_ i_ ujemne*.
6. Zmodyfikuj program zapisany w zadaniu 5. Jeśli zostanie wprowadzone z klawiatury zero, wyświetlaj komunikat „Niepoprawne dane”. Zapisz plik pod tą samą nazwą.
7. Na podstawie specyfikacji utworzonej w zadaniu 1b z tematu C1 napisz program sprawdzający, czy wśród *n* znaków wprowadzonych z klawiatury (pamiętanych w zmiennej *znak*) dany znak jest samogłoską („a”, „e”, „i”, „o”, „u”, „y”) czy spółgłoską (pozostałe). Zależnie od wyniku, wyprowadzaj napisy: „samogłoska” lub „spółgłoska”. Zapisz program w pliku pod nazwą *Litery*.
   Wskazówki:
   - W instrukcji warunkowej należy zastosować złożony warunek z alternatywą.
   - W języku C++ zmienną *znak* zadeklaruj jako typ znakowy: `char znak;`.
     W języku Python wystarczy użyć przypisania
     `znak = input("Wprowadź znak: ")`.
8. W firmie X miesięczna płaca podstawowa jest zwiększana m.in. o kwotę za przepracowane nadgodziny. Jeśli liczba nadgodzin przekroczy 30, to stawka za każdą kolejną nadgodzinę 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*.
9. Napisz program obliczający wartości funkcji przedstawionej na wykresie (rys. 1.). Zapisz program w pliku pod nazwą *Wykres_funkcji*.

10. Napisz program, który umożliwi wyprowadzenie na ekran monitora „choinki” (rys. 2.) składającej się z gwiazdek „*”. Wymiary choinki, czyli liczba gwiazdek składających się na podstawę i wysokość choinki, wprowadzane są z klawiatury. Zapisz program w pliku pod nazwą *Choinka*.



Rys. 1. *Wykres funkcji – zadanie 9.*

Rys. 2. *Układ gwiazdek – wynik działania programu dla podstawy 9 i wysokości 9 gwiazdek – zadanie 10.*

11. Napisz program realizujący algorytm, który umożliwi wyprowadzenie na ekran monitora prostokąta o bokach *n* x *m*, składającego się z liter „A” i „B”, występujących co drugi wiersz (*m* – liczba znaków „A” lub „ B” w poziomie, *n* – liczba znaków „A” lub „B” w pionie). Zapisz program w pliku pod nazwą *AB*.

Dla zainteresowanych

12. Iloczyn *n* kolejnych liczb naturalnych nazywamy silnią: $n! = 1 \cdot 2 \cdot \ldots \cdot n$. Napisz program obliczający silnię liczby *n* wprowadzanej z klawiatury, gdzie $n \leq 20$.
13. Zmodyfikuj program utworzony w zadaniu 10. z tematu C4 tak, aby warunek trójkąta był sprawdzany dla *n* trójek liczb wprowadzanych z klawiatury. Zapisz plik pod tą samą nazwą.
14. Zmodyfikuj program utworzony w zadaniu 11. z tematu C4 tak, aby warunek dla trójkąta prostokątnego był sprawdzany dla *n* trójek liczb wprowadzanych z klawiatury. Zapisz plik pod tą samą nazwą.
15. Napisz program obliczający pierwiastki równania kwadratowego $ax^2 + bx + c = 0$. Zapisz program w pliku pod nazwą *Rownanie_kwadratowe*.
16. Zmodyfikuj program z zadania 15., aby poprawnie reagował na błędne dane ($a = 0$), oraz rozróżniał sytuacje: $\Delta = 0$ i $\Delta > 0$. Zapisz plik pod tą samą nazwą.
17. Napisz program wypisujący na ekranie tabliczkę mnożenia liczb naturalnych od 1 do 10. Zapisz program w pliku pod nazwą *Tabliczka*.
18. 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*.
19. Napisz program realizujący algorytm, który umożliwi wyprowadzenie na ekran monitora „szachownicy” składającej się z zer i jedynek. Zapisz program w pliku pod nazwą *Szachownica*.
20. Napisz program tworzący na ekranie obraz składający się z gwiazdek lub innych znaków – według własnego pomysłu.
21. Zaproponuj własne zadanie, w którym zastosujesz poznane techniki: warunki i iteracje. Zapisz rozwiązania w wybranym języku programowania.