Język C++ podstawy

W języku C++ wykonanie programu zaczyna się od funkcji o nazwie `main`, która na ogół zdefiniowana jest następująco:

```
#include<iostream> //plik nagłówkowy z funkcjami wejścia i wyjścia
using namespace std; // będziemy korzystać z funkcji w standardowej (std) przestrzeni nazw
int main() //początek funkcji main i programu
{
// treść funkcji main (programu)
return 0; // gdy program kończy się poprawnie, zwracamy wartość 0
} // koniec funkcji main
```

Program w języku C++ składa się z ciągu instrukcji. Każda instrukcja kończy się znakiem średnika. Należy pamiętać, aby funkcja `main` zawsze zwracała wartość 0 w wypadku poprawnego zakończenia działania programu. Instrukcje można grupować w jedną całość zwaną blokiem, posługując się nawiasami klamrowymi `{}`. Komentarze rozpoczynamy od znaków `//` lub umieszczamy je pomiędzy znakiem `/*` a znakiem `*/`.

*

Kody źródłowe programu są standardowo zapisywane w plikach z rozszerzeniem `.cpp`. Aby nasz program mógł być wykonany, jego kod źródłowy, musi być przekształcony do ciągu instrukcji procesora. Proces ten nazywa się budowaniem i składa się z dwóch etapów: kompilacja a następnie linkowanie. Kompilacja polega na przetłumaczeniu kodu źródłowego na ciąg instrukcji procesora. Zanim jednak kod źródłowy zostanie poddany kompilacji, najpierw wykonywany jest tak zwany preprocesor, który wykonując swoje dyrektywy dołącza do naszego kodu źródłowego tak zwane pliki nagłówkowe zawierające definicje funkcji, które wykorzystujemy w naszym programie. Dyrektywy preprocesora znajdują się na ogół na samym początku programu i zaczynają się od znaku `#`. W wyniku kompilacji powstaje plik binarny z instrukcjami procesora i mający rozszerzenie `.obj` w Windows lub `.o` w Linux. Na koniec procesu budowania wykonywane jest linkowanie polegające na dołączeniu do naszych plików binarnych `.obj` innych wcześniej skompilowane programów (biblioteki) w wyniku czego powstaje finalny plik wykonywalny (`.exe`).

*

Zmienna - to obiekt w programowaniu, który przechowuje różnego rodzaju dane niezbędne do jego działania. W czasie wykonania programu zmienna może zmieniać swoją wartość. Tworząc zmienną, musimy nadać jej nazwę oraz typ, który określa, co nasza zmienna będzie przechowywać. Najważniejsze typy zmiennych w C++ to: `bool` (prawda - 1, fałsz - 0), `char` (-128 do 127), `int` (-2^{31} do $2^{31}-1$), `unsigned int` (0 do $2^{32}-1$), `size_t` (0 do $2^{32}-1$), `unsigned long long` (0 do $2^{64}-1$) oraz typy zmiennoprzecinkowe pojedynczej precyzji `float` oraz podwójnej precyzji `double`. Na przykład deklaracja zmiennej typu całkowitego będzie mieć następującą postać: `int age;`. Aby dowiedzieć się, ile bajtów zajmuje w pamięci dana zmienna lub typ, możemy posłużyć się operatorem `sizeof`. Na przykład instrukcja `sizeof(unsigned int)` zwróci ilość bajtów zajmowaną przez pojedynczą zmienną typu `unsigned int`. Programując w języku C++, wykorzystujemy na ogół zmienne lokalne. Zmienne tego typu widoczne są tylko w bloku, w którym zostały zdefiniowane. Zmienne globalne widoczne są w każdym miejscu programu. Ich użycie pogarsza czytelność programu, dlatego należy je stosować tylko w razie konieczności.

*

W języku C++ istnieje pojęcie literału, czyli sposobu zapisu w kodzie źródłowym liczb, znaków oraz napisów (ciągów znaków). Liczby całkowite możemy zapisać w kodzie źródłowym następująco: `y=10;` (system dziesiętny), `y=0xAB;` (system szesnastkowy). Opcjonalnie można dodać na koniec literału następujące znaki: ‘U’ oznaczający liczbę bez znaku, ‘L’ liczbę typu long, ‘LL’ liczbę typu long long. Na przykład: `unsigned long long x=5ULL;`. Liczby zmiennoprzecinkowe zapisujemy ze znakiem ‘.’ oddzielającym część całkowitą od dziesiętnej, na przykład: `double x=0.314;` lub w notacji naukowej z użyciem potęgi liczby 10, na przykład: `double x=3.14e-1;`.

*

Gdy mamy w programie zdefiniowane zmienne, to możemy wykonywać na nich różne operacje za pomocą tak zwanych operatorów. Najważniejsze operatory w języku C++ to: operator przypisania (`=`), na przykład wyrażenie `x = 5;` przypisuje zmiennej `x` wartość 5. Operatory arytmetyczne (`+`, `-`, `*`, `/`). Operator (`/`) realizuje dzielenie całkowite lub rzeczywiste. Jeśli argumentami są liczby całkowite, operator będzie wykonywał dzielenie całkowite, natomiast dla liczb rzeczywistych operator wykona dzielenie rzeczywiste. Operator (`%`) zwraca resztę z dzielenia dwóch liczb całkowitych, na przykład `int a = 5; int b = a % 3;`. Po wykonaniu tego kodu wartość zmiennej

`b` wyniesie 2. Operatory logiczne: lub logiczne (`||`), i logiczne (`&&`), zaprzeczenie (`!`). Operatory porównania: (`==`, `>`, `<`, `>=`, `<=`) oraz operator różne (`!=`). Ważne jest, by zwrócić uwagę na fakt, że porównanie w języku C++ realizujemy za pomocą operatora `==`, a nie operatora `=`. Na przykład, jeśli chcemy sprawdzić, czy zmienna `a` równa się zmiennej `b`, to piszemy `a == b`, a nie `a = b`. Operatory inkrementacji (`++`) i dekrementacji (`--`), czyli zwiększające lub zmniejszające wartość zmiennej o jeden. Na przykład: `++x;` zwiększa wartość zmiennej `x` o jeden. Niektóre operatory można łączyć z sobą w skrócony zapis. Na przykład zwiększenie wartości zmiennej `a` o pięć `a = a + 5;` można zapisać jako `a += 5;`.

*

Niekiedy chcemy aby nasza zmienna nie mogła być modyfikowana w programie. Na przykład, chcemy mieć w programie wartość, która reprezentuje jakąś stałą matematyczną lub fizyczną albo wyraża inną wartość, która ma być stała podczas wykonania programu. W takim przypadku aby poinformować kompilator, że wartość zmiennej nie może ulec modyfikacjom należy jej deklarację poprzedzić słowem kluczowym `const`, na przykład: `const double pi=3.14159;`. Próba modyfikacji takiej stałej spowoduje błąd kompilacji.

*

Aby w języku C++ wypisać na ekranie informacje, można posłużyć się tak zwanym standardowym strumieniem wyjściowym `cout`. Przykłady:

```
int a = 5;
cout << a; // wyświetl na ekranie wartość zmiennej a
cout<<"przykladowy tekst" << "\t" << endl; // wyświetl tekst, następnie znak tabulacji i zakończ przejściem do nowego wiersza (endl).
```

Aby wczytać wartość z klawiatury, można posłużyć się tak zwanym standardowym strumieniem wejściowym `cin`.

```
int a; char c; string tekst;
cin>>a; cin>>c; cin>>tekst; // wczytaj z klawiatury liczbę całkowitą, pojedynczy znak a następnie tekst
```

*

Bardzo często wykorzystywaną w języku C++ instrukcją jest tak zwana pętla `for`. Ma ona na ogół następującą postać:

```
for (stany_poczatkowe; warunek_końcowy; zmiany){ lista_instrukcji }
```

Najpierw wykonywany jest blok stany początkowe, następnie sprawdzany jest warunek końcowy, potem wykonywana jest lista instrukcji, a następnie blok zmiany i na koniec powtórnie sprawdzany jest warunek końcowy. Jeśli warunek jest spełniony, to ponownie wykonywana jest lista instrukcji, potem blok zmiany i sprawdzany jest warunek, i tak aż do momentu, w którym warunek nie będzie już spełniony. Na przykład pętla `for` wyświetlająca kolejne liczby całkowite od 0 do 9 będzie mieć postać:

```
for ( int i = 0; i < 10 ; i++) { cout << i; }
```

Wyrażenie `i++` powoduje zwiększenie zmiennej `i` o jeden. Słowa kluczowe `continue` oraz `break` umożliwiają nam zmianę standardowego zachowania działania pętli. Słowo kluczowe `break` przerywa aktualnie wykonywaną pętlę. Innymi słowy: jeżeli w pętli dowolnego typu zostanie wykonana instrukcja `break`, to pętla w której wykona się ta instrukcja zostanie natychmiast zakończona. Kolejną instrukcją, będzie ta, która znajduje się bezpośrednio poza przerwaną pętlą. Słowo kluczowe `continue` wykorzystuje się wtedy, gdy chcemy pominąć wykonywanie kodu znajdującego się w pętli - od miejsca wystąpienia słowa kluczowego `continue` aż do samego końca pętli. Mówiąc inaczej: jeżeli w danym kroku pętli zostanie wywołane słowo kluczowe `continue`, to krok aktualnie wykonywany zostanie zakończony, a następnie pętla przejdzie do wykonywania kolejnej iteracji.

*

Funkcja to fragment kodu wykonujący ściśle określone zadanie. Podaje się jej dane, na których ma działać, a w rezultacie otrzymuje się wynik przetwarzania tych danych. Dane wejściowe funkcji nazywają się argumentami, natomiast to, co funkcja zwraca, nazywa się wartością zwrotną funkcji. Każda funkcja ma swoją nazwę. Dzielenie programu na funkcje pozwala zapanować nad jego organizacją. Konstrukcja wszystkich funkcji jest taka sama: pierwszy składnik definicji funkcji to typ zwracany. Określa on, jakiego typu zmienną zwraca funkcja. Na przykład, jeśli funkcja ma zwracać tekst, to będzie to `string`. Jeśli funkcja nic nie zwraca, to jako typ zwracany podajemy

`void`. Drugi składnik to nazwa funkcji. Ważne jest, aby nazwa funkcji odzwierciedlała jej przeznaczenie, podobnie jak to jest w wypadku zmiennych, gdyż znacznie poprawia to czytelność kodu. Następnie, w nawiasach znajduje się lista argumentów funkcji oddzielonych znakiem przecinka. Jeśli funkcja nie pobiera żadnych argumentów, to możemy w nawiasach okrągłych napisać słowo kluczowe `void`. Ostatnim składnikiem funkcji są klamry zawierające jej instrukcje. W klamrach tych znajdują się wszystkie instrukcje, które odpowiadają za działanie funkcji. Można utworzyć kilka funkcji o tej samej nazwie. W takim wypadku każda z nich musi mieć inną listę argumentów. Zabieg taki nazywa się przeciążaniem funkcji. Do zwracania wartości z funkcji służy instrukcja `return`.

```
typ zwracany nazwa_funkcji ( argumenty )
{
    // instrukcje wykonywane przez funkcję
}
// na przykład
int sum(int x, int y) { return x + y; }
```

W szczególnym przypadku funkcja może nie posiadać nazwy. Takie funkcje nazywamy funkcjami lambda. Mają one następującą postać:

```
[] (lista argumentów)->typ_zwracany { // instrukcje wykonywane przez funkcję }
// na przykład
[] (int x, int y)->int { return x + y; }
```

Domyślnie wszystkie zmienne utworzone wewnątrz funkcji przestają istnieć w pamięci komputera w momencie zakończenia działania tej funkcji. Niekiedy jednak, przydatne jest posiadanie zmiennej, która istnieje i zachowuje swoją wartość pomiędzy kolejnymi wywołaniami funkcji. Taka zmienna nazywa się statyczną i deklarujemy ją za pomocą słowa kluczowego `static`, na przykład: `static int x;`. Przy pierwszym wejściu do funkcji zmiennej statycznej domyślnie przypisywana jest wartość 0.

*

Każdy program musi umieć podejmować decyzje. Aby mu to umożliwić, programiści używają tak zwanych instrukcji sterujących (warunkowych). Najczęściej posługujemy się instrukcją `if … else`, która ma postać:

```
if ( warunek_1) { // instrukcje_1 }
else if ( warunek_2) { //instrukcje_2 }
else { // instrukcje_3 }
```

Jeśli będzie spełniony `warunek_1`, to wykonają się `instrukcje_1`. Jeśli `warunek_1` nie będzie spełniony, ale spełniony będzie `warunek_2`, to wykonają się `instrukcje_2`. Jeśli żaden z warunków nie będzie spełniony, to wykonają się `instrukcje_3`. Wyrażenia `else if` oraz `else` są opcjonalne. Należy pamiętać, że w języku C++ każda wartość różna od zera (nawet ujemna) jest traktowana jako logiczna prawda, zaś zero jest traktowane jako logiczny fałsz. Na przykład w wyrażeniu `if (-5) { … }` wartość -5 zostanie potraktowana jako logiczna prawda i wykonają się instrukcje w nawiasach klamrowych. Instrukcję `if … else` można również zapisać w skróconej postaci:

```
warunek ? instrukcja_1 : instrukcja_2;
```

Jeśli będzie spełniony `warunek`, to wykona się `instrukcja_1`, zaś w przeciwnym wypadku wykona się `instrukcja_2`.

*

Pojedynczy znak zapisujemy w języku C++ w apostrofach, na przykład `char x = 'a';`. Instrukcja ta powoduje przypisanie zmiennej `x` wartości odpowiadającej kodowi ASCII litery `a`. Ciąg znaków, zwany również łańcuchem znaków, jest zapisywany w cudzysłowach, na przykład `string x = "napis";`. Dostępne są również znaki specjalne, na przykład znak nowej linii '`\n`' lub `endl`, znak tabulacji '`\t`', powrót kursora do początku wiersza '`\r`', cudzysłów '`\"`', ukośnik wsteczny '`\\`', znak zapytania '`\?`' oraz apostrof '`\'`'.

*

W języku C++ istnieją tak zwane wskaźniki, które można zdefiniować jako adres zmiennej w pamięci komputera. Poniżej przykład deklaracji i użycia wskaźnika do zmiennej typu całkowitego.

```
int x = 5; // deklaracja zmiennej całkowitej
int* ptr = &x; // pobranie wskaźnika do zmiennej x
```

```
*ptr = 7; // modyfikacja wartości zmiennej wskazywanej przez wskaźnik ptr
```

*

Ciąg ułożonych w pamięci jeden za drugim elementów tego samego typu nazywa się tablicą. Dostęp do elementów tablicy odbywa się za pomocą operatora indeksowania `[]`. Pierwszy element w tablicy przechowującej *N* elementów ma indeks 0, drugi 1 i tak dalej aż do *N-1*. Każda tablica ma swoją nazwę, która jest jednocześnie wskaźnikiem do pierwszego elementu w tablicy. Tablice możemy tworzyć statycznie lub, za pomocą operatora `new`, dynamicznie. Gdy tablica utworzona dynamicznie nie jest już potrzebna, należy usunąć ją z pamięci za pomocą operatora `delete[]`. Poniżej przykłady tworzenia i dostępu do elementów tablicy:

```
int tab1[10]; // statyczna tablica 10 liczb typu int
float* tab2 = new float[20]; // dynamiczna tablica 20 liczb typu float
tab1[2] = 5; // przypisanie trzeciemu elementowi tablicy wartości 5
tab2[0] += 2.0; // zwiększenie pierwszego elementu w tablicy o 2.0
delete[] tab2; // usunięcie z pamięci tablicy tab2
```

*

Do generowania liczb losowych służy zadeklarowana w pliku nagłówkowym `cstdlib` funkcja `rand()`. Zwraca ona pseudolosową liczbę całkowitą z przedziału od 0 do wartości `RAND_MAX`, również zdefiniowanej w tym pliku. Aby funkcja `rand()` przy każdym uruchomieniu programu zwracała inne liczby pseudolosowe, przed jej użyciem należy zainicjować generator liczb pseudolosowych za pomocą instrukcji: `srand(time(NULL));`. Funkcja `time()` jest zadeklarowana w pliku nagłówkowym `ctime`. Jeśli chcemy wylosować liczbę całkowitą z przedziału od 0 do 100, to możemy posłużyć się następującym wyrażeniem: `rand()%101;`. Liczby zmiennoprzecinkowe w odstępie 0.01 i przedziale od 0 do 1.0 losujemy następująco: `(rand()%101)/100.0;`.

*

W języku C++ istnieje możliwość wykonania operacji na poszczególnych bitach zmiennej. Jeżeli chcemy zmodyfikować poszczególne bity w zmiennej, to posługujemy się zmiennymi całkowitymi bez znaku (np. `unsigned int`) oraz specjalnymi operatorami. Mamy dostępne następujące operatory działające na poszczególnych bitach: `~` (negacja bitowa), `|` (suma bitowa), `&` (iloczyn bitowy), `^` (alternatywa bitowa), `>>` (przesunięcie bitowe w prawo) oraz `<<` (przesunięcie bitowe w lewo).

```
unsigned int x = 42;       // 00000000000000000000000000101010 (42)
unsigned int y = 57;       // 00000000000000000000000000111001 (57)
unsigned int z = x | y;    // 00000000000000000000000000111011 (59)
unsigned int w = z << 2;   // 00000000000000000000000011101100 (236)
```

*

W wielu wypadkach, aby zapewnić poprawne wykonanie programu konieczne jest sprawdzenie pewnego warunku. W tym celu można posłużyć się makem `assert` zdefiniowanym w pliku nagłówkowym `cassert`. Jeżeli warunek podany jako argument makra nie będzie spełniony to wykonanie programu zostanie przerwane. Można wyłączyć makro `assert` jeśli na początku kodu źródłowego zostanie umieszczona dyrektywa preprocesora `#define NDEBUG`.

```
#include<cassert>
#include<iostream>
int main()
{
  int x, y;
  cin >> x; cin >> y;
  assert(y != 0);  // jeśli y będzie równe 0 to wykonanie programu zostanie przerwane
  cout << "x/y="<< x/y << endl;
  return 0;
}
```

*

Aby poprawnie wyświetlić polskie znaki w konsoli można posłużyć się funkcją `setlocale( LC_ALL, "pl_PL" ); setlocale(LC_ALL, "polish");`

*

Niekiedy istnieje konieczność dokonania konwersji typu zmiennej. Na przykład mamy zmienną typu `float` i chcemy na jej podstawie utworzyć zmienną typu `int`. W takim przypadku posługujemy się funkcjonalnością języka C++ zwaną rzutowaniem typów. Rzutowanie zmiennej typu `float` na `int` będzie mieć następującą postać: `float x=3.14; int y=(int)x;`. Należy pamiętać, że w przypadku tego typu konwersji może wystąpić zaokrąglenie wartości zmiennej w wyniku czego otrzymana wartość może różnić się od początkowej. W tym przykładzie, wartość zmiennej `x` wyniesie 3.

*

Najczęściej używane funkcje matematyczne dostępne są po włączeniu do programu pliku nagłówkowego `cmath`. Dostępne są na przykład funkcje trygonometryczne `sin(double)`, `cos(double)`, `tan(double)`, logarytm naturalny `log(double)`, potęga o dowolnym wykładniku `pow(double, double)`, wartość bezwzględna `fabs(double)`, zaokrąglenie w dół `floor(double)` oraz górę `ceil(double)` oraz wiele innych.

*

Niekiedy zachodzi konieczność zatrzymania działania programu w jakimś miejscu. W tym celu, można posłużyć się instrukcją `cin.get()`, która zatrzymuje działanie programu i oczekuje na wciśnięcie przycisku Enter.

zadania[1]

1. Napisz program rozwiązujący równanie kwadratowe w dziedzinie liczb rzeczywistych o następującym działaniu:
   a. Program wypisuje komunikat: `Podaj współczynnik a:`
   b. Użytkownik wpisuje wartość współczynnika `a`.
   c. Program wypisuje komunikat: `Podaj współczynnik b:`
   d. Użytkownik wpisuje wartość współczynnika `b`.
   e. Program wypisuje komunikat: `Podaj współczynnik c:`
   f. Użytkownik wpisuje wartość współczynnika `c`.
   g. Program wypisuje rozwiązania równania $a \cdot x^2 + b \cdot x + c = 0$.

   Program powinien uwzględniać przypadki szczególne, np. gdy równanie nie ma rozwiązań w dziedzinie liczb rzeczywistych.

2. Napisz program, który za pomocą pętli `for` obliczy wartość procentu składanego. Dla przypomnienia: procent składany to sposób oprocentowania wkładu pieniężnego polegający na tym, że odsetki za dany okres oprocentowania są doliczane do wkładu (podlegają kapitalizacji) i w ten sposób składają się na zysk wypracowywany w okresie następnym. Program powinien poprosić użytkownika o wprowadzenie wartości początkowej kapitału, stopy procentowej oraz liczby kapitalizacji. Na wyjściu powinna zostać podana wartość końcowa kapitału.

3. Napisz program, który losuje liczbę całkowitą z przedziału od 0 do 100, a następnie prosi użytkownika o odgadnięcie wylosowanej liczby i wprowadzenie jej z klawiatury. Następnie program wyświetla komunikat, czy wprowadzona liczba jest większa, czy mniejsza od wylosowanej. Gra kończy się w wypadku wprowadzenia poprawnej liczby lub przekroczenia maksymalnej liczby prób, która wynosi 5.

4. Napisz program, który pobiera z klawiatury liczbę całkowitą będącą liczbą elementów w tablicy liczb zmiennoprzecinkowych typu double. Następnie tworzy dynamiczne tą tablicę i wypełnia ją losowymi liczbami z przedziału od -1.0 do 1.0. Tak utworzoną tablicę przekazuje do funkcji jako wskaźnik. Funkcja ta zwraca wartość średnią i odchylenie standardowe wszystkich elementów tablicy. Dla przypomnienia, odchylenie

[1] Do każdego z zadań najlepiej stworzyć oddzielny projekt w VisualStudio.

standardowe dane jest następującym wzorem: $\sigma = \sqrt{\frac{\sum_{i=0}^{N-1} (x_i - \bar{x})}{N}}$, gdzie *N* to liczba danych, zaś $\bar{x}$ to ich wartość średnia.

5. Posługując się operatorami bitowymi, napisz program, który wypisze na ekranie reprezentację bitową wprowadzonej z klawiatury liczby całkowitej. Przyjmij, że liczba może mieć maksymalnie 64 bity. Dodatkowo, oddziel każdą grupę ośmiu bitów znakiem spacji.

6. Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą *n* (*n* > 2) i wypisuje na standardowym wyjściu największą liczbę *k* taką, że *k* dzieli *n* przy założeniu, że *k* < *n*. Algorytm wyszukiwania liczby *k* spełniającej powyższe warunki umieść w oddzielnej funkcji.

7. Składnia języka C/C++ udostępnia przydatną w niektórych sytuacjach instrukcję wyboru *switch-case*. Używając instrukcji *switch-case*, napisz niewielki kalkulator, który pobiera na wejściu jeden z operatorów arytmetycznych oraz dwa argumenty, po czym wyświetla wynik obliczeń otrzymany na podstawie tych danych. Program powinien zakończyć swoje działanie po wprowadzeniu litery `q` zamiast operatora arytmetycznego.

8. Napisz program wczytujący ze standardowego wejścia dwie dodatnie liczby całkowite *n* i *m*, i wypisujący w kolejnych wierszach na standardowym wyjściu wszystkie dodatnie wielokrotności *n* mniejsze od *m*.

9. Napisz program, który obliczy sumę dwóch dodatnich wartości całkowitych. Suma do obliczenia jest podana jako a ciąg, np. “123 + 37”, “78+ 99” itd. Program powinien działać następująco:
    a. Poproś użytkownika o wprowadzenie ciągu;
    b. Wyeliminuj dodatkowe spacje z ciągu;
    c. Wyodrębnij dwa podciągi reprezentujące dwa argumenty, takie jak „123” i „37”;
    d. Oblicz wartości całkowite tych dwóch podciągów, więc „123” jest przekształcane na wartość 123 itd.
    e. Wydrukuj sumę dwóch wartości;
    f. Rozszerz swoje rozwiązanie o obsługę potencjalnych błędów.