Podstawy

O czym będziemy mówić? Krótka historia C++ (od C zaczynając)

Język C wyłonił się z języka B, który powstał z kolei z języka BCPL (Basic Combined Programming Language). BCPL został stworzony przez Martina Richardsa w roku 1967. Był to język beztypowy, który działał bezpośrednio na słowach maszynowych i adresach. Zainspirowany językiem BCPL Ken Thomspon w roku 1970 stworzył beztypowy język B. Język B i język asemblera zostały użyte do napisania pierwszej wersji systemu UNIX. W roku 1972, Dennis Ritchie zaprojektował język C, w którym zawarł szereg idei pochodzących z języka BCPL, ponadto zostały wprowadzone typy (typy całkowite, rzeczywiste itp.). Komitet ANSI określił w roku 1988 unowocześniony standard języka C. Język zgodny z tym standardem określony jest jako ANSI C.
Język C++ został stworzony przez Bjarne Stroustrup'a i jego zespół w AT&T, jako rozwinięcie C .
Przeznaczenie: symulacje- język miał umożliwić efektywne zorientowane obiektowo programowanie.
Pierwsza wersja C(++) powstała w 1980 roku jako „C with classes", a ostateczna wersja języka wg Stroustrupa - w 1985 roku, już pod nazwą C++.
1989 - utworzenie przez ANSI (American National Standards Institute) komitetu standaryzacyjnego. Cel – unifikacja języka – różnicowanego przez producentów kompilatorów i środowisk programistycznych.
1998 ISO (International Organization for Standarization) zatwierdza standard dla C++ (ISO/IEC 14882).
2011 standard C++11, a w 2017 standard C++17

Pierwsze programy

Najprostszy program w C++ ma postać:

```
int main()
{
        return 0;
}
```

Ogólnie, program w C++ składa się z jednego lub kilku (zapisanych w osobnych plikach) modułów. Dokładnie jeden moduł musi zawierać funkcję o nazwie **main**. Wykonanie programu polega zasadniczo na wykonaniu tej właśnie funkcji. Zwraca ona do powłoki systemu operacyjnego wartość typu **int** (liczba całkowita), co zaznaczamy pisząc nazwę typu **int** przed nazwą funkcji. Przedstawiony powyżej program składa się tylko z definicji funkcji **main**. Ciało funkcji, zawarte pomiędzy nawiasami klamrowymi **{ }**, składa się z jednej instrukcji:

```
  return 0;
```

zakończonej średnikiem. Instrukcja **return** oznacza, że wartość typu **int** zwracana jest do systemu i zwyczajowo powinna ona wynosić `0`, jeśli program kończy się pomyślnie.

Przyjmijmy, że nasz drugi program ma wyświetlić na konsoli tekst: **Hello, World!.**

```
/*  Komentarz wieloliniowy: Naukę każdego języka programowania
     zaczynamy zawsze od HelloWorld!!!!
*/
#include <iostream> //dyrektywa preprocesora dołączająca plik nagłówka iostream
using namespace std; /*udostępnienie wszystkich elementów przestrzeni nazw std
                       za pomocą dyrektywy using */

int main() //definicja funkcji main() - początek
{ // Komentarz jednoliniowy

    cout << "Hello, World!" << endl;

}//definicja funkcji main() - koniec
```

#include <iostream> .
Włączenie do programu zawartości pliku `iostream`. Dzięki temu w programie dostępne będą narzędzia (w postaci najrozmaitszych klas, funkcji, stałych itd.), służące do wykonywania operacji wejścia/wyjścia (w skrócie, operacje we/wy), a więc np. wczytywania z konsoli i wypisywania na ekranie danych. Zauważmy, że instrukcja `#include` powoduje rzeczywiste włączenie pliku: równie dobrze moglibyśmy w tym miejscu wpisać jego treść bezpośrednio do naszego programu. Sam plik `iostream` znajduje się w znanym kompilatorowi katalogu: użycie nawiasów kątowych ('<...>') oznacza właśnie, że nie jest to nasz własny plik, ale plik ze znanego kompilatorowi specjalnego katalogu bibliotecznego, dostarczonego zwykle wraz z kompilatorem przez producenta. Instrukcja '`#include`' nie jest przeznaczona dla kompilatora. Włączenie pliku wykonywane jest przez preprocesor, który zajmuje się wyłącznie przetwarzaniem tekstu naszego programu przed jego właściwą kompilacją - to, co „zobaczy" kompilator, to tekst naszego programu przetworzony przez preprocesor.

```
using namespace std;
```
Linia ta oznacza, że nazw (klas, stałych, funkcji) niezdefiniowanych w naszym programie należy szukać w przestrzeni nazw '`std`'. W tej przestrzeni nazw znajdują się właśnie obiekty dostarczone przez dyrektywę preprocesora '`#include <iostream>`'.
Przestrzeń nazw(**namespace**) jest narzędziem umożliwiającym uniknięcie kolizji nazw zmiennych, stałych, funkcji i obiektów. Do elementów należących do danej przestrzeni nazw możemy się dostać tylko, jeśli poinformujemy kompilator, że dany identyfikator należy do określonej przestrzeni nazw.
Na przykład wszystkie identyfikatory standardowej biblioteki są zawarte w przestrzeni nazw **std**. Możemy się do nich odwoływać na trzy sposoby (przykłady dla obiektu **cout**):

- za pomocą dyrektywy **using**, np:
```
using namespace std;
coutdalszy ciąg instrukcji;
```

- za pomocą kwalifikatora **::**, np.:
```
std::coutdalszy ciąg instrukcji;
```

- za pomocą deklaracji **using**:
```
using std::cout;
coutdalszy ciąg instrukcji;
```

My będziemy się posługiwać pierwszą metodą – dyrektywą **using**.

```
cout << "Hello, World!" << endl;
```
Jest to jedyna instrukcja tego programu. Jak widzimy, instrukcje kończą się średnikiem. Ta konkretna linia powoduje wyprowadzenie na ekran napisu podanego w cudzysłowach.
Korzystamy z obiektu **cout** (console output), który kieruje wszystkie dane wysłane do niego na standardowe wyjście (zwykle jest to konsola, ale istnieje możliwość przekierowania strumienia wyjścia do pliku lub urządzenia peryferyjnego).
Obiekt **cout** jest instancją klasy **ostream** (output stream – „strumień wyjścia"). Obiekt ten jest tworzony automatycznie podczas uruchomienia programu, jeśli dołączymy odpowiedni standardowy plik nagłówkowy przy użyciu dyrektywy preprocesora `#include <iostream>`.

Aby przesłać komunikat `"Hello, World!"` do obiektu **cout**, który wyśle go na konsolę, użyjemy operatora **<<** (operator wstawienia - insertion). Odpowiednia instrukcja ma postać:
```
cout<<"Hello, World!";
```

`"Hello, World!"` -to **literał łańcuchowy lub napisowy.** Są to bezpośrednio zapisane w programie ciągi znaków (napisy, inaczej zwane też łańcuchami znakowymi), które chcemy traktować jako teksty, a nie elementy języka. Ciągi takie ujmujemy w cudzysłów.
Dlaczego musimy zapisywać znaki i ciągi znaków w apostrofach/cudzysłowie?
Otóż nieujęte w cudzysłów napisy i nieujęte w apostrof znaki alfabetyczne traktowane są jako nazwy zmiennych lub słowa kluczowe (oznaczające instrukcje lub inne elementy języka).

W powyższym kodzie pojawiają się dwa nowe elementy :

//	symbol komentarza nakazujący kompilatorowi zignorowanie wszystkich znaków od miejsca jego wystąpienia do końca linii
/*...*/	symbol **/*** nakazuje kompilatorowi zignorować wszystkie znaki aż do sekwencji kończącej komentarz: ***/** - ten rodzaj komentarza może obejmować kilka linii kodu

Jeszcze jedna uwaga – wykonywanie instrukcji zawartych w ciele funkcji (w naszym przypadku funkcji **main**) kończy się w momencie wykonania instrukcji **return**. Oznacza to, że dla ciągu instrukcji
```
int main()          //definicja funkcji main()
{
      return 0;
      cout<<"Jestem Twoim programem";
}
```

instrukcja **`cout<<"Jestem Twoim programem";`** nigdy nie zostanie wykonana, bo po instrukcji **`return 0;`** program zwraca sterowanie do powłoki systemu operacyjnego (BCB:"Warning: unreacheable code").
Okno konsoli zamknęło się natychmiast po zakończeniu wykonywania programu.
Jeśli chcemy wstrzymać wykonywanie programu do momentu wciśnięcia dowolnego klawisza, możemy posłużyć instrukcją:
```
system("pause");
```

Trochę więcej o strumieniu wyjściowym

Przyjmijmy, że chcemy wypisać na konsoli tekst:

```
Napis w pierwszej linii.
Napis w drugiej linii.
```

Aby przejść do nowej linii możemy posłużyć się specjalnym manipulatorem strumieni wyjścia **endl**, wysyłającym znak końca linii do strumienia:

```
cout<< ”Napis w pierwszej linii.”<<endl<<”Napis w drugiej linii.”;          .
```

Jak widać na powyższym listingu do obiektu **cout** możemy wysłać dowolną liczbę wyrażeń poprzedzonych operatorem **<<**.

Równoważny jest zapis:

```
cout<< ”Napis w pierwszej linii.”
      <<endl
      <<”Napis w drugiej linii.”;   //instrukcja może obejmować kilka wierszy!
```

Identyczny efekt uzyskamy korzystając ze znaku specjalnego **\n**:

```
cout<< ”Napis w pierwszej linii.\nNapis w drugiej linii.”;       .
```

Jeśli z jakichś przyczyn (np. dla lepszej przejrzystości kodu) chcielibyśmy umieścić tekst wysyłany do obiektu **cout** w kilku liniach, możemy to zrobić także w następujący sposób:

```
cout<< ”Napis w pierwszej linii.\n”
      ”Napis w drugiej linii.”;
```

Powyższy kod jest poprawny. Preprocesor sam połączy następujące po sobie teksty zawarte w cudzysłowach (określane dalej jako stałe tekstowe lub łańcuchy tekstowe), jeśli są oddzielone spacją, znakiem tabulacji, końcem linii lub pustą linią. Linia może być zawsze przedłużona, jeśli jej ostatnim znakiem (czyli poprzedzającym znak końca linii) jest znak **** :

```
cout<< ”Napis w pierwszej linii.\n\
Napis w drugiej linii.”;
```

Więcej specjalnych sekwencji rozpoczynających się od znaku **** (escape sequences) zawiera tabela:

Tabela 1 Wybrane sekwencje specjalne.

Sekwencja	Efekt
\a	Sygnał dźwiękowy
\b	backspace
\t	Tabulacja pozioma
\n	Przejście do nowego wiersza
\r	Powrót karetki (strumień zaczyna pisać od początku bieżącego wiersza)
\”	Cudzysłów
\’	Apostrof
\?	?
****	Ukośnik \
\0	Znak końca łańcucha tekstowego
\xhh	Kod znaku ASCII (hexadecymalnie)
\888	Kod znaku ASCII (oktalnie)

Do obiektu **cout** możemy wysyłać także pojedyncze znaki:

```
cout<<'A';\\Literał znakowy określa jeden bezpośrednio zapisany w programie znak.
          \\ Do zapisania znaku będziemy stosować apostrofy.
```

oraz liczby:

```
cout<<121;\\Literał liczbowy to bezpośredni zapis konkretnej liczby.
```

Zadania

1. Napisz program BANNER drukujący na ekranie napis:

```
#     #
#  #  #     #     #####    ##         #
#  #  #     #       #     #  #        #
#  #  #     #       #    #    #       #
#  #  #     #       #    ######       #
#  #  #     #       #    #    #  #    #
 ## ##      #       #    #    #   ####
```

2. Napisz program, który wyświetli na konsoli Twoją wizytówkę:

```
*****************************
"                           "
"     'Imię Nazwisko'       "
"      wiek: 00 lat(a)      "
"                           "
*****************************
```

3. Napisz program, który wyświetli na konsoli Twoje inicjały w formacie:

```
*     *     *
*     **    *
*     * *   *
*     *  *  *
*     *   * *
*     *    **
*     *     *
```

4. Popraw błędy w kodzie:

```
*/wytęż wzrok i znajdź ??? szczegółów :-)//
#include <stream>
int main
{
  cout<<”Jeśli’,
  cout>>”program wyświetli”;
  cout<<”endl;”
  cout<<’ten tekst,’
      <<” możesz zrobić sobie przerwę.”<<endl.
  Return 0;
```

5. Co wyświetli na konsoli ten program? Skorzystaj z **Tabela 1 Wybrane sekwencje specjalne.**

```
#include <iostream>

using namespace std;

int main()
{
  cout<<"\n1\n\t2\r3";


  return 0;
}
```

6. Napisz program, który będzie zwracał następującą tabelę (ż == **\xBE)**:

```
Poniższe kursy obowiązują od dnia 2003-10-31:
+----------------+-------+---------------+-------+----------+
|Kraj             |Symbol |Waluta         |Kurs   |Kurs      |
|                 |waluty |               |kupna  |sprzedaży |
+-----------------+-------+---------------+-------+----------+
|Australia        |781    |1 AUD          |2,8111 |2,8679    |
|Czechy           |213    |1 CZK          |0,1446 |0,1476    |
|Dania            |792    |1 DKK          |0,6243 |0,6369    |
|Estonia          |233    |1 EEK          |0,2965 |0,3025    |
|Japonia          |784    |100 JPY        |3,6686 |3,7428    |
|Kanada           |788    |1 CAD          |3,0324 |3,0936    |
+-----------------+-------+---------------+-------+----------+
```

7. Napisz program, który narysuje jeden z obrazków: