C++

Wydanie II

Andrzej Stasiewicz

TABLICE INFORMATYCZNE

Helion

BUDOWA PROGRAMU

Najprostszy program w C++

Standard C++ wymaga nagłówków biblioteki standardowej bez rozszerzenia *.h*, starych nagłówków w wersji z literą c (np. **cmath**) i dołączenia przestrzeni nazw `std`. Funkcja `main()` nie musi się kończyć frazą `return`.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
 double a;
 cin >> a;
 cout << "sinus(" << a << ") = "
 << sin(a);
}
```

Najprostszy program w C++

Starsze kompilatory mogą wymagać następującej treści:

```
#include <iostream.h>
#include <math.h>
int main()
{
 double a;
 cin >> a;
 cout << "sinus(" << a << ") = "
 << sin(a);
 return 0;
}
```

Pliki źródłowe

Treść deklaracji zwyczajowo umieszczamy w plikach o rozszerzeniu *.h* (tzw. nagłówkach). Treść implementacji znajduje się w plikach o rozszerzeniu *.cpp*. W pierwszych liniach plików implementacyjnych *.cpp* zazwyczaj sytuujemy dyrektywy `#include <nazwa nagłówka>` (porównaj podrozdział „Dyrektywy preprocesora"). Proste programy przygotowujemy bez wyodrębniania części *.h*, spisując deklaracje w początkowych fragmentach pliku *.cpp*.

Przykład

Zawartość pliku *nazwa_pliku.h*:

```
#define MAXX 640
const double pi = 3.14;
double srednia(double a, double b);
```

Zawartość pliku *nazwa_pliku.cpp*:

```
#include "nazwa_pliku.h"
using namespace std;
int main()
{
...
}
double srednia(double a, double b)
{
...
}
```

DYREKTYWY PREPROCESORA

Preprocesor przetwarza tekst programu przed kompilacją. Wszystkie dyrektywy preprocesora zaczynają się od znaku `#`.

`#include <nazwa_pliku>`	Wstawia treść pliku (zazwyczaj nagłówkowego) biblioteki
`#include "nazwa_pliku"`	Wstawia treść pliku (zazwyczaj nagłówkowego) użytkownika
`#define WERSJA_1`	Określa napis `WERSJA_1` (do kompilacji warunkowej)
`#ifdef WERSJA_1`	Kompiluje, jeśli napis `WERSJA_1` jest określony
`#ifndef WERSJA_1`	Kompiluje, jeśli napis `WERSJA_1` nie jest określony
`#endif`	Kończy obszar zapoczątkowany przez `#ifdef` albo `#ifndef`
`#undef WERSJA_1`	Odwołuje napis `WERSJA_1`
`#define MAXX 640`	Napisy `MAXX` zastępuje napisem `640`
`#define MAX(a,b) ((a)>(b)?(a):(b))`	Definiuje makropolecenie, tutaj maksimum dwóch liczb

Przestrzenie nazw

Aby zapobiec konfliktom nazw w obrębie tekstu źródłowego (np. podczas pracy zespołowej), wprowadzono słowo kluczowe `namespace`.

`namespace Kowalski {treść programu}`	Zamknięcie fragmentu programu w swojej przestrzeni nazw
`Malinowski :: wydruk();`	Odwołanie do elementu określonego w innej przestrzeni nazw
`using namespace Malinowski;`	Trwałe podłączenie do innej przestrzeni nazw

Wszystkie identyfikatory biblioteki standardowej są zdefiniowane w przestrzeni nazw `std`, stąd w zasadzie każdy współczesny program zaczyna się od deklaracji `using namespace std;`.

Przykład

```
#include <iostream>
using namespace std;
int main()
{
 std::cout << "Jan Kowalski";
}
```

Standardowe wejście i wyjście

W pliku nagłówkowym **iostream** biblioteki standardowej są zadeklarowane klasy oraz operatory realizujące pobieranie danych z klawiatury i wypisywanie danych na ekran.

`cin >> a;`	Pobranie wartości do zmiennej `a` (która musi być zadeklarowana)
`cin >> a >> b >> c;`	Pobranie kaskadowe kilku wartości
`cout << a;`	Wypisanie (wyprowadzenie) wartości zmiennej `a`
`cout << "Wartosc a = " << a;`	Wypisanie kaskadowe kilku wartości
`cout.width(20);`	Ustalenie szerokości pola do wyprowadzenia zmiennej
`cout.fill('*');`	Ustalenie znaku wypełniającego nadmiar szerokości
`cout.precision(2);`	Ustalenie liczby miejsc po przecinku wyprowadzanej zmiennej

Przykład

```
#include <iostream>
using namespace std;
int main()
{
 double a;
 cin >> a;
 cout.width(20);
 cout.precision(2);
 cout << "Liczba = " << a;
}
```

INSTRUKCJE STERUJĄCE

Instrukcja grupująca (blok instrukcji)

```
{
 instrukcja 1;
 instrukcja 2;
 ...
}
```

Zbiór instrukcji ujętych w instrukcji grupującej jest traktowany jak jedna instrukcja. Taka instrukcja umożliwia deklarowanie danych lokalnych, widocznych tylko w jej obrębie. Stosowana jest głównie w warunkach logicznych i pętlach.

Przykład

```
if(a < 0)
{
 cout << "a jest mniejsze od zera
...";
 a = 10;
}
```

Instrukcja wykonania warunkowego if ... else

```
if(warunek logiczny)
{
 instrukcje A;
}
else
{
 instrukcje B;
}
```

Realizuje polecenie: „jeśli warunek jest spełniony — wykonaj instrukcje A, w przeciwnym wypadku — instrukcje B". Części od `else` w dół może nie być. Instrukcje grupujące `{...}` są potrzebne, jeśli mamy wykonać warunkowo więcej instrukcji.

Przykład

```
if(a < 100)
   b = 0;
else
{
   b = 1;
   c = 0;
}
```

Częste błędy

- Umieszczenie średnika za frazą `if(...)` i przed instrukcją grupującą.
- Pominięcie instrukcji grupującej, jeżeli jest niezbędna.

Zwrotnica wielokierunkowa switch() { case ...}

```
switch(wyrażenie_klucz)
{
 case wartosc_1: instrukcje;
                 break;
 case wartosc_2: instrukcje;
                 break;
 ...
 default: instrukcje;
}
```

Dopasowuje wartość klucza do etykietek we frazach `case` i realizuje instrukcje z odpowiedniej szufladki. Sformułowanie `wyrażenie_klucz` musi być typu wyliczeniowego (znak, liczba całkowita). Klamry są obowiązkowe — w tym wypadku nie oznaczają instrukcji grupującej. Wariant `default` jest realizowany wtedy, gdy klucz nie pasuje do etykietki żadnego wariantu `case`. Wariant `default` nie jest konieczny. Szufladki nie muszą być spisywane w jakimś ustalonym porządku.

Przykład

```
switch(a)
{
 case 0:
 case 1:
        cout << "Jan Kowalski";
        break;
 case 17:
        b = 1;
        break;
 default:
        cout << "Niewlasciwa
        wartosc !!!";
}
```

Częste błędy

- Brak w którejś szufladce frazy `break` na zakończenie algorytmu (od razu wykona się następna szufladka).
- Ta sama wartość etykiety kilku szufladek.

Pętla for (...; ...; ...)

```
for(wyrażenie inicjujące; warunek logiczny; wyrażenie modyfikujące)
{
 instrukcje;
}
```

Ma w nagłówku dwa średniki, które wyznaczają trzy pola. Pierwsze pole wykonuje się jednorazowo przy wejściu do pętli — zazwyczaj zawiera instrukcję inicjowania licznika obrotów. Drugie pole wykonuje się przed rozpoczęciem każdego obrotu pętli i zawiera warunek logiczny, warunkujący wykonanie obrotu. Trzecie pole wykonuje się na zakończenie każdego obrotu i zazwyczaj zawiera modyfikację licznika obrotów.

Przykład

```
for(i = 0; i < 100; ++i)
{
   cout << "Obrót pętli nr "
   << i + 1;
}
```

Częsty błąd

- Postawienie średnika tuż za pętlą, a przed instrukcjami, które mają być powtarzane.

Pętla for (... : ...) dla tablic i kontenerów

```
for(zmienna iterująca : tablica)
{
 instrukcje;
}
```

Dostępna w standardzie c++11.

Przykład

```
int tablica[3] = {1,2,3};
for(int &x : tablica)
{
   cout << "Element " << x << endl;
   x = x + 3;
}
```

Pętla while()

```
while(warunek logiczny)
{
   instrukcje;
}
```

Pętla o tym samym charakterze co pętla `for`, jednak bez zaimplementowanych pól inicjowania i kończenia obrotu. Zazwyczaj stosuje się ją tam, gdzie nie wiadomo z góry, ile obrotów zostanie wykonanych.

Przykład

```
i = 0;
while (i < 100)
{
   cout << " Obrót pętli nr "
   << i + 1;
   i = i + 1;
}
```

Porównaj analogiczny przykład dla pętli `for`.

Częste błędy

- Postawienie średnika za nagłówkiem pętli, a przed instrukcjami, które mają być powtarzane.
- Pominięcie klamer instrukcji grupującej, mimo że powtarzanych ma być kilka instrukcji.

Pętla do ... while()

```
do
{
   instrukcje;
}while(warunek logiczny);
```

Pętla sprawdza warunek logiczny *po* wykonaniu instrukcji, zatem zawsze wykona się przynajmniej jeden raz. Dlatego nie może zastępować pętli `for` i `while`, które sprawdzają warunki logiczne *przed* wykonaniem instrukcji.

Przykład

```
do
{
   cin >> c;
   a = a + 1;
} while(c != 'k');
```

Częsty błąd

- Umieszczenie w algorytmie, który wymaga pętli `for` lub `while`.

Instrukcja break

```
break;
```

Przerywa działanie każdej pętli. Zobacz także instrukcję `switch`, w której instrukcja `break` wyznacza koniec algorytmu szufladki `case`.

Przykład

```
while(i < 100)
{
   i = i + 1;
   if(i > 20)
      break;
}
```

Instrukcja continue

```
continue;
```

Przerywa działanie bieżącego obrotu pętli i przechodzi do następnego.

Przykład

```
while(i < 100)
{
   i = i + 1;
   if(i < 20)
      continue;
   cout << " Obrót pętli nr "
   << i + 1;
}
```

Instrukcja „wyrażeniowe if"

```
a = (warunek logiczny) ? wyrażenie_na_tak : wyrażenie_na_nie;
```

Zbliżona charakterem do instrukcji `if ... else`, może być przez nią zastąpiona. Zwraca wartość, zatem można ją wbudować w wyrażenie arytmetyczne. Dwa warianty wyrażeń muszą dostarczać wartości tego samego typu.

Ciąg dalszy na str. 2