Język C++

Różnice między C a C++

Plan wykładu

- C a C++
  - Różnice ogólne
  - Typy
  - Deklaracje zmiennych
- C++ jako rozszerzenie C
  - Domyślne argumenty funkcji
  - Przeciążanie funkcji
  - Referencje
  - Dynamiczny przydział pamięci
  - Strumienie wejścia/wyjścia
- Podsumowanie

C++ jako rozszerzenie języka C

- ulepszony język C
- abstrakcja danych (definiowanie własnych typów - klas)
- programowanie obiektowe (dziedziczenie, polimorfizm)

Różnice ogólne

Nowe słowa kluczowe:

`class, delete, new, friend`

Komentarze

```
/* Komentarz wieloliniowy w C i C++ */
// Komentarz jednoliniowy w C++
```

Wskaźniki typu *void*

W C wskaźniki dowolnego typu mogą być przydzielane do wskaźników typu *void* i odwrotnie.

W C++ wymagana jest jawna konwersja.

```
void *temp;
char *tekst = "Hej!";
temp = (void *) tekst;
tekst = (char*) temp;
```

Typy

Typ znakowy

W C pojedyncze znaki np. ‘x’ ‘\n’ są typu *int*.

W C++ pojedyncze znaki są typu *char*.

Typ wyliczeniowy

W C++ zmiennej danego typu wyliczeniowego można przypisać jedynie stałą należącą do definicji tego typu.

W C++ przy deklaracji zmiennych typu wyliczeniowego można pominąć słowo enum.

W C++ można definiować lokalne typy wyliczeniowe (w obrębie struktur lub klas)

```
enum boolean (FALSE, TRUE};

boolean b = TRUE; // OK
b = 0; // błąd !!!
```

Deklaracje zmiennych

W C++ deklaracja jest typem instrukcji. Można deklarować zmienne w dowolnym miejscu programu.

```
main()
{
    printf("Poczatek\n");
    int temp = 3;
    for (int i = 0; i < 10; i++)
    {
        //...
    }
}
```

Struktury i unie

- W C++ przy deklaracji zmiennych typów strukturalnych można pominąć słowo *struct*.

```
struct punkt {
  int x;
  int y;
};

struct punkt p; /* zmienna w C i C++ */
punkt p; // zmienna w C++
```

- W C++ struktury i unie są klasami

Domyślne argumenty funkcji

```
void fun(int a, int b = 2, int c = 3)
{
    //...
}

main()
{
    fun(1);          // fun(1, 2, 3)
    fun(1,1);        // fun(1, 1, 3)
    fun(1,1,1);      // fun(1, 1, 1)
}
```

Przeciążanie funkcji

Definiowanie zbioru funkcji o tej samej nazwie, różniących się liczbą i/lub typem parametrów

```
void fun(int a) { ... }
void fun(float a) { ... }
```

Uwaga! Przeciążanie funkcji w połączeniu z argumentami domyślnymi może prowadzić do niejednoznaczności - błędu.

```
void fun(int a) { ... }
void fun(int a, int b = 0) { ... }

main()
{
    fun(1); // ????? - błąd
}
```

Referencje

W C++ można tworzyć tzw. zmienne referencyjne. Jeśli zadeklarujemy zmienną referencyjną do innej zmiennej, to jakiekolwiek zmiany wartości dowolnej z powiązanych w ten sposób zmiennych będą odnosiły się również do drugiej z nich.

```
int a = 5;
int &b = a; // b jest zmienną referencyjną do a

b = 10;     // a == 10 && b == 10
a = 20;     // a == 20 && b == 20
```

Referencje c. d.

Referencja jest najczęściej wykorzystywana w funkcjach modyfikujących swoje argumenty.
(W C konieczne było w tym celu przekazywanie do funkcji wskaźników do zmiennych)

```
void zamien(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

main()
{
    int a = 1, b = 2;
    zamien(a, b);
    // a == 2 && b == 1
}
```

Dynamiczny przydział pamięci

W C++ operatory *new* i *delete*
(W C *malloc*, *calloc*, *realloc*, *free*, *sizeof*, konwersje wskaźników z *void**)

```
char *str1, *str2;
str1 = new char;
*str1 = 'a';
str2 = new char('b');

char *str3 = new char('c');

char *str4;
str4 = new char[11];

if (str4)
{
   strcpy(str4, "1234567890");
}
else
{
    cout << "blad przydzialu pamieci" << endl;
}

cout << "str1=" << *str1 << endl;
```

Dynamiczny przydział pamięci (2)

```
float *f;
f = new float;
*f = 3.14;
cout << *f << endl;
delete f;

int *tab;            // tablica dynamiczna
tab = new int[10];   // rozmiar tablicy = 10

for (int i = 0; i < 10; i++)
{
    tab[i] = i;
    cout << tab[i] << endl;
}

delete[] tab;
```

Dynamiczny przydział pamięci (3)

- Obiekty utworzone za pomocą **new** istnieją do momentu ich jawnego usunięcia operatorem **delete**
- Obiekty takie nie maja nazwy; operowanie na nich tylko za pomocą wskaźników
- Po utworzeniu zawierają "śmieci"

```
delete str1;
delete str2;
delete str3;

// usunięcie dynamicznej tablicy
delete[] str4;
```

Dynamiczny przydział pamięci (4)

Próba usunięcia już usuniętego obiektu operatorem **delete** kończy się błędem "... access violation ..."

Rozwiązanie: zerowanie wskaźnika po jego usunięciu

```
delete str3;
str3 = NULL;

//...

delete str3;
// lub
// if (str3) delete str3;
```

Strumieniowe wejście/wyjście

Strumienie (iostream.h):

- wejściowy: *cin* (klawiatura) // nie wczytuje spacji!
- wyjściowy: *cout* (ekran)
- wyjściowy dla błędów: *cerr* (ekran)
- wyjściowy dla błędów: *clog* (buforowany, ekran)

Operatory:

>> - wejście << - wyjście

```
main()
{
    int a;
    cin >> a;
    cout << "Odczytano: " << a << '\n';
    return 0;
}
```

Przestrzenie nazw

- Zapewniają lokalność nazw w kodzie, pozwalając uniknąć konfliktów nazw
  - Domyślnie nazwy należą do globalnej przestrzeni nazw
- Nie były dostępne w pierwszych wersjach C++
  - Stare książki, tutoriale, mogą ich nie uwzględniać
- Deklaracja przestrzeni nazw:
  `namespace` `przestrzen { … }`
- Odwołanie do nazwy z innej przestrzeni nazw:
  `przestrzen::nazwa`
- Dyrektywa **`using`** pozwala na odwołania do nazw z innej przestrzeni bez prefiksu
  - **`using namespace`** `przestrzen;`
  - **`using`** `przestrzen::nazwa;`

C++ Standard Library

- Standardowa biblioteka C++ (objęta standardem ISO języka C++)
- Kolekcja podstawowych klas i funkcji
  - Kontenery (list, map, set, …)
  - Łańcuchy znaków (string, regex)
  - Strumienie (istream, ostream, …)
  - …
  - Obiekty strumieni (cin, cout, …)
- Pliki nagłówkowe bez rozszerzenia **`*.h`**
- Składniki zadeklarowane w przestrzeni nazw **`std`**

Przestrzenie nazw – Przykład 1

```
#include<iostream>

namespace space1 {
  void fun() {
    std::cout << "Hello from space1!" << std::endl;
  }
}

namespace space2 {
  void fun() {
    std::cout << "Hello from space2!\n" << std::endl;
  }
}

int main() {
   space1::fun();
   space2::fun();
}
```

Przestrzenie nazw – Przykład 2

```
#include<iostream>

// using namespace std; // niezalecane
using std::cout;
using std::endl;

namespace space1 {
  void fun() {
    cout << "Hello from space1!" << endl;
  }
}

namespace space2 {
  void fun() {
    cout << "Hello from space2!\n" << endl;
  }
}

int main() {
   space1::fun();
   space2::fun();
}
```

Literatura

- Bjarne Stroustrup, „Język C++”
- Bruce Eckel, „Thinking in C++”
- Andrzej Zalewski, „Programowanie w językach C i C++ z wykorzystaniem pakietu Borland C++”
- Paweł Chomicz, Robert Ulijasz, „Programowanie w języku C i C++”
- ...