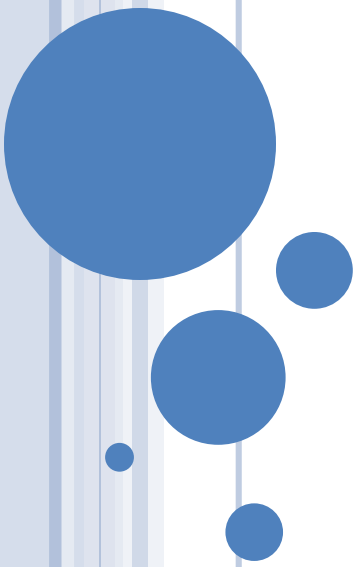

Kurs języka C++ – wykład 1 (21.02.2018)

Łagodne wprowadzenie do języka C++

Spis treści

- Pierwsze programy w C++
- Struktura programu w C++
- Zmienne ustalone `const` i ulotne `volatile`
- Referencje
- Pętla `for` dla przeglądania tablic
- Napisy typu `string`
- Typ `void`
- Wskaźnik pusty `nullptr`
- Standardowe wejście i wyjście
- Klasy, obiekty, pola, metody

Pierwszy program

- Najprostszy program w języku C++:

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

- Oto program powitalny w języku C++:

```
#include <iostream>

using namespace std;

int main (int argc, char *argv[]) {
    cout << "witaj na kursie C++" << endl;
    return 0;
}
```

Drugi program

- Oto program, który zamieni milimetry na cale:

```
#include <iostream>

using namespace std;

int main () {
    cerr << "[mm]: ";
    double mm;
    cin >> mm;
    double inch = mm/25.3995;
    cout << inch << endl;
    cerr << mm << "[mm] = " << inch << "[in]" << endl;
    return 0;
}
```

Struktura programu w C++

- Podział na pliki:
  - nagłówkowe (rozszerzenie `.hpp`) z deklaracjami,
  - źródłowe (rozszerzenie `.cpp`) z definicjami.
- W plikach nagłówkowych stosujemy włączanie warunkowe:

```
#ifndef moje_h
#define moje_h
/* właściwa zawartość pliku moje_h */
#endif
```

- Aby otrzymać uruchamialny plik wynikowy w jednym z plików źródłowych musi się znaleźć definicja funkcji `main()`.

Standardowe pliki nagłówkowe

- Pliki nagłówkowe odnoszące się do biblioteki standardowej nie mają żadnego rozszerzenia, na przykład:

```
#include <iostream>
#include <iomanip>
#include <string>
```

- Nazwy odnoszące się do starych plików nagłówkowych z języka C są poprzedzone literą ”c”, na przykład:

```
#include <cmath>
#include <cstdlib>
```

- Wszystkie definicje z biblioteki standardowej są umieszczone w przestrzeni nazw `std`, dlatego wygodnie jest na początku (małego) programu włączyć tą przestrzeń poleceniem:

```
using namespace std;
```

Podstawowe typy danych

- `bool` – typ boolowski (`true`, `false`)
- `char16_t`, `char32_t`, `wchar_t` – długie znaki
- `auto` – dedukcja typu na podstawie inicjalizatora
- `decltype` – dedukcja typu na podstawie typu wyrażenia

Stałe, czyli zmienne ustalone

- Stałe są oznaczone deklaratorem `const` w deklaracji:
  `const` *`TYP zmienna = wyrażenie;`*
- Stałą należy zainicjalizować podczas deklaracji.
- Inicjalizacja stałego argumentu w funkcji następuje podczas wywołania funkcji.
- Do stałej nie wolno w programie nic przypisać – jej wartość określamy tylko podczas inicjalizacji.
- Przykład:

```
const double phi = 1.618033989;
```

Stałe w porównaniu z makrodefinicjami

- Dlaczego stałe są bezpieczniejsze od makrodefinicji?
  - znany jest typ stałej
  - można określić zasięg nazwy stałej
  - nazwa stałej jest znana kompilatorowi
  - stała to komórka pamięci posiadająca swój adres
  - łatwiejsza praca z debugerem

Wyrażenia stałe

- Stałe wyrażenia są obliczane przez kompilator.
- Stałe wyrażenia definiuje się za pomocą `constexpr`.

Zmienne ulotne

- Zmienne ulotne są oznaczone deklaratorem `volatile` w deklaracji:
  `volatile` *`TYP zmienna;`*
- Obiekt ulotny może się zmieniać w sposób wymykający się spod kontroli kompilatora (na przykład w obliczeniach współbieżnych).
- Kompilator nie optymalizuje kodu ze zmiennymi ulotnymi.
- Przykład:

```
volatile int counter;
```

Referencje

- Operatory, które umożliwiają tworzenie typów złożonych:
  - `()` funkcja
  - `[]` tablica
  - `*` wskaźnik
  - `&` referencja
  - `&&` r-wyrażenie
- Referencja odnosi się do istniejącego w pamięci obiektu.
- Referencję trzeba zainicjalizować.
- Referencja nie może zmienić obiektu, z którym została związana w czasie inicjalizacji.
- Referencję implementuje się jako stały wskaźnik.

Referencje

- Definicja referencji:
```
typ &ref = obiekt;
```
- Przykład referencji:
```
int x = 4;
int &r = x;
```
- Referencje mają zastosowanie głównie jako argumenty funkcji i jako wartości zwracane przez funkcje.
- Przykład funkcji, która zamienia miejscami wartości zewnętrznych zmiennych:
```
void zamiana (double &a, double &b) {
    double c = a;
    a = b;
    b = c;
}
```

Popularne typy obiektowe

- Używaj łańcucha znakowego typu `string` zamiast tablicy znaków `char[]`.
- Używaj wektora `vector<T>` zamiast tablicy `T[]`.
- Używaj pary `pair<K, V>` z kluczem `K` (`first`) i wartością `V` (`second`).

Pętla `for` oparta na zakresie

- Zakresy reprezentują kontrolowaną listę pomiędzy dwoma jej punktami. Kontenery uporządkowane są nad zbiorem koncepcji zakresu i dwa iteratory w kontenerze uporządkowanym także definiują zakres.
- Nowa pętla `for` została stworzona do łatwej iteracji po zakresie; jej ogólna postać jest następująca:
  `for (TYP &x: kolekcja<TYP>) instrukcja;`
- Przykład:
```
int moja_tablica[5] = {1, 2, 3, 4, 5};
for(int &x: moja_tablica) { x *= 2; }
```
- Pierwsza sekcja nowego `for` (przed dwukropkiem) definiuje zmienną, która będzie użyta do iterowania po zakresie. Zmienna ta, tak jak zmienne w zwykłej pętli `for`, ma zasięg ograniczony do zasięgu pętli.
- Druga sekcja (po dwukropku), reprezentuje iterowany zakres. W tym przypadku, zwykła tablica jest konwertowana do zakresu. Mógłby to być na przykład `std::vector` albo inny obiekt spełniający koncepcję zakresu.

Napisy

- C-string to napis umieszczony w tablicy typu `char[]` zakończony znakiem o kodzie 0 `'\0'`.
- String to napis typu `string` przechowywany w obiekcie.
- Stringi są zadeklarowane w pliku nagłówkowym `<string>`.
- Stringi można ze sobą konkatenować za pomocą operatorów `+` i `+=`.
- W przypadku stringów nie trzeba się martwić o miejsce na napis – zostanie ono automatycznie zaalokowane.

Typ `void`

- Typ `void` informuje nas o braku typu.
- Typ `void` jest typem fundamentalnym, jednak nie wolno zadeklarować zmiennej typu `void`.
- Słowo `void` może wystąpić jako typ prosty w deklaracji typu złożonego:
  - `void *ptr;`
    oznacza wskaźnik do pamięci na obiekt nieznanego typu;
  - `void fun ();`
    oznacza, że funkcja nie będzie zwracała żadnego wyniku.

Wskaźnik pusty `nullptr`

- W starszym C++, stała 0 spełnia dwie funkcje: stałej całkowitej i pustego wskaźnika; programiści obchodzili tę niejednoznaczność za pomocą identyfikatora `NULL` zamiast 0.
- W języku C identyfikator `NULL` jest makrem preprocesora zdefiniowanym jako `((void*)0)`; w starym C++ niejawna konwersja z `void*` do wskaźnika innego typu jest niedozwolona, więc nawet takie proste przypisanie jak `char* c = NULL` mogłoby być w tym przypadku błędem kompilacji.
- Sytuacja komplikuje się w przypadku przeciążania:
  ```
  void foo(char*);
  void foo(int);
  ```
  Gdy programista wywoła `foo(NULL)`, to wywoła wersję `foo(int)`, która prawie na pewno nie była przez niego zamierzona.

Wskaźnik pusty `NULLPTR`

- Wskaźnik pusty , który nie pokazuje na żaden obiekt w pamięci zapisujemy jako `nullptr` – zastępuje makro `NULL` albo `0` (jest to adres o wartości 0 – adres pierwszej komórki w pamięci operacyjnej) i jest typu `nullptr_t`.
- Wskaźnik `nullptr` nie może być przypisany do typów całkowitych, ani porównywany z nimi.
- Wskaźnik `nullptr` może być porównywany z dowolnymi typami wskaźnikowymi.

Stos i sterta

- Stos to pamięć zarządzana przez program.
- Zmienne lokalne tworzone w instrukcji blokowej są automatycznie usuwane przy wychodzeniu z bloku.
- Sterta to pamięć, którą zarządza programista.
- Programista przydziela obszar pamięci dla zmiennej operatorem `new`, ale musi pamiętać o zwolnieniu tej pamięci operatorem `delete`.

Standardowe wejście i wyjście

- W bibliotece standardowej są zdefiniowane cztery obiekty związane ze standardowym wejściem i wyjściem:
  - `cin` standardowe wejście,
  - `cout` standardowe wyjście,
  - `clog` standardowe wyjście dla błędów,
  - `cerr` niebuforowane standardowe wyjście dla błędów.
- Do czytania ze strumienia wejściowego został zdefiniowany operator `>>`:

```
cin >> zmienna;
```

- Do pisania do strumieni wyjściowych został zdefiniowany operator `<<`:

```
cout << wyrażenie;
clog << wyrażenie;
cerr << wyrażenie;
```

- Operatory czytające `>>` ze strumienia i piszące `<<` do strumienia można łączyć kaskadowo w dłuższe wyrażenia (wielokrotne czytanie albo pisanie).