Programowanie w C++

Wykład 1

Konrad Kosmatka

Język C++

C++ to język programowania ogólnego przeznaczenia, który jest rozwinięciem języka C. Autor to **Bjarne Stroustrup**, a pierwsza wersja języka pojawiała się w latach **80 XX. wieku**.

Języki C/C++ zostały zaprojektowane pod kątem możliwości uzyskania wysokiej wydajności i efektywności, a także łatwości tworzenia oprogramowania (w porównaniu np. do Asemblera), ale nadal myśląc o architekturze sprzętowej komputera (m.in. typy zmiennych, zarządzanie pamięcią).

Dzięki temu są one do dnia dzisiejszego wykorzystywane tam, gdzie sprzęt posiada ograniczone zasoby (np. systemy embedded lub mikroprocesory), w systemach operacyjnych, gdzie operacje odbywają się blisko sprzętu oraz tam, gdzie potrzebna jest wysoka wydajność.

C with classes

Pierwszy kompilator języka C++ został napisany w... języku C++ („C with classes" kompilowany po przekształceniu do języka C).

„C with classes" - początkowo był nakładką na język C, tj. kod był parsowany i konwertowany na program w jęzku C, który mógłbyć skompilowany przez istniejące kompilatory języka C. Odbywało się to w postaci tzw. pre-procesora.

Język doczekał się także swojej właściwej nazwy **C++**.

Typy w języku C/C++

Typowanie w językach programowania dzielimy na

- **statyczne** (np. C, C++, Java, Go, Rust):
  - lepsza optymalizacja przez kompilator,
  - pozwala wyłapać więcej błędów w czasie kompilacji,

```
int calkowita;
float liczba;
struct struktura;
klasa instnacja;
```

- **dynamiczne** (np. Python, JavaScript, PHP):
  - pozwala na szybsze pisanie kodu.

Ze względu na to, że język C/C++ jest statycznie typowany, wszystkie zmienne muszą mieć określony typ, w tym również typ wartości zwracanej z funkcji.

Zmienne i typy danych

Typy:

- całkowite: int, long, short, unsigned int, itp.
- zmiennoprzecinkowe: float, double,
- znakowe: char,
- logiczne: bool (true/false),
- złożone: np. std::string.

Deklaracja zmiennej w postaci typ nazwa (opcjonalnie inicjalizacja):

```
int year = 2024;
```

Użycie niezainicjowanej zmiennej to UB (undefined behavior). Wartość zmiennej jest nieokreślona (zawartość pamięci na którą wskazuje zmienna może być dowolna), co prowadzi do nieprzewidywalnych wyników w programie.

Instrukcje warunkowe

- if,
- else if,
- else.

W językach C/C++ bloki kodu są zawarte w klamrach { }.

Instrukcje warunkowe

```
if (warunek) {
    ...
} else if (warunek2) {
    ...
} else {
    ...
}
```

W przypadku tylko jednej linii kodu klamry można pominąć:

```
if (warunek)
    std::cout << "Warunek spełniony" << std::endl;
```

Short-hand if:

```
int value = (warunek ? 1 : -1);
```

Pętle

- for,

```
for (int i = 0; i < N; i++) {
    ...
}
```

- while,

```
while (condition) {
    ...
}
```

- do ... while.

```
do {
    ...
} while (condition);
```

Słowa kluczowe: break (przerywa pętle), continue (przerywa bieżący przebieg pętli).

Funkcje

Funkcje służą do wyodrębnienia fragmentu kodu, aby podzielić go na mniejsze części oraz uniknąć jego powtarzania. Jest to również użyteczne przy testowaniu działania kodu (łatwiej przetestować krótkie funkcje).

Funkcja może zostać **zadeklarowana** i **zdefiniowana**.

Deklaracja funkcji informuje kompilator o jej nazwie, zwracanym typie i przekazywanych parametrach. Deklaracja nie zawiera implementacji funkcji i zwykle jest dodawana do **plików nagłówkowych** (rozszerzenia **h, hpp, hh, hxx**).

Definicja funkcji powtarza deklarację funkcji uzupełnioną o jej implementację. Jest zawarta w plikach źródłowych z kodem (rozszerzenia **c, cpp, cc, cxx**).

Deklaracja funkcji

Deklaracja funkcji informuje kompilator o jej nazwie, zwracanym typie i przekazywanych parametrach. Deklaracja nie zawiera implementacji funkcji.

Składnia:

```
return_type function_name(param1, param2, ..., paramN);
```

Przykład:

```
int add(int, int);
int add(int a, int b);
```

W deklaracji można pominąć nazwy parametrów, zostawiając jedynie ich typy. Oba powyższe zapisy są równoważne.

Definicja funkcji

Definicja funkcji powtarza deklarację funkcji uzupełnioną o jej implementację.

Składnia:

```
return_type function_name(param1, param2, ..., paramN) {
    /* function body */
}
```

Przykład:

```
int add(int a, int b) {
    return a + b;
}
```

Jeżeli funkcja nic nie zwraca, to oznaczamy ją jako typ **void**.

Entry point

Entry point to miejsce w programie, w którym rozpoczyna się jego wykonywanie. W języku C i C++ funkcja, która zostanie wywołana po starcie programu nazywa się **main** i zwraca liczbę całkowitą.

Najprostszy program w jezyku C/C++:

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

Wartość zawracana w funkcji **main** to status przekazywany po zakończeniu działania programu, który otrzymuje system operacyjny. Zwyczajowo wartość zero oznacza poprawne zakończenie programu, a inna wartość oznacza wystąpienie jakiegoś błędu w trakcie działania programu (tzw. run-time).

Dyrektywy preprocesora

Dyrektywy preprocesora to instrukcje, które są przetwarzane przed kompilacją programu.

Preprocesor przygotowuje faktyczny kod dla kompilatora przed wygenerowaniem jakiegokolwiek kodu maszynowego.

Dyrektywy preprocesora zaczynaja się od znaku # i mogą służyć m.in. do:

- definicji stałych wartości (**define**, **undef**),
- dołączania innych plików źródłowych (**include**),
- warunkowej kompilacji fragmentów kodu (**if, ifdef, ifndef, else, elif, endif**),
- generowania błędów w czasie kompilacji programu (**error**).

Więcej: https://cplusplus.com/doc/tutorial/preprocessor/

Entry point

Funkcja **main** w języku C i C++ uzyskuje od systemu operacyjnego dwa argumenty:

- **argc** - ilość argumentów przekazanych do programu,
- **argv** - stringi w stylu C, które zawierają argumenty.

Pierwszy z argumentów jest zwykle nazwą (ścieżką) pliku wykonywalnego, który jest uruchamiany z poziomu systemu operacjnego.

Entry point

Wypiszmy wszystkie argumenty programu:

```
#include <iostream>

int main(int argc, char *argv[]) {
    for (int i = 0; i < argc; i++) {
        std::cout << argv[i] << std::endl;
    }
    return 0;
}
```

Wykorzystujemy dyrektywę preprocesora **#include** aby dołączyć plik nagłówkowy **iostream**, który jest częścią biblioteki wejścia/wyjścia.

Komentarze

Aby utworzyć komentarz w języku C/C++ stosuje się:

- // - dla komentarza z zasięgiem do końca linii,
- /* ... */ - dla komenatrza z zasięgiem dowolnej ilości linii.

Uwaga: znak # jest zarezerwowany dla preprocesora, więc nie służy do tworzenia komentarzy w kodzie tak jak w niektórych jezykach programowania (np. Python).

Komentarze

```
#include <iostream>

int main(int argc, char *argv[]) {
    for (int i = 0; i < argc; i++) {
        std::cout << argv[i] << std::endl;
    }
    /* tutaj
       zrobimy
       komentarz
       na kilka
       linii */
    return 0; // lub tutaj do końca linii
}
```

Namespaces

W języku C łatwo o kolizję nazw funkcji w projekcie lub pomiędzy nimi.

Dlatego zwyczajowo zewnętrzne biblioteki posiadają funkcje i inne elementy, które zaczynają się od pewnego prefiksu, np.

- GLib: **g_** (np. **g_**get_user_name),
- GSL: **gsl_** (np. **gsl_**spline_eval),
- JSON-C: **json_** (np. **json_**tokener_parse),
- gzip: **gz** (np. **gz**open).

Namespaces

Język C++ wprowadza **przestrzenie nazw** (namespaces).

```
namespace nazwa {
    ...
}
```

Do wszystkich elementów (funkcje, struktury, wyliczenia, klasy, ...) zapisanych w danej przestrzeni nazw odwołujemy się przez operator **::**, np. **nazwa::funkcja()**.

Biblioteka standardowa znajduje się w przestrzeni o nazwie **std**. Przykładowo aby wykorzystać funkcję **cout** wywołujemy **std::cout**.

Można również pominąć zapis **std::** w danym pliku korzystając z:

```
using namespace std;
```