Podstawy programownia (w języku C++)

Pierwsze kroki (płytka woda)

Marek Marecki

Polsko-Japońska Akademia Technik Komputerowych

19 października 2020

Overview

Pierwszy program

Argumenty wiersza poleceń

Wskaźnik i tablica w stylu C

Tradycyjny pierwszy program

Tradycją w światku programistów jest, aby pierwszym programem napisanym w nowym języku było wypisanie na ekran tekstu "Hello, World!". W C++ taki program wygląda następująco:

```
#include <iostream>

auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Na jego przykładzie omówię strukturę programu.

Pliki nagłówkowe

```
#include <iostream>

auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Pliki nagłówkowe zawierają deklaracje funkcji, zmiennych, stałych, oraz struktur danych. Bez ”podłączenia” nagłówków do naszego kodu źródłowego kompilator nie będzie świadomy istnienia rzeczy zdefiniowanych w tych plikach i nie będzie możliwe ich użycie.
Sposobem na dołączenie plików nagłówkowych do naszego kodu źródłowego jest *dyrektywa* `#include`.

Nazwa funkcji

Definicja funkcji

```
#include <iostream>

auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Definicję funkcji rozpoczyna *słowo kluczowe* `auto`, po którym podana jest nazwa funkcji – np. `main`.
Nazwa funkcji może składać się wyłącznie z małych i wielkich liter, znaków podkreślenia (`_`) i cyfr, oraz **musi** rozpocząć się literą.

W szkole na matematyce pisało się `f(x) = x + 1`. Nazwą funkcji było oczywiście `f`.

Lista parametrów formalnych funkcji

Definicja funkcji

```
#include <iostream>

auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Po nazwie podana jest lista *parametrów formalnych* funkcji, czyli wartości, których podania funkcja będzie wymagała przy jej wywołaniu. Lista ta może być pusta.

W szkole na matematyce pisało się `f(x) = x + 1`. Parametrem formalnym funkcji `f` jest w tym przypadku `x`.

Typ zwracany

Definicja funkcji

```
#include <iostream>

auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Kolejnym elementem jest *typ zwracany* (ang. *return type*) funkcji, czyli typ wartości produkowanych przez daną funkcję. Zapisuje się go po "strzałce".
Jeśli funkcja nie produkuje żadnych wartości należy użyć typu `void`.

W szkole na matematyce pisało się `f(x) = x + 1` i nie podawało sie typu zwracanego. W domyśle typem tym była "liczba".

Deklaracja funkcji

Definicja funkcji

```
#include <iostream>

auto main() -> int;
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Gdyby po typie zwracanym wpisać średnik powstałaby *deklaracja funkcji*. Takie deklaracje umieszcza się w plikach nagłówkowych. Są one deklaracją tego, że definicja takiej funkcji będzie dla kompilatora dostępna i powinien on pozwolić na jej użycie.

W szkole na matematyce pisało się `f(x) = x + 1` i nie występowało coś takiego jak deklaracja.

Ciało funkcji 1

Definicja funkcji

```
#include <iostream>

auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Po typie zwracanym zapisuje się *ciało funkcji*, czyli grupę instrukcji, których dana funkcja jest abstrakcją[1]. Ciało funkcji mysi być otoczone nawiasami klamrowymi.

W szkole na matematyce pisało się `f(x) = x + 1` i trzeba było otaczać ciała funkcji (czyli `x + 1`) nawiasami klamrowymi.

[1] patrz *procedural abstraction* z poprzedniego wykładu

Ciało funkcji 2

Definicja funkcji

```
#include <iostream>

auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Ciało funkcji może składać się z dowolnej liczby instrukcji.

W szkole na matematyce pisało się `f(x) = x + 1`, a ciałem funkcji było zazwyczaj jakieś proste działanie.

Ciało funkcji 3

Definicja funkcji

```
#include <iostream>

auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Jeśli funkcja produkuje jakąś wartość (jej typem zwracanym nie jest `void`), to **musi** pojawić się w jej ciele instrukcja `return`.
Typ wartości zwróconej przez instrukcję `return` **musi** się zgadzać z typem zwracanym deklarowanym przez funkcję.

W szkole na matematyce pisało się `f(x) = x + 1` i oczywistym było, że zwracaną wartością jest wynik dodawania.

Nazwa funkcji

"W szkole na matematyce" *vs* C++

```
f
```

vs

```
auto f
```

Lista parametrów formalnych

"W szkole na matematyce" *vs* C++

```
f(x)
```

vs

```
auto f(int const x)
```

Typ zwracany

"W szkole na matematyce" *vs* C++

```
f(x)
```

vs

```
auto f(int const x) -> int
```

Ciało funkcji

"W szkole na matematyce" *vs* C++

```
f(x) = x + 1
```

vs

```
auto f(int const x) -> int
{
    return (x + 1);
}
```

Nazwa funkcji

"W szkole na matematyce" *vs* C++

```
main
```

vs

```
auto main
```

Lista parametrów formalnych

"W szkole na matematyce" *vs* C++

```
main()
```

vs

```
auto main()
```

Typ zwracany

"W szkole na matematyce" *vs* C++

```
main()
```

vs

```
auto main() -> int
```

Ciało funkcji

"W szkole na matematyce" *vs* C++

```
main(x) = print("Hello, World!"), 0
```

vs

```
auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Operator przekierowania: <<

Wracając do przykładu ze slajdu 3. Może nie być jasne co dzieje się w tej linijce:

```
std::cout << "Hello, World!\n";
```

Otóż...

Operator przekierowania: <<

Do zmiennej globalnej `std::cout`

```
std::cout << "Hello, World!\n";
```

...za pomocą *operatora przekierowania*

```
std::cout << "Hello, World!\n";
```

...wysyłany jest napis `Hello World!\n` (w C++ napisy ograniczane są znakami cudzysłowu).

```
std::cout << "Hello, World!\n";
```

Co spowoduje wypisanie tego napisu na ekran. Można w ten sposób wypisać na ekran różne wartości (liczbowe, logiczne, itd.).

Przykładowy kod

Kod dla programu `Hello, World!` znajduje się w repozytorium z zajęciami[2] w pliku '`src/00-hello-world.cpp`' Można go skompilować następującym poleceniem:

```
make build/00-hello-world.bin
```

Uruchomienie:

```
./build/00-hello-world.bin
```

Zadanie: zmienić kod tak, żeby wypisywał `Hello,` a potem imię studenta.

[2] `https://git.sr.ht/~maelkum/education-introduction-to-programming-cxx`

Overview

Pierwszy program

Argumenty wiersza poleceń

Wskaźnik i tablica w stylu C

Argumenty do programu

W repozytorium z zajęciami znajduje się plik ‘`src/01-hello-argv.cpp`’ Zawiera on kod źródłowy programu, który używa argumentów przekazanych mu na wierszu poleceń.
Można go skompilować następującym poleceniem:

```
make build/00-hello-argv.bin
```

Uruchomienie:

```
./build/01-hello-argv.bin Kasia
```

Zadanie: sprawdzić się stanie jak nie poda się argumentu (tj., uruchomi program bez ”`Kasia`”.).

Parametry formalne funkcji main

Argumenty wiersza poleceń

Żeby mieć możliwość odczytania argumentów podanych do programu na wierszu poleceń, lista parametrów formalnych funkcji `main` musi wyglądać następująco:

```
auto main(int argc, char* argv[]) -> int
```

Na kolejnych slajdach objaśnię znaczenie każdego z tych parametrów.

Parametry formalne funkcji main – argc

Argumenty wiersza poleceń

```
auto main(int argc, char* argv[]) -> int
```

`argc` (od *argument count*) przechowuje liczbę argumentów przekazanych do programu jako liczbę całkowitą.

Parametry formalne funkcji main – argv

Argumenty wiersza poleceń

```
auto main(int argc, char* argv[]) -> int
```

`argv` (od *argument values*) przechowuje wartości argumentów przekazanych do programu. Typ parametru `argv` może być nieco zagadkowy, ale ta zagadka zostanie rozwiązana na kolejnych slajdach.

Parametr argv – tablica wskaźników do char

Argumenty wiersza poleceń

Parametr argv jest

```
char* argv[]
```

...tablicą w stylu C (unikamy ich)

```
char* argv[]
```

...wskaźników (oznaczanych przez * za nazwą typu) do char

```
char* argv[]
```

char* jest sposobem na reprezentację napisów w stylu C.

Overview

Pierwszy program

Argumenty wiersza poleceń

Wskaźnik i tablica w stylu C
- Wskaźnik
- Tablica w stylu C

The good, the bad, and the ugly ...w stylu C

Język C++ wywodzi się z języka C. Jeśli dla jakiejś konstrukcji język C++ oferuje swój zamiennik, to ta odziedziczona jest określana jako ”w stylu C” (ang. *C-style*).

The ugly - wskaźniki

The good, the bad, and the ugly ...w stylu C

Konstrukcją "brzydką" są wskaźniki.

Ich typy są nieintuicyjne w zapisie, a wskaźniki same w sobie nie oferują żadnej gwarancji poprawności - nigdy nie wiemy czy wskaźnik przypadkiem nie jest *wiszący*[3].

W C++ część zadań wskaźników przejęły *referencje* (ang. *reference*).

[3] wytłumaczenie na slajdzie 37

The bad - tablice w stylu C

The good, the bad, and the ugly ...w stylu C

Konstrukcją "złą" są tablice.

Bardzo łatwo gubią rozmiar (który dla pewności musi być przechowywany w osobnej zmiennej), a jeśli zostanie on zgubiony niemożliwe jest jego odtworzenie. Na dodatek, tablice bardzo "chętnie" degradują się do wskaźników zatracając całkowicie informację o tym, że przechowują n elementów i nabywając wszystkie wady wskaźnika.

W C++ zamiennikiem (dużo lepszym) tablic w stylu C są: `std::array` (dla tablic o stałym rozmiarze) i `std::vector` (dla tablic o zmiennym rozmiarze).

Wskaźnik

Czym jest wskaźnik[4]? Wskaźnik jest adresem fragmentu pamięci (zawierającego dane typu t).

Mając wskaźnik można "dostać się" do danych umieszczonych w pamięci pod adresem, na który wskazuje wskaźnik.

Typ wskaźnika zapisuje się jako: $t*$
gdzie t jest typem danych leżących pod adresem, na który wskazuje wskaźnik.
Typem oznaczającym "wskaźnik do `int`" będzie `int*`

[4]Część "Data structures" na poprzednim wykładzie.

Tworzenie wskaźników

Wskaźnik

Aby otrzymać adres fragmentu pamięci zawierającego zmienną x używa się operatora & (ampersand)

```
auto x         = int{42};
auto x_pointer = &x;
```

Wskaźnik otrzymujemy również w sytuacji gdy alokujemy pamięć dynamicznie.

Używanie wskaźników

Wskaźnik

Aby użyć (odczytać lub zmodyfikować) danych z fragmentu pamięci wskazywanego przez wskaźnik używa się operatora * (gwiazdka)

```
auto x         = int{42};  // x contains 42
auto x_pointer = &x;  // x_pointer contains address of x

auto y         = (*x_pointer + 1);  // y contains 43
*x_pointer = 44;  // x contains 44
```

Po co są wskaźniki?

Wskaźnik

Wskaźniki są niezbędne jeśli chcemy korzystać z dynamicznej alokacji pamięci. Jeśli jakaś wartość jest alokowana w czasie wykonywania programu, jest ona umieszczana w innym obszarze pamięci (na *stercie*, ang. *heap*) niż wartości alokowane podczas kompilacji (na *stosie*, ang. *stack*) i niemożliwy byłby bezpośredni dostęp do niej.

Wskaźniki są niezbędne również jesli chcemy przekazać dane jako argument do funkcji, ale ich kopiowanie (domyślny sposób przekazywania argumentów) byłoby kosztowne. W takim wypadku przekazujemy funkcji jedynie adres danych i pozwalamy jej używać ich "w miejscu".

Problemy ze wskaźnikami

Wskaźnik

Wskaźniki mogą być *zerowe* (ang. *null pointer*, słowo kluczowe `nullptr`), czyli wskazywać na adres 0 w pamięci.

Wskaźniki mogą być *wiszące* (ang. *dangling pointer*), czyli wskazywać na adres w pamięci, który już nie należy do naszego programu (np. został zdealokowany i oddany do systemu operacyjnego).

Jeśli spróbujemy użyć wiszącego lub zerowego wskaźnika do odczytania lub modyfikacji danych nasz komputer może wybuchnąć i mogą nam "z nosa wylecieć demony[5]". Jest to tzw. *zachowanie niezdefiniowane* (ang. *undefined behaviour*) - jest ono źródłem wielu awarii i naruszeń bezpieczeństwa w programach.

[5] `https://en.wikipedia.org/wiki/Undefined_behavior`

Tablica w stylu C

Czym jest tablica w stylu C[6]? Tablica w stylu C jest fragmentem pamięci wystarczająco dużym żeby pomieścić n wartości typu t.

Typ tablicy zapisuje się jako: $t[n]$ (n można pominąć)
gdzie t jest typem danych leżących pod adresem, na który wskazuje tablica, a n rozmiarem tablicy.
Typem oznaczającym "tablicę *nie-wiadomo-ilu* wartości `int`" będzie `int[]`
Typem oznaczającym "tablicę 4 wartości `int`" będzie `int[4]`

[6]Część "Data structures" na poprzednim wykładzie.

Tworzenie tablic w stylu C

Tablica w stylu C

Aby stworzyć tablicę n wartości typu t używa się następującej składni:

```
t array[n];
```

Dla przykładu, tablica 4 wartości typu `int`:

```
int array[4];
```

Kompilator może też określić rozmiar tablicy automatycznie jeśli podamy elementy jakimi tablica powinna zostać zainicjalizowana:

```
int array[] = { 0, 1, 2, 3 };
```

Używanie tablic w stylu C

Tablica w stylu C

Aby dostać się do elementu n w tablicy używa się operatora `[]`

```
int array[] = { 42, 64, 127, -1 };
auto x = array[0];  // x contains 42
```

Po co są tablice?

Tablica w stylu C

Tablice służą do przechowywania wielu wartości tego samego typu.

W C++ dostępne są lepsze zamienniki tablic: `std::array` i `std::vector`.