Początki z C++ 2

Po kilku zajęciach za sobą prawdopodobnie czujesz, jak wiatr wieje Ci w plecy i posuwasz się w szybkim tempie do przodu. Od tej pory wiatr będzie wiał Ci prosto w oczy, jeśli nie będziesz czytał notatek uważnie. Dochodzimy bowiem do zagadnień, które nie są już takie oczywiste i intuicyjne dla osoby, która nigdy nie zajmowała się programowaniem.

> *Cieśla, który chce dobrze robotę wykonać,*
> *musi wpierw naostrzyć swe narzędzia.*
> *~Konfucjusz*

2.1 Typy danych

Na samym początku musimy zapoznać się z pojęciem zmiennej. Zmienna jak sama nazwa wskazuje będzie *UWAGA! UWAGA!* zmieniać się w trakcie programu. Zmienna to pewien stosunkowo mały obszar w pamięci, w którym możemy przechowywać dane różnego typu np. liczby całkowite, liczby zmiennoprzecinkowe(rzeczywiste), znak, tekst oraz kilka innych. Nie można jednak wszystkiego zapisać w jednej zmiennej (*to nie python*).

Nazwa typu	Ilość Bajtów	Zakres wartości
bool	1	**false** lub **true**
char	1	od -128 do 127
unsigned char	1	od 0 do 255
wchar_t (reprezentuje Unicode)	2	od 0 do 65 535
short	2	od -32 768 do 32 767
unsigned short	2	od 0 do 65 535
int	4	od -2 147 483 648 do 2 147 483 647
unsigned int	4	od 0 do 4 294 967 295
long	4	od -2 147 483 648 do 2 147 483 647
unsigned long	4	od 0 do 4 294 967 295
Typy zmiennoprzecinkowe		
float	8	3.4E +/- 38 (7 cyfr)
double	8	1.7E +/- 308 (15 cyfr)

Często pojawia się słowo “**Inicjalizacja**”, a co to znaczy? Inicjalizacja oznacza nadawanie początkowej wartości zmiennej w chwili jej tworzenia.

Nazewnictwo zmiennych

Nazwy zmiennych nie mogą zawierać polskich znaków. Dozwolone znaki to (a..z), (A..Z), podkreślenie (_) i cyfry (0..9). Nazwa zmiennej nie może zaczynać się od liczby. Warto wspomnieć, że `int abc;` i `int ABC;` to dwie różne zmienne, o różnych nazwach.

Zasięg zmiennych

Każda zmienna ma swój zasięg, w którym jest widoczna. Zasięg ten determinują klamry **{ … }.** Zmienna, którą tworzysz będzie widoczna tylko w obrębie danego bloku.

2.2 Funkcje

Funkcja jest częścią kodu, która reprezentuje sekwencję instrukcji. Funkcja może przyjmować argumenty oraz może zwracać jedną wartość (lub nic). Każda funkcja ma sygnaturę typu - typu jej argumentów i typu który zwraca.

Funkcje często są tworzone, aby wykonywać specyficzne zadania. Mogą być wywoływane z innych części programu, ale wcześniej trzeba je zdefiniować i zadeklarować.

2.2.1 Deklaracja funkcji

Deklaracją funkcji musi deklarować nazwę funkcji oraz typ jej sygnatury.

```
int dodaj2(int liczba);
```

Powyższy przykład jest tłumaczony dla kompilatora w taki sposób:

- Zwracany typ - `int`
- Nazwa funkcji - `dodaj2`
- Liczba argumentów - 1:
    - Pierwszy argument jest typu `int`
    - Pierwszy argument w treści funkcji zostanie określony nazwą `liczba`

Pierwsza zasada definiowania funkcji:

Funkcja z określonym typem/nazwą może zostać zadeklarowana tylko raz w całym kodzie C++. Innymi słowy: Nie możemy zadeklarować dwóch identycznych funkcji w obrębie jednego programu C++.

```
int dodaj2(int i); //Funkcja zostanie zadeklarowana
int dodaj2(int j); //Błąd kompilatora - istnieje już funkcja o takiej
                   //definicji.
```

Jeśli chcemy zadeklarować funkcję, która nic nie zwraca jako typ należy użyć `void`.

```
void zrob_cos(); //Funkcja nic nie zwraca
```

2.2.2 Definicja funkcji

Definicja funkcji jest bardzo podobna do jej deklaracji. Musimy tylko dodać kod, który ma się wykonywać w momencie wywołania funkcji.

Przykład definicji dla funkcji `dodaj2` jest:

```
int dodaj2(int liczba)
{
	int j = liczba + 2;
	return j;
}
```

2.2.3 Wywołanie funkcji

Aby wywołać funkcję wystarczy podać jej nazwę (łącznie z nawiasem) po tym jak została zadeklarowana. Przykład programu z funkcją `dodaj2`:

```
#include <iostream>

int dodaj2(int liczba)
{
	int j = liczba + 2;
	return j;
}

int main()
{
	std::cout << dodaj2(4) << std::endl;
	return 0;
}
```

2.2.4 Przeładowanie nazw funkcji

Możesz stworzyć wiele funkcji o tej samej nazwie, ale każda z nich musi zawierać inne parametry.

```
int dodaj2(int i)              //Ta funkcja zostanie wywołana w momencie
{                              //gdy podamy tylko jeden parametr
      int j = i + 2;
      return j;
}

int dodaj2(int i, int j)       //Ta funkcja zostanie wywołana gdy podamy
{                              //dwa parametry
      int k = i + j + 2;
      return k;
}
```

Obie funkcje wywoływane są za pomocą tej samej nazwy `dodaj2`. To, która zostanie ostatecznie wywołana w większości przypadków zależy od kompilatora. On dopasowuje wywołanie do odpowiedniej przeładowanej funkcji.

2.2.5 Parametry domyślne

```
int mnozenie(int a, int b = 7)
{
      return a * b;
}
```

Gdy wywołamy powyższą funkcję z jednym argumentem - wartość mnożnika (b) będzie równa 7.

Każdy parametr może mieć wartość domyślną, ale gdy decydujemy się, że nie wszystkie będą taką wartość posiadać - musimy najpierw wypisać argumenty bez wartości.

```
int mnozenie (int a = 10, int b); //Źle
int mnozenie (int a = 10, int b = 10); //Dobrze
int mnozenie (int a, int b = 10); // Dobrze
```

2.3 Plik

C++ uzyskuje dostęp do plików za pomocą strumieni wejścia/wyjścia (I/O). Kluczowe wyrażenia:

- `std::istream` służy to wczytywania tekstu.
- `std::ostream` służy do zapisywania tekstu
- `std::streambuf` służy do wczytywania i zapisywania znaków.

2.3.1 Zapisywanie do pliku

Jest kilka możliwości zapisywania do pliku. Najłatwiejszą w użyciu jest użycie plikowego strumienia wyjścia (output file stream - `ofstream`) razem z operatorem przepływu strumienia (`<<`):

```
std::ofstream os("foo.txt");
if(os.is_open()){
    os << "Hello LearnIT!";
}
```

Zamiast `<<` możesz również użyć funkcji `write()` należącej do biblioteki plikowego strumienia wyjścia:

```
std::ofstream os("foo.txt");
if(os.is_open()){
    char data[] = "Foo";

    //zapisuje 3 znaki z tablicy -> "Foo"
    os.write(data, 3);
}
```

Po zapisaniu do strumienia, powinieneś zawsze sprawdzać czy zostanie zwrócony błąd, np. wywołując należącą do biblioteki plikowego strumienia wyjścia funkcję `bad()`:

```
os << "Hello Badbit!";
if (os.bad())
    // Nieudana próba zapisania!
```

2.3.2 Otwieranie pliku

Otwieranie pliku jest wykonywane w ten sam sposób dla wszystkich 3 strumieni plików (`ifstream`, `ofstream` i `fstream`)
Możesz otworzyć plik bezpośrednio w konstruktorze:

```
std::ifstream ifs("foo.txt"); //ifstream: otwiera plik "foo.txt" tylko do czytania.

std::ofstream ofs("foo.txt"); //ofstream: otwiera plik "foo.txt" tylko do zapisywania.

std::fstream iofs("foo.txt"); //fstream: otwiera plik "foo.txt" do czytania i zapisywania
```

Inną możliwością jest użycie funkcji `open()` będącej członkiem strumienia pliku:

```
std::ifstream ifs;
ifs.open("bar.txt"); //ifstream: otwiera plik "bar.txt" tylko do czytania.

std::ofstream ofs;
ofs.open("bar.txt"); //ofstream: otwiera plik "bar.txt" tylko do zapisywania.

std::fstream iofs;
fs.open("bar.txt"); //fstream: otwiera plik "bar.txt" do czytania i zapisywania.
```

Powinieneś **zawsze** sprawdzać czy plik został otwarty pomyślnie (nawet przy zapisywaniu). Błędy mogą wyniknąć, gdy:

- plik nie istnieje,
- plik nie ma właściwych praw dostępu,
- plik jest już używany,
- wystąpi błąd dysku,
- dysk został odłączony...

Sprawdzanie może być wykonane w następujący sposób:

```
// Próba wczytania pliku 'foo.txt'.
std::ifstream ifs("fooo.txt"); // zanotuj literówkę; plik nie może zostać otwarty.

// Sprawdzenie czy plik został otwarty pomyślnie.
if (!ifs.is_open()) {
    // plik nie został otwarty; wykonaj akcje poniżej.
    throw CustomException(ifs, "File could not be opened");
}
```

Gdy ścieżka pliku zawiera “backslashe” (np. w systemach Windows) powinieneś “pominąć je”:

```
// Otwieranie pliku 'c:\\folder\\foo.txt' w Windowsie.
std::ifstream ifs("c:\\\\folder\\\\foo.txt"); //użycie "pominiętych backslashów"
```

Wersja ≥ C++11

lub użyć “surowej” ścieżki:

```
// Otwieranie pliku 'c:\\folder\\foo.txt'  Windowsie.
std::ifstream ifs(R"(c:\\folder\\foo.txt)"); // użycie "surowej ścieżki"
```

lub użyć w zamian slashów:

```
 // Otwieranie pliku 'c:\\folder\\foo.txt'  Windowsie.
std::ifstream ifs("c:/folder/foo.txt");
```

Wersja ≥ C++11

2.3.3 Czytanie z pliku

Istnieje kilka sposobów na wczytywanie danych z pliku.

Jeśli wiesz jak dane są sformatowane, możesz użyć operatora wydobycia strumienia (>>). Powiedzmy że masz plik nazwany *foo.txt*, który zawiera następujące dane:

```
John Doe 25 4 6 1987
Jane Doe 15 5 24 1976
```

Wtedy możesz użyć następującego kodu, by wczytać te dane z pliku:

```
// Definicja zmiennych.
std::ifstream is("foo.txt");
std::string firstname, lastname;
int age, bmonth, bday, byear;

// Wyodrębnij firstname, lastname, age, bday month, bday day, and bday
// year w tej kolejności.
// Pamiętaj: '>>' zwróci false gdy osiągnie EOF (end of file - koniec  //
pliku) lub gdy dane wejściowe nie mają poprawnego typu zmiennych (np. //
string "jan" nie może być wyodrębniony jako zmienna 'int'.
while (is >> firstname >> lastname >> age >> bmonth >> bday >> byear)
    // Wykonaj polecenia na danych które zostały wczytane.
```

Operator wydobycia strumienia (>>) wydobywa/wyodrębnia każdy znak i kończy, jeśli znajdzie znak, który nie może zostać przechowany lub jeśli jest znakiem specjalnym:

- dla typów string, operator kończy się na białym znaku () lub znaku nowej linii (\n).
- dla numerów, operator kończy się na znaku niebędącym numerem *(WOW).*

To oznacza, że następująca wersja pliku *foo.txt* również będzie poprawnie odczytana przez poprzedni kod:

```
John
Doe 25
4 6 1987
Jane
Doe
15 5
24
1976
```

Operator wydobycia strumienia (>>) zawsze zwraca strumień przekazany do niego. W związku z tym, operatory te mogą być ze sobą połączone w kolejności by czytać

sukcesywnie dane z pliku. Strumień może zostać także użyty jako wyrażenie Boolean (tak jak zostało to pokazane w pętli `while()` w poprzednim kodzie). To jest przez to że klasy strumienia mają własny sposób konwersji typu `bool`. Operator bool() będzie zwracać true tak długo, jak strumień nie będzie miał błędów. Jeśli strumień napotka się na błędne wyrażenie (np. gdy nie będzie już więcej danych do wyodrębnienia), wtedy operator bool() zwróci false. W związku z tym, pętla while z poprzedniego kodu zakończy się, gdy plik zostanie wczytany do końca.

(Dla chętnych)

1. Jeśli chciałbyś wczytać cały plik jako string, możesz użyć następującego kodu:

```
// otwiera 'foo.txt'.
std::ifstream is("foo.txt");
std::string whole_file;

// ustawia pozycję na koniec pliku
is.seekg(0, std::ios::end);

// rezerwuje pamięć dla pliku
whole_file.reserve(is.tellg());

// ustawia pozycję na początek pliku
is.seekg(0, std::ios::beg);

// ustawia zawartość 'whole_file' jako wszystkie znaki z pliku
whole_file.assign(std::istreambuf_iterator<char>(is),
    std::istreambuf_iterator<char>());
```

Ten kod rezerwuje miejsce dla stringu w celu usunięcia niepożądanych alokacji pamięci.

2. Jeśli chcesz wczytać plik linia po linii, możesz użyć funkcji getline():

```
std::ifstream is("foo.txt");

// Funkcja getline zwraca false gdy nie ma więcej linii.
for (std::string str; std::getline(is, str); ) {
    // Przetwarza linię która została wczytana
}
```

3. Jeśli chcesz wczytać podaną przez Ciebie liczbę znaków, możesz użyć funkcji `read()` należącej do klasy strumienia:

```
std::ifstream is("foo.txt");
char str[4];

// wczytaj 4 znaki z pliku
is.read(str, 4);
```

(Koniec dla chętnych)

2.3.4 Kopiowanie pliku

```
std::ifstream src("source_filename", std::ios::binary);
std::ofstream dst("dest_filename", std::ios::binary);
dst << src.rdbuf();
```

Wersja ≥ C++17

W standardzie C++17 standardową drogą kopiowania pliku jest dołączenie biblioteki nagłówkowej <filesystem> i użycie funkcji copy_file:

```
std::filesystem::copy_file("plik_źródła", "pik_docelowy");
```

2.3.5 Zamykanie pliku

Szczerze, zamykanie pliku jest rzadko potrzebne w C++, gdyż strumień pliku automatycznie zamyka powiązany plik w swoim destruktorze(o nim będzie później). Niemniej jednak, powinieneś spróbować ograniczyć czas korzystania z pliku, po to aby nie trzymać pliku otwartego dłużej niż to wymagane. Dla przykładu, może to zostać wykonane umieszczając wszystkie operacje pliku w jednym zakresie (`{ }`);

```
std::string const prepared_data = prepare_data();
{
    // otwórz plik dla zapisu.
    std::ofstream output("foo.txt");

    // zapisz dane
    output << prepared_data;
} // ofstream wyjdzie poza zakres w tym miejscu.
  // jego destruktor zajmie się zamykaniem pliku
```

Wywoływanie funkcji close() jest wymagane tylko wtedy, gdy chcesz użyć ponownie później ten sam obiekt fstream, ale nie chcesz zostawiać otwartego pliku pomiędzy tymi użyciami:

```
// otwórz plik "foo.txt" po raz pierwszy.
std::ofstream output("foo.txt");

// pobierz dane by móc je gdzieś zapisać
std::string const prepared_data = prepare_data();

// zapisz dane do pliku "foo.txt"
output << prepared_data;

// zamknij plik "foo.txt"
output.close();

// przygotowanie danych może zająć trochę czasu. W związku z tym, nie
otwieraj strumienia pliku przed momentem, gdy będziesz chciał coś do
niego wpisać
std::string const more_prepared_data = prepare_complex_data();

// otwórz plik "foo.txt" po raz drugi gdy jesteś gotowy na zapisywanie.
output.open("foo.txt");

// zapisz dane do pliku
output << more_prepared_data;

// zamknij plik ponownie
output.close();
```

2.4 Obsługa wyjątków

Po słowie kluczowym `try` następuje blok kody. W tym bloku try umieszczamy instrukcje, które mogą zwrócić wyjątek.

Wyjątkiem nazywamy niespodziewany / niepoprawny wynik kodu. (np. błąd łączenia z bazą danych, dzielenie przez 0 itd.)

```
      jakasFunkcja(); // tu wyjątek nie zostanie “złapany”

try {
      jakasFunkcja(); //Tu wyjątek zostanie “złapany” i przekazany do
                      //bloku catch
} catch (/* tu podajemy rodzaj wyjatku */){
      //Obsługa wyjątku
}
```

W bloku `catch` możemy sprecyzować jaki wyjątek będziemy obsługiwać, ale również możemy obsłużyć wszystkie wpisując `...`. Możemy też korzystać z wielu bloków `catch`.

```
try{
      // kod z wyjątkiem
} catch (const std::bad_alloc&){
      // obsługa wyjątku
} catch (const std::runtime_error& e){
     // obsługa wyjątku
} catch (...){
     // obsługa wyjątku
}
```

2.5 Struktury

Struktura to typ danych definiowany przez użytkownika, aby go użyć należy skorzystać z słowa kluczowego `struct`.
Części struktury i w przyszłości klasy w C++ nazywamy:

- polami (ang. fields) - są to zmienne wewnątrz danego typu,
- metodami (ang. methods) - są to funkcje wewnątrz danego typu

Struktury posiadają modyfikatory dostępu.

Wyróżniamy 3 rodzaje:

- Modyfikator **public** oznacza, że atrybut/pole lub metoda jest widoczna zarówno ze środka klasy, klas dziedziczonych jak i z zewnątrz klasy.
- Modyfikator **private** oznacza, że atrybut/pole lub metoda nie jest widoczna na zewnątrz klasy i nie może być dziedziczona.
- Modyfikator **protected** oznacza, że atrybut/pole lub metoda nie jest widoczna na zewnątrz klasy i może być dziedziczona

Struktury tworzymy w taki sposób.:

```
struct LearnIT
{
    int x;
    int y = 10;
    int funkcja(){
       return 11;
    };
};
```

Deklarując strukturę “dodajemy” nowy typ do programu i możliwe jest tworzenie instancji obiektów:

```
LearnIT moja_zmienna;
```

Dostęp do członków nowego typu jest możliwy dzięki “kropce”:

```
moja_zmienna.x = 10;
moja_zmienna.y = moja_zmienna.y + 1 //moja_zmienna.y jest równa 11
```