Rozdział 1

Wprowadzenie, typy zmiennych instrukcje sterujące

1.1 Wprowadzenie

Język C++ to nowoczesny język programowania. Jego główną zaletą jest możliwość programowania techniką obiektowo orientowaną (OOP). Technika ta pozwala na bardziej naturalne odwzorowywanie oprogramowywanego systemu (naszego zadania, które mamy zaprogramować). Wymaga ona więcej pracy i pisania kodu, ale dzięki temu nasz program staje się czytelniejszy i łatwiejszy w utrzymaniu oraz rozwijaniu. W ramach naszego kursu najpierw poznamy podstawy języka C++ (pierwsze 5 zajęć), a następnie przejdziemy do elementów techniki OOP. Być może będzie to początek pięknej przygody ze światem C++.

1.2 Struktura programu C++

Zwykle programy pisane w C++ składają się z wielu plików. (Np. w przypadku oprogramowania `OpenFOAM` służącego do numerycznego modelowania zjawisk cieplno-przepływowych ilość plików przekracza 50000! My oczywiście w ramach kursu nie będziemy tworzyć aż tak rozbudowanych programów, bo wymaga to wielu lat pracy i oblbrzymiego zespołu programistów.) Zawsze jednak musi być minimum jeden plik, we wnętrzu którego znajdować się będzie funkcja `main`. Jest to plik główny każdego programu C++. Przykładowy plik główny przedstawiony jest w Listingu 1.1. W linii 1 następuje załadowanie potrzebnych bibliotek za pomocą dyrektywy `include`. W tym przypadku potrzebujemy biblioteki do wyświetlania napisu na ekran (`iostream`), która zawiera funkcję `cout`. Następnie w linii 3 zaczyna się nasz program - widzimy wywołanie funkcji `main`. Ogólnie możemy zapamiętać, że funkcje (małe podprogramy) rozpoznajemy po tym, że po jej nazwie są dwa nawiasy. Wewnątrz nawiasu może nic nie być - jest to tzw. funkcja bezargumentowa - lub mogą tam być różnego rodzaju zmienne, które są przesyłane do funkcji (argumenty). To co robi funkcja - mówimy ciało funkcji -

znajduje się wewnątrz nawiasów klamrowych `{ }`.

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Nasz pierwszy program w C++!\n";
6 }
```

Listing 1.1: Minimalny program w C++

Nasz program składa się z jednej instrukcji, która po prostu wypisuje na ekran monitora napis `Nasz pierwszy program`. Wszystkie instrukcje C++ musimy kończyć średnikiem `;` . Aby to zrobić musimy po prostu wstawić tekst między znaczki `" "`. Widzimy też dodatkowy napis `\n` - jest to znak nowej linii. Jednak, aby nasz program zadziałał najpierw musimy go skompilować. W środowisku MS Visual Studio jest to proste - wystarczy nacisnąć klawisze `Ctrl + F5`.

Zadanie nr 1

Skopiuj komendę z linii 5 i usuń znak nowej linii. Zobacz jaki ma to skutek.

Do wytłumaczenia pozostało nam tajemnicze `std::cout <<` . Takie zapisy będziemy w przyszłości widywać częściej, więc lepiej od razu dowiedzieć się o co chodzi. Komenda `std::cout <<` składa się z 4 członów:

1. `cout` jest to nazwa funkcji pozwalająca wypisać napis na ekran.
2. `::` jest to tzw. operator zakresu.
3. `std` jest to nazwa przestrzeni nazw. Ogólnie w programie C++ nie możemy mieć dwóch takich samych nazw zmiennych bądź funkcji. Jeśli sami piszemy program to możemy dopilnować tego warunku przez wymyślanie coraz to nowych nazw. Jednak może się zdarzyć, że będziemy używać plików innych programistów i co bardzo prawdopodobne będą tam nazwy, które my już wykorzystaliśmy. W takim przypadku pojawia się błąd. Aby temu zapobiec wprowadzono przestrzeń nazw. Jest to, mówiąc obrazowo, nazwa folderu z różnymi nazwami. Jeśli chcemy wykorzystać nazwę z danego folderu musimy najpierw podać nazwę przestrzeni nazw, następnie operator zakresu i dopiero naszą nazwę. A przestrzenie nazw mogą być zagnieżdżone (tak jak foldery) i wtedy nasze nazwy się wydłużają, np. jeśli wewnątrz `przestrzenA` jest jeszcze `przestrzenB` i w `przestrzenB` jest funkcja o nazwie `mojaFun()` to odnosimy się do niej `przesrzenA::przetrzenB::mojaFun()`.
4. `<<` jest to operator do wypisywania na ekran (więcej o operatorach w następnej lekcji).

Przestrzenie nazw pomagają nam unikać kolizji nazw jednak przez to musimy więcej pisać kodu. Jeśli jesteśmy pewni, że na pewno będziemy używać tylko jednej przestrzeni nazw w danym pliku możemy posłużyć się dyrektywą `using namespace` co pokazano na Listingu 1.2. Dzięki temu możemy opóścić część `std::`.

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Nasz pierwszy program w C++!\n";
}
```

Listing 1.2: Wykorzystanie dyrektywy using namespace

1.3 Komentarze

Podczas pisania programów powinniśmy opisywać nasz kod przez stosowanie tzw. komentarzy. Pozwoli to nam na łatwe przypomnienie sobie co dany program robi jak otworzymy plik za jakiś czas. Jest to też pomocne dla użytkowników naszego programu (lub np. prowadzącego zajęcia C++ :).

W C++ mamy do dyspozycji dwa rodzaje komentarzy:

1. jednolinijkowe - `//` - wszystko co jest znakiem komentarza do końca linii jest ignorowane przez kompilator
2. wielolinijkowe - `/* */` - wszystko co jest pomiędzy znakami `/*` i `*/` jest traktowane jako komentarz.

Przykład użycia komentarzy przedstawiono w Listingu 1.3.

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Nasz pierwszy program w C++!\n";
    cout << "Komentarze w C++" << endl; //cout << "To jest komentarz
     jednolinijkowy w C++!\n";
    /*cout << "To jest komentarz wielolinijkowy w C++!\n";
  cout << "Tu tez jest komentarz\n";
  TUTAJ TEZ JEST KOMENTARZ!
  cout << "I tutaj\n";
  */
  cout << "Tu juz nie ma komentarza" << endl;
}
```

Listing 1.3: Przykład wykorzystania komentarzy w C++

Należy pamiętać, że komentarze wielolinijkowe nie mogą być zagnieżdżone.

1.4 Typy zmiennych

W programach oprócz funkcji występuje też wiele zmiennych, czyli nazw pod którymi przechowujemy różnego rodzaju dane. Zmienne mogą przechowywać różne dane. Przykłady to:

- znaki → zmienna typu `char`
- tekst → zmienna typu `string`
- liczba całkowita → zmienna typu `int`
- liczba rzeczywista (zmiennoprzecinkowa) → zmienna typu `double`
- stan logiczny → zmienna typu `bool`

Wyróżniamy dwa typy zmiennych:

1. globalne - dostępne w każdym miejscu naszego programu
2. lokalne - dostępne tylko w zakresie "życia"danej zmiennej. Zakres może być wyznaczony przez nawiasy klamrowe np. zmienne występujące wewnątrz funkcji są lokalne.

Inny podział zmiennych to:

1. wbudowane (int, bool, double, itd.)
2. użytkownika - zmienne tworzone przez użytkownika. Ten sposób tworzenia zmiennych poznamy później.

Listing 1.4 pokazuje przykład utworzenia różnych zmiennych lokalnych i globalnych.

```
#include <iostream>
using namespace std;

double Pi = 3.14;            // Zmienna globalna

int main()
{
    cout << "Tworzymy pierwsze zmienne w C++!\n";
  int lCalk = 2;             // Zmienna lokalna w main
  char znak = 'e';           // Zmienna lokalna w main
  bool stan = true;          // Zmienna lokalna w main
  bool innyStan;             // Zmienna lokalna w main
                             // nie zainicjalizowana
  double 44stopnie = 44.0;   // Blad - nazwa zmiennej nie moze
                             // zaczynac sie od cyfry
}
```

Listing 1.4: Przykłady zmiennych w C++

Przydatne zasady odnośnie zmiennych:

1. Nazwa zmiennej nie może zacząć się od cyfry.
2. Małe i wielkie litery są rozróżniane.
3. Zaleca się nadawać znaczące nazwy zmiennym.

4. W C++ każda zmienna zanim zostanie użyta musi zostać zadeklarowana, czyli musimy powiedzieć kompilatorowi jaki typ i jaka jest nazwa zmiennej zanim będziemy jej chcieli gdzieś użyć.

5. Nazwa zmiennej nie może być identyczna z żadnym słowem kluczowym w C++.

Kompletny zestaw zmiennych wbudowanych przedstawiono na rys. 1.1.

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

Rys. 1.1 Kompletny zestaw zmiennych wbudowanych w C++ [1]

Zadanie nr 2

Przeczytaj rozdziały 3.1-3.7 o typach wbudowanych w C++ z [2].

1.5 Wczytywanie z klawiatury

Do wczytywania danych z klawiatury służy komenda `std::cin >>`. Oto przykład wczytywania do zmiennej wartości liczbowej z klawiatury:

```
std::cout << "Podaj liczbe od 0-100" << std::endl;
double liczba;
std::cin  >> liczba;
std::cout << "Podana przez Ciebie liczba to: " << liczba << "." <<
   std::endl;
```

Listing 1.5: Przykład wczytywania wartości do zmiennej z klawiatury

1.6 Instrukcje sterujące

1.6.1 Wstęp

Ogólnie w języku C++ wykonywane są linijka po linijce. Jednak czasem pojawiają się części kodu, które chcielibyśmy wykonać jeśli byłby spełniony jakiś warunek. Czasem też jakiś fragment kodu chcielibyśmy wykonać wielokrotnie. Do tych celów służą nam tzw. instrukcje sterujące.

1.6.2 Prawda i fałsz w C++

W instrukcjach sterujących często występują wyrażenia, których wartością jest prawda lub fałsz. Jest kilka sposobów oceny czy coś jest prawdziwe czy fałszywe w C++:

- Najprostszy sposób to wpisanie do zmiennej `bool` wartości `true` (prawda) lub `false` (fałsz).
- Innym sposobem jest sprawdzenie czy jakieś wyrażenie jest prawdziwe bądź nie. Przykładem może być instrukcja sprawdzająca czy pod zmienną `litera` znajduje się wartość `'P'`. Jeśli tak to wynikiem jest prawda, jeśli nie to fałsz.
- Każda wartość niezerowa (nie równa zeru) jest rozumiana jako prawda. Wartość `0` oznacza fałsz.

1.6.3 Instrukcja warunkowa if

Instrukcję tą możemy stosować w dwóch formach (schematach):

1. if (warunek) instrukcja;
2. if (warunek) instrukcjaA;
   else instrukcjaB;

Działa ona w ten sposób, że sprawdza wartość logiczną wyrażenia `warunek`. Jeśli jest prawdziwa to instrukcja jest wykonywana. Jeśli nie to kod przechodzi dalej (przypadek 1)) lub wykonywana jest instrukcja po `else`. Często jest tak, że `instrukcja` składa się z wielu linii kodu. Wtedy należy te linie opatrzyć nawiasami klamrowymi jak pokazano w przykładowym kodzie Listingu 1.6. Zaleca się stosowanie klamer nawet gdy nasza instrukcja jest jednolinijkowa. Ponadto ważny jest sposób zapisu, który ułatwia czytelność kodu. Uzyskujemy to przez stosowanie znaków tabulatora lub określonej liczby spacji. Po klamrze `}` nie stawiamy średnika. Zauważ też, że możemy dowolnie zagnieżdżać instrukcje warunkowe. Stosowanie tabulatorów (spacji) klamer pomaga nam łatwiej zauważyć, do której części kodu odnosi się dany `if`.

```
if (a > b)
{
  std::cout << "Liczba " << a << " jest wieksza od " << b << ".";
  std::cout << "Ich iloczyn wynosi: " << a*b << "." << std::endl;
}
else
{
  std::cout << "Liczba " << b << " jest wieksza od " << a << ".";
  if (a > 0)
  {
    std::cout << "Liczba " << a << " jest dodatnia.\n";
  }
}
```

Listing 1.6: Przykład instrukcji if

Istnieje jeszcze dodatkowy sposób zapisywania instrukcji wielowariantowej. Jej schemat jest następujący:

- if (warunekA) instrukcjaA;
  else if (warunekB) instrukcjaB;
  else if (warunekC) instrukcjaC;
  else if (warunekD) instrukcjaD;

1.6.4 Pętla while

Instrukcja `while` ma następujący schemat:

```
while (warunek)
{
  instrukcje;
}
```

Listing 1.7: Schemat instrukcji while

Jeśli `warunek` jest fałszywy to `instrukcje` (może być to jedna bądź wiele instrukcji) nie są wykonywane. W przeciwnym razie są wykonywane. Po ich wykonaniu znowu następuje sprawdzenie wartości `warunek`. Można więc stworzyć pętlę nieskończoną jeśli `warunek` będzie zawsze `true`. Pętla zostaje przerwana dopiero, gdy `warunek` przyjmie wartość `false`. Należy pamiętać, że jeśli przed pierwszym wykonaniem pętli `warunek` będzie `false` to instrukcje pętli nie zostaną wykonane ani razu.

1.6.5 Pętla do while

Instrukcja `do while` ma następujący schemat:

```
do
{
  instrukcje;
} while (warunek);
```

Listing 1.8: Schemat instrukcji do while

Różnica w stosunku do `while` jest taka, że pętla ta wykona się minimum jeden raz. Czyli najpierw wykonywane są `instrukcje`, potem sprawdzany jest `warunek`. Jeśli jest prawdziwy to pętla zaczyna się od nowa, a jak nie to się kończy.

1.6.6 Pętla for

Instrukcja `for` ma następujący schemat:

```
for (instrukcjaInicjalizujaca; warunek; instrukcjaKroku)
{
  instrukcje;
}
```

Listing 1.9: Schemat instrukcji for

Jest to najczęściej używana pętla. Składa się ona z czterech części:

1. Wartość `instrukcjaInicjalizujaca` podaje początkową wartość zmiennej używanej w pętli.

2. `warunek` jest to wyrażenie logiczne sprawdzane w każdym obrocie pętli.

3. `instrukcjaKroku` podaje o ile ma się zwiększać wartość zmiennej używanej w pętli. Często jest to wartość 1.

4. `instrukcje` są to wykonywane w każdym obrocie pętli instrukcje.

Pracę pętli można podzielić na następujące etapy:

1. Wykonaj `instrukcjaInicjalizujaca`.

2. Sprawdź `warunek`. Jak jest fałszywy pętla jest przerywana. Jeśli prawdziwy program przechodzi do wykonywania `instrukcje`.

3. Po wykonaniu całego kodu zawartego w `instrukcje` następuje wykonanie `instrukcjaKroku` po czym znowu wykonywany jest punkt 2.

Cechy pętli `for`:

1. `instrukcjaInicjalizujaca` może się składać z wilu instrukcji, które w takim przypadku oddzielamy przecinkiem. Podobnie jest z `instrukcjaKroku`.

2. Dowolny z trzech elementów: `instrukcjaInicjalizujaca`, `warunek` lub `instrukcjaKroku` można opuścić. Należy jednak zachować średniki (`;`).

3. Nieskończoną pętlę `for` można stworzyć w ten sposób: for(;;) { instrukcje; }

1.6.7 Instrukcja break

Służy do przerywania danej instrukcji sterującej. Jeśli zastosujemy ją w pętli to po napotkaniu instrukcji `break` nastąpi natychmiastowe przerwanie pętli.

Przykład

Napisz program wypisujący liczby od 0 do 10. Następnie wykorzystaj instrukcję `break`, aby pętla została przerwana po liczbie 5.

```
for (int i=0; i<10; i++)
{
  std::cout << i << std::endl;
  // if (i > 5) break;
}
```

Listing 1.10: Przykład wykorzystania instrukcji break

1.6.8 Instrukcja continue

Zastosowana w pętli powoduje nie wykonanie instrukcji występujących po niej, ale nie przerywa pętli lecz przechodzi do jej następnego obrotu.

Przykład

Przykład wykorzystania instrukcji `continue`.

```
for (int i=0; i<10; i++)
{
  // if (i > 5) continue;
  std::cout << i << std::endl;
}
```

Listing 1.11: Przykład wykorzystania instrukcji continue

1.6.9 Instrukcja switch

Służy do wyboru jednego z kilku wariantów wykonania programu. Schemat instrukcji `switch` jest następujący:

```
switch (wyrazenie)
{
  case wyrazenieA:
    instrukcjeA;
    break;

  case wyrazenieB:
    instrukcjeB;
    break;

  case wyrazenieC:
```

```
12      instrukcjeC;
13      break;
14
15    default:
16      instrukcjeZ;
17      break;
18  }
```

Listing 1.12: Schemat instrukcji switch

Instrukcja `switch` działa w następujący sposób:

1. Obliczana jest wartość `wyrazenie`.
2. Wartość `wyrażenie` porównywane jest z etykietami stojącymi obok słowa `case`.
3. Jeśli wartość `wyrażenie` jest równa jednej z etykiet to wykonywane są instrukcje dla tej etykiety (np. `wyrazenie = wyrazenieB` to wykonają się `instrukcjeB` i następne chyba, że wcześniej wystąpi instrukcja `break`, która jest nieobowiązkowa.
4. Jeśli wartość `wyrazenie` nie jest równa żadnej etykiecie wtedy zostaną wykonane instrukcje `default`.
5. Jeśli wartość `wyrazenie` nie jest równa żadnej etykiecie i nie ma etykiety `default` to program opuszcza instrukcję `switch` nie wykonując żadnego wariantu.

1.7 Zadania do rozwiązania

Zadania na ocenę 3,0

Zad. 1. Napisz program, który wypisuje na ekranie tekst „Witaj świecie”.

Zad. 2. Napisz program, który pobiera dwie liczby całkowite, wyświetla w kolejnych wierszach ich sumę, różnicę, iloczyn i iloraz. Wyniki podaj w formie napisów, np. Suma = ..., Roznica = ..., itd. Wyniki zapamiętaj w zmiennych suma, roznica, iloczyn, iloraz (Podpowiedź: Spróbuj wykorzystać komendę `cin >>` (opcja nieobowiązkowa), służącą do wczytywania zmiennych z klawiatury.).

Zadania na ocenę 3,5-4,0

Zad. 3. Napisz program, który pobiera trzy znaki, a następnie w kolejnych wierszach wypisuje wszystkie możliwe permutacje (z powtórzeniami) tych

znaków. Przykładowo dla znaków ”a”, ”b” i ”c” program powinien wypisać „abc”, „acb”, „bac”, „bca”, „cab” i „cba” - w dowolnej kolejności.

Zad. 4. Napisz program, który pobiera długość boku kwadratu i wypisuje jego obwód oraz pole.

Zadania na ocenę 4,5-5,0

Zad. 5. Napisz program, który oblicza silnię liczby całkowitej (do n=20).

Zadania nieobowiązkowe

Zad. 6. Napisz program symulujący menu prostego kalkulatora. Program ma polegać na tym, że po uruchomieniu na ekranie pojawia się kilka opcji do wyboru: 1) oblicz kwadrat podanej liczby, 2) oblicz sześcian podanej liczby, 3) oblicz sumę 5 podanych liczb, 4) oblicz iloczyn 5 podanych liczb. Następnie użytkownik wybiera jedną z nich i program wykonuje odpowiednie instrukcje i wynik wyświetla na ekranie.

Bibliografia

[1] cplusplus.com. Variables and types. https://www.cplusplus.com/doc/tutorial/variables/.

[2] J. Grębosz. *Opus magnum C++11. Programowanie w języku C++*. Helion, 2019.