Część #1: Zmienne

W tej części kursu opiszę zmienne. Zmienna służy do przechowywania danych i odwoływania się do nich. Jest to najprościej mówiąc kawałek pamięci, do którego możemy się odnieść, zapisać w nim dane, lub te dane z niego odczytać. Jest to jeden z najczęściej wykorzystywanych elementów języka, oraz wiążę się z nim kilka pojęć oraz zasad ich używania. Nie jestem w stanie opisać wszystkich na raz w tej chwili, ponieważ wtedy ta część rozrosła by się do niebotycznych rozmiarów, dlatego opiszę je ogólnie, pokażę najprostsze przykłady ich wykorzystania, a ewentualne dodatkowe informacje na ich temat będą pojawiać się w trakcie kursu.

Oprócz tego w tym rozdziale przedstawię również najbardziej podstawowe koncepcje i składnię języka C++. Uznałem że nie ma sensu pisać osobnego rozdziału w celu wytłumaczenia tych rzeczy, ponieważ bardziej zrozumiale będzie wprowadzać je krok po kroku.

Jak tworzyć zmienne

Przykładowym stworzeniem zmiennej jest poniższy zapis:

```
int x;
```

W tym zapisie, tworzymy zmienną typu int o nazwie x. Szczegółem wartym zapamiętania jest to, że na końcu tego **wyrażenia** dajemy średnik ;. **Wyrażeniem**, ogólnie mówiąc, jest kawałek kodu wykonujący pewną czynność. Na przykład stworzenie zmiennej. Wyrażenia w języku C++ oddzielamy za pomocą średników, dlatego można (*ale niekoniecznie trzeba*) na przykład umieszczać kilka wyrażeń w jednej linijce kodu.

Każda zmienna w języku C++ - z racji, że jest to język *statycznie typowany* - musi mieć swój określony typ. W tym przypadku jest to int, czyli *typ całkowitoliczbowy ze znakiem*, co oznacza że zmienna tego typu może trzymać wartości liczbowe bez przecinka - na przykład 12 lub 24952, albo -245. O typach powiem w kolejnej części kursu.

Zasady nazywania zmiennych

1. **Nazwa zmiennej nie może zaczynać się od cyfry**. Nazwy w stylu 0zmienna albo 21x nie są dozwolone.
2. **Nazwa zmiennych może składać się jedynie ze znaków A-Z, a-z, 0-9 oraz - i _**. Nie wolno używać w nich innych znaków, na przykład spacji, ani specjalnych liter (typu ąłęćźż).
3. **Nazwy zmiennych są *case-sensitive***. Czyli, zmienna o nazwie Var i zmienna o nazwie var to dwie różne zmienne. Wielkość liter ma znaczenie.

Terminologia tworzenia zmiennych

Powyższy napis nazwałem prawidłowo, lecz potocznie, *stworzeniem* zmiennej. Tak na prawdę zachodzą tu dwie rzeczy.

1. **Deklaracja zmiennej** - Deklaracja jest to wprowadzenie nowej nazwy do kodu. Mówiąc potocznie,

mówimy kompilatorowi "Ej, patrz - masz tutaj nazwę x i będzie to zmienna typu int". W momencie deklaracji, kompilator wie że ta nazwa będzie *gdzieś* **zdefiniowana**.
2. **Definicja zmiennej** - To właśnie ona *tworzy* nam daną zmienną. Definiujemy tą zmienną i od tego momentu ona fizycznie istnieje w pamięci, możemy jej normalnie używać.

W większości przypadków będziemy się posługiwać terminem **definicja**, ponieważ łączy on dwie powyższe czynności. Szczegółowe różnice dotyczące deklaracji i definicji opiszę *kiedyś*, ponieważ jest to wchodzenie w szczegóły języka, których przez większość czasu nie będziemy potrzebowali. Możesz potraktować to jako ciekawostkę.

Inicjalizacja, czyli co dalej

Zadajmy sobie pytanie - jaką wartość ma zmienna po jej stworzeniu? Odpowiedź brzmi - w przypadku zmiennych podstawowych typów, takich jak int, char albo double - w większości przypadków, nie wiemy. Bierze się to stąd, że taka zmienna nie ma z automatu przypisanej żadnej wartości - w procesie *definiowania* zmiennej, zostaje zarezerwowana pamięć dla niej, ale ta pamięć nie zostaje w żaden sposób nadpisana. To co tam było, nadal tam jest, a my nie jesteśmy w stanie przewidzieć jaka to będzie wartość. Jest to tak zwany **undefined behaviour**, jeden z bardziej przykrych problemów z jakimi będzie dane się nam spotkać podczas programowania w C++ie. Jak ten problem rozwiązać? Poprzez *inicjalizację* zmiennej!

Inicjalizacja to, innymi słowy, przypisanie zmiennej jej pierwszej wartości. Jak możemy zainicjalizować zmienną? Na trzy różne sposoby!

1. **Przypisanie wartości**: int x = 10;. Jest to chyba najpopularniejsza metoda, a na pewno najbardziej *naturalna*. Zmienną możemy oczywiście inicjalizować nie tylko wartością liczbową, ale również wartością innej zmiennej (int x = y;) albo funkcji (int x = pow(2, 4);), lub też listą inicjalizacyjną (int x = {10};). Ten sposób inicjalizacji nazywany jest również *inicjalizacją kopiującą* (*copy-initialization*).
2. **Bezpośrednia inicjalizacja**: int x(10);. W tym wypadku, inicjalizujemy zmienną bezpośrednio, za pomocą *konstruktora* typu. Na razie załóżmy że nie różni się ona od inicjalizacji kopiującej, ponieważ w praktyce zazwyczaj robi to samo, oprócz jednego przypadku - nie należy inicjalizować zmiennej z użyciem nawiasów, jeśli nic w nich nie podamy, ponieważ wtedy inicjalizacja zostanie potraktowana jako deklaracja funkcji!
3. **Bezpośrednia inicjalizacja listą**: int x{10};. Ta inicjalizacja również wykorzystuje konstruktor typu, tak jak bezpośrednia inicjalizacja, ale dodatkowo sprawdza również czy typ wartości, którą inicjalizujemy naszą zmienną, jest prawidłowy - w przeciwnym wypadku, spowoduje błąd kompilacji. Ten kij ma dwa końce, ponieważ z jednej strony pozwala nam unikać dziwnych zachowań związanych z niejawnymi konwersjami typów, ale z drugiej czasami zmniejsza czytelność kodu - ponieważ musimy jawnie konwertować typy.

Notka: powyższe sposoby opisują konstruowanie typów prostych. W przypadku typów złożonych, takich jak klasy, inicjalizacja może być niekiedy nieco bardziej skomplikowana, co opiszę w kolejnych częściach. Na razie nie masz się czym przejmować 👌🏽

Tak więc, warto zapamiętać sobie zasadę, że **każdą zmienną należy zainicjalizować przed jej pierwszym użyciem**. Dzięki temu, unikniemy wielu trudno wykrywalnych błędów, które na pewno będą mogły się wkraść do naszego kodu. Przykładowo, ja zazwyczaj inicjalizuję zmienne w ten sposób:

```
int x{};
```

Ponieważ inicjalizacja z użyciem klamr bez argumentów automatycznie inicjalizuje mi zmienną jej wartością domyślną - w tym wypadku, jest to 0. Tutaj ponownie zaznaczam, że - jak opisałem to w punkcie 2. powyższej listy - int x(); nie będzie inicjalizacją, tylko deklaracją funkcji - dlatego unikajmy używania tego typu inicjalizacji, dla bezpieczeństwa.

Używanie zmiennych - odczyt i zapis

Dużo teorii za nami, teraz czas na odrobinę praktyki. Przyjmijmy następujący kod:

```
#include <iostream>

int main() {
  int a{};

  std::cout << "Enter a number: ";
  std::cin >> a;

  std::cout << "You have entered " << a << '!' << std::endl;
}
```

Okej, pojawiło się nagle dużo kodu. Przeanalizujmy go, linijka po linijce.

#include <iostream> - instrukcje zaczynające się od # to *dyrektywy prekompilatora*. Prekompilator to narzędzie wykorzystywane podczas kompilacji - na przykład do dołączania bibliotek do kodu, tak jak w tym przypadku. My dołączamy do naszego kodu bibliotekę iostream która zawiera narzędzia pozwalające nam między innymi na odczyt i wypisywanie tekstu z terminala.

int main() - jest to deklaracja funkcji main. Funkcja main to tak zwany *entry point* każdego C++'owego programu. Przy uruchomieniu go, zostaje ona automatycznie wywołana. Kod funkcji należy umieszczać w klamrach.

int a{}; - deklaracja naszej zmiennej.

std::cout << "Enter a number: "; - to wyrażenie, ogólnie mówiąc, wrzuca nam tekst do standardowego wyjścia - zazwyczaj jest to terminal, dzięki czemu zostaje on wyświetlony.

std::cin >> a; - a to wyrażenie odczytuje nam tekst ze standardowego wejścia do zmiennej a.

std::cout << "You have entered " << a << '!' << std::endl; - tutaj, wypisujemy komunikat "You have entered ", następnie wartość zmiennej i znak ! oraz *manipulator* nowej linii.

Zauważ różnicę między tekstem w "" a znakiem w ''. W języku C++, możemy trzymać łańcuchy znaków w podwójnych apostrofach, a pojedynczy znak pomiędzy pojedynczymi. Próba umieszczenia łańcucha znaków w '' skończy się błędem kompilacji.

Natomiast manipulator jest to specjalna funkcja, którą można umieścić w strumieniu. W przypadku std::endl, ten manipulator umieszcza w strumieniu znak nowej linii i powoduje flush strumienia, czyli czeka z wykonaniem następnej instrukcji do momentu w którym wszystko co zostało umieszczone w strumieniu zostanie zapisane (w naszym przypadku - wyświetlone). Szybszą alternatywą jest po prostu użycie znaku nowej linii - '\n' i sugeruję używać std::endl tylko na końcu umieszczania dużej ilości danych w strumieniu, a w reszcie przypadków '\n'.

std::cin i std::cout to obiekty standardowego wejścia i wyjścia. Pozwalają one na interakcję programu z terminalem, tak jak to wyżej zademonstrowałem. Będziesz ich bardzo często używać podczas nauki języka.

Oczywiście zmiennych możemy wykorzystywać na wiele innych sposobów, niż zwykły odczyt wartości z wejścia i wypisanie go. Ich możliwości będę pokazywał stopniowo. Zacznę od typów, jakie istnieją w języku C++, które opiszę w następnej części kursu.

Następna część - Podstawowe typy