

# IV. Pojęcie zmiennej i podstawowe typy danych

## 4.1. Zmienne

Programy, które wypisują komunikaty są zazwyczaj mało interesujące. Biblioteka `<iostream>` pozwala nam nie tylko wypisywać znaki, ale również je wczytywać do zmiennych. Zanim jednak zapoznamy się z instrukcją wczytywania, należy zapoznać się z pojęciem zmiennej. Zmienna, jak sama nazwa wskazuje będzie się zmieniać w trakcie programu. Zmienna to pewien dość mały obszar w pamięci, w którym możemy przechowywać dane różnego typu np. liczby całkowite, liczby rzeczywiste (zmiennoprzecinkowe), znak, tekst oraz kilka innych informacji, które będą nas w przyszłości interesowały. Nie można jednak wszystkiego zapisywać do jednej zmiennej. Każda zmienna ma swoje przeznaczenie, wielkość i właściwości. Na zmiennych liczbowych możemy wykonywać operacje matematyczne, w innych z kolei możemy przechowywać tekst. Zademonstruję teraz na przykładzie, jak się deklaruje zmienne i jak się z nich korzysta.

```
#include <iostream>
```

```
#include <conio.h>
```

```
int main()
```

```
{  
    using namespace std;  
    /*Deklaracja zmiennej typu "int" (liczby całkowite)  
o nazwie "zmienna_liczba" */  
    int zmienna_liczba;
```

```
    /*Deklaracja zmiennej typu "int" bez znaku  
(czyli bez znaku minusa (liczby większe lub równe 0))  
o nazwie "dodatnie"*/  
    unsigned int dodatnie;
```

```
    /*Deklaracja zmiennej typu "float" (liczby zmiennie przecinkowej)  
o nazwie "zmiennoprzecinkowa"*/  
    float zmiennoprzecinkowa;
```

```
//deklaracja zmiennej znakowej typu "char" o nazwie "jeden_znak"  
char jeden_znak;
```

```
/*deklaracja zmiennej znakowej typu "char" bez znaku minus  
(to będzie wymagało szerszego wyjaśnienia)*/  
unsigned char znak;
```

```
//Deklaracja zmiennej "abc" i przypisanie jej wartości początkowej równej -53;  
int abc = -53;  
//Przypisanie wartości 22 do zmiennej o nazwie "dodatnie";  
dodatnie = 22;  
//Przypisanie wartości 12.42 do zmiennej o nazwie "zmiennoprzecinkowa";  
zmiennoprzecinkowa = 12.42;  
/*Przypisanie do zmiennej o nazwie "znak" wartości 'c'  
(to również będzie wymagało szerszego wyjaśnienia)*/  
znak = 'c';
```

```
//wyświetlenie wszystkich zmiennych na ekranie  
cout << "wypisujemy zmienne:" << endl  
<<" zmienna_liczbowa: " << zmienna_liczba  
<< endl  
<<" dodatnie: " << dodatnie << endl
```

```
<<" abc: " << abc << endl
<<" zmiennoprzecinkowa: " << zmiennoprzecinkowa
<< endl
<<" jeden_znak: " << jeden_znak << endl
<<" znak: " << znak << endl;
```

```
getch();//Czekaj na dowolny znak z klawiatury
return(0);
}
```

## 4.2. Instrukcja przypisania

Powyższy przykład pokazuje nam, jak tworzy się zmienne w programie. Ogólnie można to zapisać następująco:

- `typ_zmiennej nazwa_zmiennej;`,
- lub `typ_zmiennej nazwa_zmiennej=wartość_zmiennej;`

Pierwszy zapis, rezerwuje nam pamięć w programie. Trzeba wiedzieć, że wartość w zmiennej będzie przypadkowa! **Kompilator nie zeruje wartości zmiennych!** Jeśli chcemy od razu nadać zmiennej początkową wartość, możemy to zrobić korzystając z zapisu drugiego. Jeśli nie chcemy, możemy zrobić to zawsze później w następujący sposób: `nazwa_zmiennej=nowa_wartość_dla_zmiennej;`. Należy wspomnieć tu jeszcze jak możemy nazywać zmienne (a raczej jak nie możemy ich nazywać).

## 4.3. Nazewnictwo zmiennych

Zasad tworzenia nazw w C++:

- Dozwolone znaki to: (a..z), (A...Z), podkreślenie ( \_ ) i cyfry (0...9) (nie można używać polskich znaków)
- Nazwa zmiennej nie może też się zaczynać od liczby
- Wielkie liter są odróżniane od małych czyli zmienna np.:  
int abc,  
int ABC,  
są to dwie różne zmienne.
- Nazwa nie może być słowem kluczowym C++ np. int int, char int, itp.
- Nazwa nie zaczyna się od podkreślenia ( \_ lub \_\_ ) takie nazwy są zarezerwowane dla kompilatora.
- Długość nazwy jest nieograniczona.

Warto stosować takie nazw zmiennych, które Tobie będą mówił do czego został użyta dana zmienna(nawet po upływie roku od napisania kodu). Dlatego warto poświęcić trochę więcej czasu na odpowiedni komentarz oraz umiejętnie(odpowiednie) nazewnictwo zmiennych.

## 4.4. Zasięg zmiennych

Każda zmienna ma również swój zasięg, w którym jest widoczna. Zasięg ten determinują klamry { ... }. Zmienna, którą utworzysz będzie widoczna tylko w obrębie danego bloku. Narazie nie będziemy wgłębiali w szczegóły. Jak przyjdzie na to pora, przypomnę tą informację, z szerszym wyjaśnieniem i odpowiednim przykładem.

## 4.5. Słowo kluczowe unsigned

Kolejną sprawą, której nie wyjaśniłem to słówko **unsigned** przed niektórymi zmiennymi. Wyraz ten możemy wpisywać przed każdą zmienną całkowitą/znakową, jeśli chcemy, aby wartości były nieujemne. Dzięki temu do zmiennej możemy zapisać dwa razy większą liczbę dodatnią. Nie możemy natomiast zapisać już do niej liczby ujemnej.

Wypisywanie wartości jest bardzo proste w C++. Przykład uważam na tyle prosty, że nie wymaga to żadnego komentarza z mojej strony.

## 4.6. Liczby całkowite - int, short i long

Do tej pory wiesz, iż by przedstawić jakąś wartość całkowitą liczbową należy użyć typu **int**,

```
int i = 6;
```

jednak to nie jedyny typ, który reprezentuje wartość liczbową całkowitą. Do dyspozycji mamy jeszcze dwa inne typy **short**(dawniej short int) i **long**(dawniej long int). Czy one się różnią prezentuje następujący program.

```
//dyrektywy preprocesora-----  
#include <iostream>  
#include <conio.h>  
#include <climits>  
//główny blok programu -----  
int main()  
{  
    using namespace std;  
    int max_int = INT_MAX;  
    short max_short = SHRT_MAX;  
    long max_long = LONG_MAX;  
    //wyświetlenie maksymalnej wartości typów  
    cout << "Maksymalna wartosci int: " << max_int  
    << endl  
    << "Maksymalna wartosci short: " << max_short  
    << endl  
    << "Maksymalna wartosci long: " << max_long  
    << endl;  
    getch();  
    return 0;  
}  
//-----
```

Ktoś mógłby zapytać jaka jest różnica między int a long, a no w nowych systemach operacyjnych nie ma (w DOS była bo int = 32 767). Plik nagłówkowy **<climits>** zawiera stałe symboliczne, który jest przygotowany przez producenta kompilatora. Dzięki niemu możemy sprawdzić jakie są wartości maksymalne i minimalne poszczególnych typów(nie tylko liczbowych).

A co się stanie gdy do wartości maksymalnych dodamy jakąś liczbę ?

```
//dyrektywy preprocesora-----  
#include <iostream>  
#include <conio.h>
```

```

#include <climits>
//główny blok programu -----
int main()
{
    using namespace std;
    // unsigned short liczba = max_short; //BŁĄD ponieważ kompilator nie wie co to max_short
    int max_int = INT_MAX;
    short max_short = SHRT_MAX;
    long max_long = LONG_MAX;
    unsigned short liczba = max_short; // działa bo już się dowiedział

    //wyświetlenie maksymalnej wartości typów
    cout << "Maksymalna wartosci int: " << max_int
    << endl
    << "Maksymalna wartosci short: " << max_short
    << endl
    << "Maksymalna wartosci long: " << max_long
    << endl
    << "Maksymalna wartosci unsigned short: " << liczba
    << endl << endl;

    //dodanie wartości 2 do typów
    cout << "To zobaczymy co sie stanie po dodaniu 2 \n";
    liczba = liczba + 2;
    cout << "Teraz wartosc unsigned short ma " << liczba
    << endl;
    max_short = max_short + 1;
    cout << "Teraz wartosc short ma " << max_short
    << endl;

    //zerowanie typów i odjecie 1
    cout << "\nTo zobaczymy co sie stanie po zerowaniu i odjeciu 1 \n";
    liczba = 0;
    liczba = liczba - 1;
    cout << "A teraz wartosc unsigned short ma " << liczba
    << endl;
    max_short = 0;
    max_short = max_short - 1;
    cout << "A teraz wartosc short ma " << max_short
    << endl;
    getch();
    return 0;
}
//-----

```

Żeby lepiej zrozumieć musimy wiedzieć jakie typy przyjmują [zakresy wartości](#). Gdy do wartości maksymalnej dodam liczbę to powinno spowodować przekroczyć jej wartość. Jednak tak się nie dzieje, działa to na zasadzie licznika kilometrów, po jego przebiciu nie następuje jego zepsucie, tylko licznik liczy od nowa(zera). Analogicznie jest z wartościami minimalnymi i odejmowaniem.

## 4.7. Zmienna typu char a kody ASCII

Kolejną rzeczą, która jest do poruszenia, to kwestia zmiennej **char**. Zmienna char jest rozmiaru 1 bajta. Przechowywany jest w niej pojedynczy znak kodu ASCII z zakresu od (0..255). Aby zapisać **znak** do zmiennej należy podawać go w **pojedynczych** apostrofach. Niektóre znaki są znakami specjalnymi dla C++, dlatego też

trzeba je poprzedzić znakiem `\`. Lista takich znaków będzie w dodatkowym rozdziale. Jeśli chcemy zapisać do zmiennej znak za pomocą kodu ASCII, wystarczy zrobić to w następujący sposób: **char zmienna=50;**. Kod ASCII o numerze **50** odpowiada znakowi **'2'**. Zapis **unsigned char** i zapis **char** jest nam obojętny tak długo, jak nie mamy potrzeby wypisania kodu ASCII tego znaku. Jeśli będziemy chcieli wypisać kod ASCII zmiennej **char** to wartość kodu ASCII, jaką zwróci nam komputer, może nie być taka, jakiej byśmy chcieli się spodziewać. Wszystkie znaki których kod ASCII jest większy od 127, będą zwracały liczby ujemne, zamiast faktycznego numeru kodu ASCII, jaki jest im przyporządkowany. Warto więc posługiwać się zapisem **unsigned char**, zamiast **char**.

## 4.8. Ułamki – czyli float, double, long double

Liczby zmiennoprzecinkowe bo o nich mowa daje nam możliwość zapisana liczb(w postaci ułamka) jak 36,2832; 8,9. Istnieją dwie formy zapisu tego typu liczby:

### 1. Zwykła

36.2832 → należy pamiętać o tym, że używamy kropek( . ), a nie przecinków ( , )

### 2. Naukowa

3.45E6 → zapis **E6** jest równoważny zapisowi 1000 000(czyli 10 do potęgi 6) ,

432.12e-3 → można stosować małą literkę „e”, a minus ( - ) oznacza, że liczba może być ujemna,

-1833.2312e9 → można stosować znak minusa ( - / + ) przed liczbą.

Zakresy tych typów są ograniczone przez mantysę i cechę. Temat nie jest prosty dlatego jedynie wspomnę, iż dzięki temu można uzyskać liczby znacznie większe niż przy typach całkowitych, a do tego posiadające część ułamkową.

Jednak istnieje pewna wada tego typu, a mianowicie dokładność, o to przykład:

```
//dyrektywy preprocesora-----  
#include <iostream>  
#include <conio.h>  
//główny blok programu -----  
int main()  
{  
    using namespace std;  
    float a = 34.23E+9f;//Dodanie na końcu litery f oznacza stałą  
    float b = a + 1.0f;  
  
    cout << "a = " << a  
    << endl  
    << "b = " << b  
    << endl;  
  
    getch();  
    return 0;  
}  
//-----
```

Dlaczego tak się dzieje? Ponieważ dokładność typów na to nie pozwala (temat jest związany z mantysą i cechą dlatego nie będę go omawiał), jednak warto o tym wiedzieć. W kodzie został użyty zapis **34.23E+9f**, gdzie **f** oznacza stałą. Domyślnie zapis 7.23 czy 7.23E8, oznacza stałą typu double. Dla pozostałych typów trzeba na końcu liczby dodać odpowiednio literkę „f” (dla typu float, wielkość literki nie ma znaczenia) i literkę „l” (dla typu long wielkość literki nie ma znaczenia). Przykłady stałych:

5.232E3F → stała typu float,

34.44342E20 → stała typu double

5.4L → stała typu long.

## 4.9. Na koniec trochę o stałych symbolicznych

Przyjrzyjmy się następującemu przykładowi:

```
//dyrektywy preprocesora-----
#include <iostream>
#include <conio.h>
#define ROK 365 //stała symboliczna preprocesora
#define R "R" //stała symboliczna preprocesora
#define O "o" //stała symboliczna preprocesora
#define K "k" //stała symboliczna preprocesora
//główny blok programu -----
int main()
{
    using namespace std;
    const char ODSSTEP = ' '; //stała symboliczna
    const char M = 'm';
    const char A = 'a';
    const char NOWA_LINIA = '\n';

    cout << R << O << K
    << NOWA_LINIA << ODSSTEP
    << M << A << ODSSTEP << ROK << ODSSTEP
    << "dni!"
    << NOWA_LINIA;

    getch();
    return 0;
}
//-----
```

W przykładzie użyto dwóch sposobów zastosowanie stałych symbolicznych **dyrektywa preprocesora #define** i **klasyfikatora const** (które jest lepszym rozwiązaniem!!!). Po co używać stałych symbolicznych? By opisać dane, które nie powinny się zmieniać np. const int STYCZEN = 31 (styczeń ma zawsze 31 dni), lub gdy w kodzie występują kilkakrotnie ta sama dana wartość np.

```
const int LIMIT_KM = 100;
int limit_km_pracownik, limit_km_pracownik1, limit_km_pracownik2, limit_km_pracownik3;

limit_km_pracownik = 50;
limit_km_pracownik1 = 50;
limit_km_pracownik2 = 50;
```

**//chcąc zmienić limit pracownikom na 100 należało by ręcznie wprowadzać**

```
limit_km_pracownik2 = 100;
limit_km_pracownik = 100;
limit_km_pracownik1 = 100;
```

**//zastosowanie stałej**

**//tak wystarczy iż wartość LIMIT\_KM zmienisz na 100 lub na 200 i nie musisz się tym martwić oszczędzasz w ten sposób czas**

```
limit_km_pracownik = LIMIT_KM;
limit_km_pracownik1 = LIMIT_KM;
limit_km_pracownik2 = LIMIT_KM;
limit_km_pracownik3 = LIMIT_KM;
```

Dlaczego **const** jest lepsze niż **#define**, ponieważ określany jest **typ** zmiennej(**int** liczba, **char** znak). Dodatkowo można używać zaawansowanych typów, które dopiero poznasz.

Dobrym rozwiązaniem jest pisanie nazw stałych wielką literą, od razu widać w kodzie gdzie użyto takiej zmiennej.

Należy również pamiętać, iż nie dopuszczalny jest taki zapis:

```
const int liczba;
liczba = 3;//kompilator poinformuje o Błędzie!!!
```

## 4.10. Ćwiczenia

1. Proszę napisać program, który wyświetli na ekranie następujące liczby:

→ 5

→ 8372189379

→ 8,0

→ 73,21

W przykładzie należy użyć odpowiednich typów dla zmiennych.

2. Napisz program, który będzie korzystał tylko ze stałych symbolicznych.

Na ekranie monitora ma się pojawić taki fragment:

```
** Godz. 15:23 **
** 23:15 .Godz **
```

Nie wolno używać zapisów **cout << „Godz.”**; **tylko stałe symboliczne!!!** Dodatkowo pierwszy program napisz używając tylko stałych symbolicznych preprocesora! Natomiast drugi tylko klasyfikatora **const**.

3. Kod ten zawiera kilka błędów, popraw je:

```
//główny blok programu -----
int main()
{
using namespace std;
const STALA;
```

```
int liczba;
char znak;
unsigned short = -5;
float ulamek;
STALA = 10;
liczba = 50,
znak = '!';
cout << "C++ to bardzo ciekawy jezyk" << znak;
cout << endl.
cout << "Moja stala wynoski << STAla;"
cout << endl;
cout << "W programie naliczyłem az << short Bledow";
<< zNak "\n";
"Wszystkiego najlepszego z okazji" << liczba.
<< Urodzin";
getch();
return 0;
}
//-----
```

Uważnie przeanalizuj kod i popraw błędy, może nie od razu ci się to uda.