# C - 1 - Zmienne

Utwórz w Visualu nowy projekt, wykasuj od razu linijkę drukującą na ekranie napis **Hello World**.

## 1.1 Liczby całkowite (int)

W funkcji **main** zadeklarujemy teraz zmienną całkowitoliczbową, przypiszemy do niej wartość i wypiszemy na ekranie.

Dopisz w funkcji **main** (nad instrukcją **return** !!!):

```
int x;     // to jest deklaracja zmiennej całkowitoliczbowej  (deklarujemy zmienną)
x = 7;     // przypisujemy zmiennej wartość 7 ... (inicjujemy zmienną)
printf("Oto moja zmienna: %d \n",x);    // ... i drukujemy na ekranie
```

Skompiluj i uruchom program.

Znaczek **%d** oznacza, że tu, w tym konkretnym miejscu w napisie, trzeba wstawić zmienną całkowitoliczbową, która znajduje się w **printf**ie po przecinku. Dla liczb zmiennoprzecinkowych nie będziemy używać **%d**, lecz innych oznaczeń.

**Zadanie 1:** Zamiast **int x;** wpisz **int x = 7;** i wykasuj poprzednią linijkę, która przypisywała zmiennej siódemkę. Sprawdź, że program działa tak samo – wykonaliśmy tutaj jednoczesną deklarację i inicjalizację zmiennej.

**Zadanie 2:** Sprawdź, czy program skompiluje się, jeżeli w ktorymś miejscu funkcji **main** wpiszesz jeszcze jedną linijkę **int x;**. Oczywiście, że się nie skompiluje – przeczytaj dokładnie błąd, który wyświetli się na dole - najprawdopodobniej będzie to: **error C2086: 'x' : redefinition**

**W obrębie danego zasięgu** (o zasięgu działania zmiennych będzie za chwilę) **można mieć tylko jedną zmienną o danej nazwie.**

Sprawdź, co by się stało, gdyby z programu wykasować wszystkie **int x;** - program nie skompiluje się, ponieważ zmienna **x** nie została zadeklarowana – ponownie przeczytaj błąd, który wyświetli się na dole – powinien brzmieć następująco: **error C2065: 'x' : undeclared identifier**

Zmienne nie mogą nazywać się w pełni dowolnie – nazwa zmiennej może zawierać litery (małe i duże), cyfry oraz znak podkreślenia _ , ponadto nazwa zmiennej musi się zaczynać od litery lub znaku _ (a więc nie od cyfry). Należy unikać zmiennych o nazwie **O** (duże „o") oraz **l** (małe „L"), ponieważ swoim kształtem do złudzenia przypominają cyfry zero i jeden. Ważna jest wielkość liter w nazwie zmiennej - dla kompilatora **int x;** oraz **int X;** to dwie różne zmienne.

**Zadanie 3:** Sprawdź, że możesz zadeklarować zmienne **int x;** oraz **int X;** i program skompiluje się prawidłowo.

Częstym błędem jest pisanie spacji w nazwie zmiennej, np: **int moja zmienna;**.

**Zadanie 4:** Sprawdź, że zapis **int moja zmienna;** wywoła przy próbie skompilowania aż dwa rodzaje błędów:
**error C2146: syntax error : missing ';' before identifier 'zmienna'**
**error C2065: 'zmienna' : undeclared identifier**

Nie trzeba każdej zmiennej tego samego typu deklarować osobno, można to zrobić w jednej instrukcji.

**Zadanie 5:** Sprawdź, że możesz napisać w programie coś takiego: **int a,b,c;** a nawet **int a=62, b=78;**.

**Zadanie 6:** Sprawdź, że za pomocą znaczków **%x** oraz **%o** można drukować liczby całkowite w systemie szesnastkowym i ósemkowym, nawet jeżeli nadano im wartość w systemie dziesiętnym.

**Zadanie 7:** Sprawdź, że jeżeli chcesz wydrukować sam znaczek **%**, to w **printf**ie musisz napisać **%%**.

**Zadanie 8:** Utwórz nad **printf**em zmienną całkowitoliczbową **y** i przypisz jej od razu wartość **5**. Zaraz pod **printf**em wpisz nowego **printf**a:

```
printf("Oto moje dwie zmienne: %d oraz %d \n",x,y);
```

**Bardzo częstym błędem jest zapominanie o przecinkach w printfie.**

Skompiluj i uruchom program. Zauważ, że zmienne są wydrukowane w takiej kolejności, w jakiej występują po przecinku w drugim **printf**ie.

**Zadanie 9:** W drugim **printf**ie, po przecinku zamień miejscami **x** oraz **y**. Zobacz rezultaty.

**Zadanie 10:** W dodanym przed chwilą **printf**ie dopisz w odpowiednim miejscu jeszcze jedno **%d**. Po **y** dopisz jeszcze jeden **przecinek** oraz **x+y**. Zauważ, że w **printf**ach można też wpisywać operacje arytmetyczne, a nie tylko „gotowe" zmienne.

**Zadanie 11:** Zaraz pod **printf**ami dopisz linijkę **x=x+2;** pod tą linijką wydrukuj jeszcze raz na ekran wartość zmiennej **x** (powinno wyjść 9).

Instrukcja **x=x+2;** zwiększyła o dwa wartość zmiennej **x**.

**Studenci nagminnie wpisują x=2; jeżeli chcą zwiększyć wartość zmiennej, a przecież taka instrukcja przypisuje dwójkę do zmiennej x zamiast zwiększyć jej wartość :(**

**Zadanie 12:**
**1)** Zamiast instrukcji **x=x+2;** wpisz instrukcję **x+=2;**. Sprawdź, że program działa tak samo.
**2)** Sprawdź, jak się zmieni wartość zmiennej **x** po wpisaniu instrukcji **x -= 2;** lub **x=x-2;**.
**3)** Sprawdź jak zadziała instrukcja **x++;**.
**4)** Sprawdź jak zadziała instrukcja **x--;**.
**5)** Sprawdź, czy można wykonać operacje: **x+=y;** , **x-=y;** , **x*=y;** i jak zmienią one wartość zmiennej **x**.

Instrukcja **x++;** zwiększa wartość zmiennej **x** o jeden. Instrukcja **x--;** zmniejsza wartość zmiennej **x** o jeden. Istnieją też instrukcje **- -x;** oraz **++x;** ale zapoznamy się z nimi bliżej podczas omawiania pętli.

## 1.2 Liczby zmiennoprzecinkowe (float i double)

Jeżeli potrzebujemy zmiennych dla wartości ułamkowych, to możemy skorzystać z typów:
**float -** mniejsza dokładność, znaczek **%f**
**double** – większa dokładność, znaczek **%lf** (<- to nie jest pionowa kreska ani jedynka, tylko litera „**małe L**” oraz **f**)

**Zadanie 13:** Zadeklaruj kilka zmiennych typu **float** i **double**, przypisz im jakieś wartości niecałkowite, np. **2.3**, i wyświetl je. Sprawdź, co się stanie, jeżeli podczas drukowania zmiennej zamiast **%lf** wpiszesz **%d** (zwykle wyświetla się jakaś zupełnie bzdurna wartość).

Zauważ, że Visual zwróci warninga dla liczby typu **float** – poinformuje, że musiał specjalnie zaokrąglić liczbę typu **double**, by utworzyć liczbę typu **float**. W Visualu lepiej korzystać z liczb typu **double**.

**Zadanie 14:** Dopisz w funkcji **main** linijki:

```
double ulamek = 1/3;
printf("Jedna trzecia: %lf \n",ulamek);
```

Skompiluj i uruchom program, najprawdopodobniej wyświetli się .... zero.
Teraz popraw **1/3** na **1.0/3.0** i jeszcze raz uruchom program, zauważ różnicę.
Widząc 1/3 Visual pomyślał, że ma podzielić przez siebie dwie liczby całkowite i że wynik też ma być całkowity.
Właśnie dlatego należy używać liczb z kropkami.
Wykonaj podobny eksperyment w przypadku, gdy tylko jedna z liczb jest z kropką.

**Studenci nagminnie piszą przecinek zamiast kropki w liczbach zmiennoprzecinkowych i dziwią się, że program się nie kompiluje :(**

Jeżeli nie chcemy, by wyświetlana liczba była zbyt „długa”, to możemy nakazać wyświetlenie tylko określonej liczby miejsc po przecinku, np:

```
printf("Jedna trzecia: %2.3lf \n",ulamek);  // co najmniej 2 miejsca przed kropką i najwyżej 3 po kropce
```

**Zadanie 15:** Sprawdź, czy dla zmiennych typu **double** też działają operacje ++, --, +=, itd ....

**Zadanie 16:** Wpisz w funkcji **main** instrukcje:

```
int t = 3;
double d = 0.234;
t = t + d;
```

Wydrukuj na ekran wartość zmiennej **t** – mimo dodania **0.234** wartość nadal będzie wynosić **3**, ponieważ **t** jest liczbą całkowitą. Zmień wartość zmiennej **d** na **1.234** i powtórz eksperyment.
Przywróć zmiennej **d** poprzednią wartość i wykonaj podobny eksperyment z mnożeniem.

## 1.3 Zmienne znakowe i boolowskie (char, bool)

Zmienne typu **char** przechowują znaki. Przykład użycia:

```
char znaczek = 'A';   // tu są apostrofy, a nie cudzysłowy
printf("Moj znak jako znak: %c \n",znaczek);
printf("Moj znak jako kod ASCII: %d \n",znaczek);
```

**Zadanie 17:** Zadeklaruj kilka zmiennych typu **char**. Zbadaj jaki kod ASCII mają poszczególne litery Twojego imienia (małe i duże litery).

**Studenci nagminnie mylą typ char z typem napisowym i próbują wpisywać między apostrofy kilka znaków. Próbują też stosować cudzysłowy zamiast apostrofów przy zmiennych typu char :(**

**Zadanie 18:** Spróbuj pododawać, pomnożyć, poodejmować wartości dwóch zmiennych, z których jedna jest typu **char**, a druga typu **int** lub **double**. Spróbuj też wyświetlić parę zmiennych typu **int** jako znaki (czyli **%c**).

Zmienne boolowskie mają tylko dwa rodzaje wartości: **true** oraz **false**.
Przykład deklaracji:

```
bool czyTak = true;      // true i false piszemy małą literą
bool czyToPrawda = false;
```

**Zadanie 19:** Wyświetl na ekranie zawartość powyższych zmiennych korzystając z **%d**.

## 1.4 Rzutowanie

Czasem chcemy, by liczba całkowita stała się na chwilę liczbą zmiennoprzecinkową (tak było przy obliczaniu 1/3).

**Zadanie 20:** Wpisz w funkcji **main**:

```
double ddd = (double)1/(double)3;   // to jest właśnie rzutowanie (na inny typ)
printf("Znowu jedna trzecia: %lf \n",ddd);
```

Można też oczywiście rzutować liczby zmiennoprzecinkowe na całkowite:

**Zadanie 21:** Wpisz w funkcji **main**:

```
double eksper = 17.63456;
printf("kolejny ekperyment - bez rzutowania: %d \n",eksper);
printf("i z rzutowaniem : %d \n",(int)eksper);
```

Chcemy wydrukować liczbę zmiennoprzecinkową jako całkowitą

Bardzo częstym błędem jest pisanie:
int(eksper)
zamiast:
(int)eksper

## 1.5 Instrukcje warunkowe / Zasięg zmiennych, zmienne globalne i lokalne

Instrukcje warunkowe zostaną szczegółowo omówione w następnym materiale. Teraz zostanie przedstawiona najprostsza wersja instrukcji warunkowej:

```
if([warunek logiczny]){
        [instrukcja 1]
        [instrukcja 2]
        ....
        [instrukcja n]
} else if([jakiś inny warunek]){
        [tu jakieś inne instrukcje 1]
        ...
        [tu jakieś inne instrukcje n]
} else{ // else bez ifa oznacza „a jeżeli żaden z powyższych warunków nie zachodzi ...”
        [jeszcze inne instrukcje ...]
}
```

Tych else-ifów może być dowolnie dużo, a nawet zero :)

else bezifowy może być tylko jeden i tylko na końcu.

Jeżeli po **if**ie jest tylko jedna instrukcja, to można pominąć klamerki:

```
if([warunek logiczny])
        [instrukcja 1]
```

Nie należy to jednak do dobrego stylu programowania – jeżeli zawartość **if**a się rozrośnie, a programista zapomni dopisać klamerki, to po **if**ie wykona się tylko pierwsza instrukcja, zaś pozostałe wykonają się zawsze, np.:

```
if([warunek logiczny])
        [instrukcja 1]     // <- ta instrukcja wykona się, jeżeli warunek logiczny będzie spełniony
        [instrukcja 2]     // pozostałe instrukcje nie należą już do ifa z powodu brakujących klamerek ...
        ....               // przez co wykonają się zawsze :(
        [instrukcja n]
```

**Studenci nagminnie zapominają dopisywać nawiasów klamrowych po if**ie **:(**

Załóż nowy projekt, wykasuj od razu linijkę drukującą napis **Hello world**.
U góry zaincluduj bibliotekę **math.h** – będziemy korzystać z funkcji **sqrt**, która oblicza pierwiastek.

Napiszemy program, który oblicza pierwiastki równania kwadratowego.

**Zadanie 22:** Wpisz w funkcji **main**:

```
double a,b,c;
a=1;
b=5;
c=6;
double delta = b*b - 4*a*c;
if(delta < 0){
        printf("Równanie nie ma pierwastków \n");
}
else if(delta == 0){
        double x0 = -b/(2.0*a);
        printf("Równanie ma jeden pierwastek: %lf \n",x0);
}
else{
        double x1 = (-b-sqrt(delta))/(2.0*a);
        double x2 = (-b+sqrt(delta))/(2.0*a);
        printf("Równanie ma dwa pierwastki: %lf oraz %lf \n",x1,x2);
}
```

Studenci ciągle próbują pisać b^2 (b do drugiej) bo tak jest w Mathematice, więc według nich zadziała to wszędzie :(

Studenci często zapominają o gwiazdkach w mnożeniu i piszą 4ac – a przecież taki zapis nic nie znaczy !

Taaaak, tu jest == a nie =

Zwróć uwagę na przepiękne wcięcia w kodzie programu, które ułatwiają jego zrozumienie :)

1) Poeksperymentuj z innymi wartościami zmiennych **a**, **b** oraz **c**.

2) Podczas obliczania dwóch pierwiastków, x1 oraz x2, mamy dobry przykład umieszczania nawiasów w skomplikowanym wyrażeniu arytmetycznym: **(-b+sqrt(delta))/(2.0*a)**
Większość studentów napisałoby **(-b+sqrt(delta))/2.0*a -** w takim zapisie zmienna **a** byłaby pomnożona przez cały ułamek, czyli przez licznik – usuń nawiasy w mianowniku i sprawdź, że program nie obliczy poprawnie pierwiastków.

3) Zauważ, że w drugim **if**ie występuje == - jeżeli coś porównujemy, to zawsze korzystamy z ==, ponieważ jedno = przypisuje tylko wartość.
Zapis **if(delta = 0)** oznaczałby tyle, co „jeżeli do delty **uda się przypisać** zero" – a zawsze się uda, bo przecież typ delty jest liczbowy i wpisanie w deltę zera nigdy nie spowoduje błędu.

**Studenci nagminnie używają jednego = w porównaniach i potem dziwią się, że warunek jest zawsze spełniony :( - jest to najczęstszy błąd świata.**

Mamy następujące porównania:

| | |
|---|---|
| == | równe |
| != | różne |
| >= | większe równe (częstym błędem jest pisanie =>) |
| <= | mniejsze równe (a tego jakoś nikt nie przekręca) |
| > | większe |
| < | mniejsze |

Wykrzyknik oznacza zaprzeczenie – sprawdź, że jeżeli napiszemy warunek:

```
if(  !(delta >=0)  ){ //coś-tam...
```

to program zadziała tak samo dobrze :)

4) Czasem chcielibyśmy zastosować kombinację warunków – służą do tego operatory logiczne:

| | |
|---|---|
| && | **and** (ma pierwszeństwo przed **or**em) |
| \|\| | **or** |
| ! | **negacja** też jest operatorem |

Można jeszcze skomplikować warunek w pierwszym **if**ie: wpisz **!(delta>0 || delta==0)**
Spróbuj zagmatwać w podobny sposób warunek w drugim **if**ie używając operatorów **!** oraz **&&**.

(*) **Ciekawostka:** w języku PHP istnieje porównanie === i oprócz porównania wartości sprawdza też, czy zmienne są tego samego typu :)

**If**y są doskonałym przykładem ilustrującym zasięg zmiennych:

**Zadanie 23:** Na dole programu, nad **return**em spróbuj ponownie wydrukować **x1** i **x2** – program nie skompiluje się, ponieważ te dwie zmienne zostały zdefiniowane wewnątrz **if-else** i „umierają" wraz z nawiasem klamrowym po tym **if-els**ie – po prostu nie istnieją dalej.

**Zadanie 24:** Teraz wykasuj oba słowa **double** z **else**a (zostaw tylko przypisanie wartości do zmiennych), na górze programu jako pierwszą instrukcję funkcji **main** wpisz **double x1,x2;** i skompiluj program – teraz program da się już skompilować, ponieważ **x1** i **x2** „żyją" aż do klamerki domykającej funkcję **main**.

W baaaaardzo dużym uproszczeniu: **zmienne „żyją" aż do nawiasu klamrowego zamykającego ich zasięg.**

Istnieją też tzw. **zmienne globalne**. Przypuśćmy, że chcemy z jakiejś zmiennej korzystać często, przez cały czas trwania programu, we wszystkich funkcjach. Wówczas deklarujemy taką zmienną nad funkcją **main** np.:

```
#include "stdafx.h"

int mojaZmiennaGlobalna;

int main(int argc, char* argv[])
{
 ....
```

Jeżeli zmienna globalna jest wykorzystywana przez wiele funkcji, to istnieje ryzyko, że jej wartość zostanie niechcący zmieniona. Należy więc raczej unikać używania zmiennych globalnych i przekazywać wartości zmiennych jako argument funkcji. Jeżeli zmienna nie jest globalna, a więc została zadeklarowana wewnątrz jakiejś funkcji, to jest ona **zmienną lokalną**.

## 1.6 Pobieranie danych z klawiatury

**Zadanie 25:** Przywróć program obliczający pierwiastki do początkowego stanu. Wykasuj linijki przypisujące zmiennym wartości 1, 5 oraz 6 i zamiast nich dopisz:

```
printf("Podaj a i naciśnij enter:");
scanf("%lf",&a);
printf("\nPodaj b i naciśnij enter:");
scanf("%lf",&b);
printf("\nPodaj c i naciśnij enter:");
scanf("%lf",&c);
```

Przetestuj program, następnie sprawdź, że zamiast tych linijek zadziała też:

```
printf("Podaj a, b, c  oddzielone spacjami i naciśnij enter:");
scanf("%lf %lf %lf",&a,&b,&c);
```

**W funkcji scanf podajemy w cudzysłowiu znaczek adekwatny do typu zmiennej, w której będziemy przechowywać wartość (tutaj: %lf, bo chcemy otrzymać liczbę typu double). Następnie po przecinku wpisujemy znaczek & oraz nazwę zmiennej.**

**Znaczek & oznacza, że podajemy adres (w pamięci komputera) zmiennej, a nie jej wartość –chcemy pobrać liczbę i umieścić ją pod wskazanym adresem.**

**Zadanie 26:** Przekształć program obliczający pierwiastek kwadratowy następująco:

Po podaniu przez użytkownika współczynników **a**, **b** oraz **c** poproś go, by podał (zgadł) liczbę pierwiastków – należy więc wyświetlić **printf**a z odpowiednim zapytaniem i odczytać z klawiatury liczbę pierwiastków do jakiejś zmiennej typu **int** (jakiej literki użyjesz po **%** ?). Dopisz w każdym **if**ie wewnętrznego **if**a sprawdzającego, czy liczba pierwiastków podana przez użytkownika jest prawidłowa – jeżeli jest prawidłowa, wyświetl jakiś komunikat pochwalny. Jeżeli nie jest prawidłowa, wyświetl komentarz naganny.

**Zadanie 27:** Sprawdź, co się stanie, jeżeli poprosisz o liczbę całkowitą, a użytkownik poda liczbę z kropką. Po pobraniu z klawiatury wyświetl zawartość swojej zmiennej całkowitoliczbowej.

## 1.7 Stałe

Jeżeli w programie będziesz często używać jakiejś stałej, np. liczby $\pi$, a nie chce Ci się co chwilę pisać 3.14159..., to możesz pod includami a nad funkcją **main** wpisać:

```
#define pi 3.1415926537497
```

a następnie korzystać z **pi** jak ze zwykłej zmiennej.

**Zadanie 28:** Napisz program, który zawiera powyższą definicję zmiennej **pi**. Program powinien pytać użytkownika o promień (najlepiej typu **double**) oraz wyświetlić małe menu typu:
*Jeżeli chcesz obliczyć pole koła, naciśnij **p**.*
*Jeżeli chcesz obliczyć objetość kuli, naciśnij **o**.*
(opcję użytkownika pobierzemy do zmiennej typu **char**, gdyż będzie to literka).
Następnie, w zależności od wyboru oblicz daną wartość i wyświetl ją na ekranie.
Jeżeli masz problemy z działaniem menu, poeksperymentuj z funkcją **getchar()** z biblioteki **conio.h**.

## 1.8 Zmienne ze znakiem i bez / Rozmiar pamięci dla poszczególnych typów

Oprócz podstawowych typów omówionych powyżej, mamy też typy modyfikowane słowem kluczowym **signed / unsigned** oraz **short** i **long**. **Signed** (ang. podpisany, oznakowany) oznacza, że zmienna tego typu ma znak +/-. **Unsigned** oznacza brak znaku. Jeżeli nie jest użyty modyfikator, to domyślnie zmienna jest zazwyczaj typu **signed**.

Dla kompilatorów 32- i 64-bitowych Visual rozróżnia typy:

| Nazwa typu | Bajty | Nazwy zastępcze | Zakres wartości |
|---|---|---|---|
| Int | 4 | signed | –2,147,483,648 do 2,147,483,647 |
| unsigned int | 4 | unsigned | 0 do 4,294,967,295 |
| __int8 | 1 | char | –128 do 127 |
| unsigned __int8 | 1 | unsigned char | 0 do 255 |
| __int16 | 2 | short, short int, signed short int | –32,768 do 32,767 |
| unsigned __int16 | 2 | unsigned short, unsigned short int | 0 do 65,535 |
| __int32 | 4 | signed, signed int, int | –2,147,483,648 do 2,147,483,647 |
| unsigned __int32 | 4 | unsigned, unsigned int | 0 do 4,294,967,295 |
| __int64 | 8 | long long, signed long long | –9,223,372,036,854,775,808 do 9,223,372,036,854,775,807 |
| unsigned __int64 | 8 | unsigned long long | 0 do 18,446,744,073,709,551,615 |
| Bool | 1 | brak | false lub true |
| Char | 1 | brak | –128 do 127 (0 do 255 przy opcji kompilowania /J) |
| signed char | 1 | brak | –128 do 127 |
| unsigned char | 1 | brak | 0 do 255 |
| Short | 2 | short int, signed short int | –32,768 do 32,767 |
| unsigned short | 2 | unsigned short int | 0 do 65,535 |
| Long | 4 | long int, signed long int | –2,147,483,648 do 2,147,483,647 |
| unsigned long | 4 | unsigned long int | 0 do 4,294,967,295 |
| long long | 8 | brak | –9,223,372,036,854,775,808 do 9,223,372,036,854,775,807 |
| unsigned long long | 8 | brak | 0 do 18,446,744,073,709,551,615 |
| Float | 4 | brak | 3.4E +/- 38 (7 cyfr) |
| Double | 8 | brak | 1.7E +/- 308 (15 cyfr) |
| long double | tak jak double | brak | tak samo jak double |
| wchar_t | 2 | __wchar_t | 0 do 65,535 |

**Zadanie 29:** Zadeklaruj zmienną typu **signed char** i przypisz jej wartość 126. Następnie dodaj do tej zmiennej piątkę i wyświetl na ekranie wartość tej zmiennej. Zauważ, że przekroczono zakres – wyświetlana wartość jest ujemna.
Powtórz ten eksperyment ze zmienną typu **unsigned char**.
Następnie spróbuj przypisać do zmiennej typu **unsigned char** wartość 252 i dodaj do zmiennej 10. Wyświetl wartość zmiennej na ekranie.

Oczywiście nikt przy zdrowych zmysłach nie zna na pamięć takiej tabelki typów. Jeżeli chcemy się dowiedzieć, ile bajtów zajmuje dany typ, możemy napisać np.:

```
printf("Typ char zajmuje %d bajtów", sizeof(char));
```

Zauważ, że w tym **printf**ie zawarliśmy po przecinku wywołanie funkcji. Dotychczas wpisywaliśmy tam tylko zmienne, a w jednym przykładzie sumę zmiennych.

**Zadanie 30:** W podobny sposób zbadaj „objętość” pięciu innych typów danych z powyższej tabelki.

**Zadanie 31:** Wpisz w funkcji **main** linijki:

```
short int zmienna;
printf("\nTyp short int zajmuje %d bajtów", sizeof(zmienna));
```

Skompiluj i uruchom program. Zauważ, że funkcja **sizeof** może też przyjmować nazwy zmiennych, a nie tylko nazwy typów.
Przy okazji zwróć uwagę na warning, który się pojawił:

```
warning C4101: 'zmienna' : unreferenced local variable
```

Visual zwraca nam uwagę, że zadeklarowaliśmy zmienną, z której nigdzie nie korzystamy - samo wyświetlenie nie jest, niestety, uważane za wykorzystanie zmiennej :|

## 1.9 Najczęstsze błędy

Próba używania zmiennej bez jej zadeklarowania.
Próba deklarowania wielu zmiennych o tej samej nazwie.
Przy wypisywaniu na ekran: stosowanie **%** i litery nieadekwatnej do typu zmiennej.
Pisanie wartości ułamkowych po przecinku, a nie po kropce.
Brak umiejętności zwiększenia wartości zmiennej.
Pomijanie przecinków w **printf**ach, które drukują wartość zmiennych.
Próba pisania znaków typu **char** w cudzysłowach, np. "A".
Próba umieszczania kilku znaków w jednej zmiennej typu **char**, np. 'ABC'.
Złe umieszczanie nawiasów podczas rzutowania.
Pomijanie nawiasów klamrowych w **if**ach i **else**ach.
**Pisanie = zamiast == w porównaniach.**
Niestosowanie nawiasów w wyrażeniach arytmetycznych, zwłaszcza w mianownikach ułamków.
Stosowanie nawiasów **{**, **}**, **[**, **]** w wyrażeniach arytmetycznych – a można tam stosować tylko **(** i **)**.
Nadużywanie zmiennych globalnych.
Zapominanie o znaczku **&** w funkcji **scanf**.
**Brak wcięć w kodzie programu, nieczytelny kod.**

## 1.10 Zadanie z gwiazdką

Wpisz w funkcji **main**:

```
int w = 7;
int z = 5;
// tu wpisz instrukcje, które zamienią wartości miejscami tak, by w w było 5, a w z było 7
// NIE WOLNO PRZYPISAĆ TYCH WARTOŚCI „NA SZTYWNO", trzeba „pobawić się" zmiennymi
// na końcu wydrukuj wartości zmiennych na ekranie
```

## 1.11 Quiz

1. Zaznacz w tabelce, które znaczki w **printf**ie nie wydrukują błędnie wartości zmiennych danego typu.

| | int | char | float | double |
|---|---|---|---|---|
| %d | | | | |
| %x | | | | |
| %o | | | | |
| %c | | | | |
| %f | | | | |
| %lf | | | | |

2. Które z poniższych instrukcji zmienią wartości zmiennej o jeden? Podkreśl na czerwono instrukcje, które nie dadzą się skompilować.
   **a)** x+=1; **b)** x=x++; **c)**+x+; **d)** x=1++; **e)** x=x+1; **f)** x++;

3. Która z deklaracji zmiennej wywoła błąd kompilatora?
   **a)** int zmienna1; **b)** int zmienna 1; **c)** int _zmienna1; **d)** int Zmienna1;
   Jaki będzie brzmiał ten błąd? _____

4. Jaki błąd wypisze się w dolnym oknie, jeżeli zadeklarujesz dwie zmienne o jednakowej nazwie?
   _____

5. Jaki błąd wypisze się w dolnym oknie, jeżeli nie zadeklarujesz zmiennej, z której chcesz korzystać?
   _____

6. Jeżeli wydrukowaliśmy zawartość zmiennej za pomocą **%x** i na ekranie pojawiła się liczba **23**, to jaką wartość ma ta zmienna w systemie dziesiętnym? _____

7. Ile znaczków musimy zużyć, żeby wydrukować znak procenta? _____

8. Która instrukcja ogranicza prawidłowo liczbę cyfr drukowanych po kropce? Popraw błędne instrukcje na czerwono.
   **a)** %2.2lf **b)** %.3lf **c)** 3.3%lf **d)** 3.%4.lf **e)** %2.0lf

9. Co wydrukuje fragment kodu:
```
double i = 7.2;
printf("w tym roku inflacja wyniesie %1.1lf %%, a w nastepnym %1.2lf %%",i,i+3.21);
```
**a)** w tym roku inflacja wyniesie 7.2 %%, a w nastepnym 10.41 %%
**b)** w tym roku inflacja wyniesie 7.2 %, a w nastepnym 10.4 %
**c)** w tym roku inflacja wyniesie 7.2 %, a w nastepnym 10.41 %
**d)** w tym roku inflacja wyniesie 7.20 %, a w nastepnym 10.41 %

10. Jaka litera jest zawarta w zmiennej **char**owej **x**, jeżeli instrukcja **printf("%d",x);** wydrukowała liczbę **122**?
_____

11. Jaka liczba jest zawarta w zmiennej **int**owej **x**, jeżeli instrukcja **printf("%c",x);** wydrukowała literę **K**?
_____

12. Która z poniższych instrukcji zawiera poprawne rzutowanie? Popraw błędne instrukcje na czerwono.
**a)** k(int) **b)** int(k) **c)** (int k) **d)** (int)k **e*)** (int)(k)

13. Która z poniższych instrukcji prawidłowo przypisuje wartość do zmiennej **char**owej? Popraw błędne instrukcje na czerwono.
**a)** char znak = A; **b)** char znak = 'A'; **c)** char znak = "A"; **d)** char znak = 'AA';

14. Jak zachowa się poniższy kawałek kodu? Które instrukcje wykonają się po **if**ie, a które zawsze i dlaczego?
```
int k=23;
if(k%2==0)    // % zwraca resztę z dzielenia, w tym przypadku przez 2
        printf("Liczba %d jest parzysta\n",k);
        printf("Gdyby nie byla parzysta, to bylaby nieparzysta :P");
```
_____
_____

15. Dlaczego poniższy kawałek kodu nie skompiluje się? Zaznacz błędy na czerwono.
```
int k=23;
if(k%2==0)
        printf("Liczba %d jest parzysta\n",k);
        printf("Gdyby nie byla parzysta, to bylaby nieparzysta :P");
else
        printf("Liczba %d jest nieparzysta\n",k);
        printf("Gdyby nie byla nieparzysta, to bylaby parzysta :P");
```
_____
_____

16. Jaka biblioteka przechowuje funkcje matematyczne? _ _ _ _ . h

17. Podaj nazwę funkcji obliczającej pierwiastek. _ _ _ _

18. Gdzie deklarujemy zmienne globalne?
_____
_____

19. Która pobranie z klawiatury zadziała prawidłowo? Popraw na czerwono instrukcje, które nie zadziałają.
**a)** scanf("&d",%zmienna); **b)** scanf("%d",&zmienna); **c)** scanf("&zmienna");

20. Co się stanie, jeżeli zamiast liczby całkowitej podasz z klawiatury liczbę zmiennoprzecinkową?
_____
_____

21. Gdzie i jak definiujemy stałe? Podaj przykład.
_____
_____
_____
_____

22. Jaka funkcja zwraca liczbę bajtów zajmowanych przez zmienną danego typu? _ _ _ _ _ _
Jakie typy argumentów może pobierać taka funkcja?
_____ lub_____

23. Który **print**f da się skompilować? Zaznacz na czerwono błędy w nieprawidłowych **printf**ach.
**a)** printf("Wartosc zmiennej wynosi %d",int zmienna);
**b)** printf("Wartosc zmiennej wynosi %d",zmienna);
**c)** printf("Wartosc zmiennej wynosi %d"+zmienna);
**d*)** printf("Wartosc zmiennej wynosi %d",&zmienna);