Sprawozdanie podstawy programowania – zmienne

Język C umożliwia przechowywanie danych i operowanie na nich za pomocą zmiennych. Przeprowadzanie operacji matematycznych zachodzi za pomocą operatorów (*,+,-,%,/, …), które mają swoje priorytety. Kolejność wykonywania działań jest zbliżone do zasad, które zachodzą w matematyce, priorytet danego działania można ustalić za pomocą nawiasów. Typ zmiennej całkowitej (int) charakteryzuje się tym, że niemożliwe jest zapisanie "w nim" wartości zmiennoprzecinkowej. Zatem po podstawieniu pod int b = 5/2, b będzie przechowywało wartość 2 (zawsze zaokrąglamy w dół).

Za pomocą "#define NAZWA wartość", można przypisać dowolnie wybranemu wyrażeniu daną wartość, kompilator czytając zdefiniowane wyrażenie automatycznie zastąpi je wybraną wartością.

Wartości zmiennoprzecinkowe można przechowywać za pomocą typów float oraz double. Float jest w stanie przechować liczby z dokładnością do 6 cyfr znaczących. Większą pojemność posiada typ double – 15 cyfr znaczących. W przypadku próby zapisania do wspomnianych typów liczb przekraczających dopuszczone normy, kompilator "utnie" nadmiar. Podczas wywoływanie funkcji printf przy dopisaniu "20.15" przed zmienną można uzyskać jej dokładny obraz.

Przyczyna utraty precyzji zmiennych float i double:

Łatwo zauważyć, że kiedy przypiszemy do zmiennej float lub double wartość zmiennoprzecinkową, to prawdziwa wartość zmiennej będzie się różniła od stanu pożądanego o bardzo niewielką wartość, np. (float f = 0.2, zmienna f może mieć wartość 0.200000000001). Wynika to ze sposobu zapisu liczb zmiennoprzecinkowych przez komputer, który posługuje się liczbami binarnymi. W przypadku zmiennej typu float, która zajmuje 32 bity w pamięci na zapis wartości po przecinku są przeznaczone 23 bity (mantysa), za jej pomocą można jedynie wyznaczyć **przybliżoną** wartość. Z każdym kolejnym bajtem float f będzie dążyć do 0.2, aczkolwiek nigdy nie osiągnie takiej wartości ze względu na charakterystykę systemu binarnego. Podobny przypadek można zaobserwować w systemie dziesiętnym, gdy chcemy wyznaczyć wartość liczby niewymiernej, np. π lub $\sqrt{2}$

Zmienna typu bool przechowuje dwie wartości 1 (true) oraz 0 (false). Za jej pomocą łatwo sprawdzić tautologie wyrażeń.

Typy danych można konwertować, przykładowo, jeżeli (double n= 2.7;) podstawimy pod float f (f = (int)n;), to kompilator przekonwertuje n na liczbę całkowitą (2) i zapisze taką wartość w f. Jest to jawna konwersja typu. Przykład konwersji niejawnej: (int j = n;) w takiej sytuacji kompilator automatycznie przekonwertuje 2.7 -> 2.0.

Operacja n=m+++j zapisze w zmiennej n wynik wyrażenia m+j, a następnie dokona inkrementacji m. n=++m+++j wpierw zinkrementuje m, następnie zapisze w n (m+1)+j i ponownie zinkrementuje m.

Zapisanie zmiennej w następujący sposób (float fx1 = 1.23e7) jest jednoznaczne z zapisem fx1 = 1.23*10^7; fx2 = 1.23e-7 (1.23*10^(-7). W przypadku przypisania tak małych wartości do zmiennych typu float, a nawet double będzie skutkowało pewną niedokładnością w obliczeniach. fx2 zawiera w sobie tak minimalną wartość, że kompilator potraktuje ją jako równą zeru. Dzięki czemu równanie (fx1+fx2) - fx1 == 0 będzie spełnione. W takim przypadku w momencie zapisania działania 1/((fx1+fx2) - fx1) będzie tożsame z dzieleniem przez zero, co jest niedopuszczalne. Można temu zapobiec definiując ustaloną tolerancję (np. #define TOLERANCE 1e-7), a następnie sprawdzić czy nasze wyrażenie jest mniejsze od zdefiniowanej wartości.

Własny kod i screeny z konsoli

1. STANDARDOWE DEKLARACJE-DEFINICJE I WYPISYWANIE WARTOŚCI ZMIENNYCH

```
int z1 = 10;
    unsigned int a1 = 12345;
    char b1 = 'a';
    double c1 = 10.1;
    float d1 = 11;
    long double e1 = 20.123456789;
    printf("\nint: %d\nunsigned int: %u\nchar:
%c\ndouble: %lf\nfloat: %f\nlong double: %Lf\n ",
        z1, a1, b1, c1, d1, e1);
```

```
int: 10
unsigned int: 12345
char: a
double: 10.100000
float: 11.000000
long double: 20.123457
```

2. PROSTE OPERACJE

```
int wynik1 = m * n + o, wynik2 = m * (n + 0);
printf("wynik 1:%d\nwynik 2: \%d", wynik1, wynik2);
```

```
wynik 1:139
wynik 2: 119
```

```
int inkr = n + p++, inkr2 = ++p + n;
printf("inkr: %d\ninkr2: %d", inkr, inkr2);
```

```
inkr: 14
inkr2: 16
```

```
int m1 = 10, n1 = m1 % 3, o1 = m1 / 3;
printf("\nm1/3 = %d\nreszta n%3: %d\n m1 = (m1/3)*3 +
(m1%3) = %d\n", m1 / 3, n1, o1 * 3 + n1);
```

```
m1/3 = 3
reszta n: 1
 m1 = (m1/3)*3 + (m1) = 10
```

3. DEFINICJE STAŁYCH SYMBOLICZNYCH

```
#define trzyipol 3.5
float r12 = trzyipol;
printf("zmienna zdefiniowana trzyipol %f\n", r12);
```

```
zmienna zdefiniowana trzyipol 3.500000
```

4. ZMIENNE ZMIENNOPRZECINKOWE: FLOAT I DOUBLE

```
float x1 = 10, y1 = 3;
double x11 = 10, y11 = 3;
printf("float 10/3 = %20.15f\ndouble 10/3 =
%20.15lf\n", x1 / y1, x11 / y11);
```

```
float 10/3 =    3.333333253860474
double 10/3 =    3.333333333333333
```

5. OPERATORY RELACJI I WARTOŚCI LOGICZNE

```
bool p1 = 1, q1 = 0, r1 = 1;
if (((p1 || q1) && r1) == ((p1 && r1) || (q1 && r1)))
{
    printf("\ntautologia 1 sie zgadza\n");
}
if (((p1 && q1) || r1) == ((p1 || r1) && (q1 || r1)))
{
    printf("\ntautologia 2 sie zgadza\n");
}
```

```
tautologia 1 sie zgadza

tautologia 2 sie zgadza
```

6. Operatory oraz niejawne i jawne konwersje typów

```
double hd = 2.5;
float hf = 0.5;
double hdd = (int) hd / hf, hddd = hd / hf;
printf("(int)2.5/0.5: %lf\n 2.5/0.5: %lf\n", hdd, hddd);
```

```
(int)2.5/0.5: 4.000000
 2.5/0.5: 5.000000
```

7. DEFINICJE STAŁYCH SYMBOLICZNYCH (CD.)

```
double t1 = jednaczwartafloat * 4.0;
double t2 = jednaczwarta * 4.0;
printf("\nfloat: %20.15lf\n double: %20.15lf\n", t1, t2);
```

```
float:      1.000000053920860
 double:    1.000000000000000
```

8. PROBLEMY Z PRECYZJÄ„ LICZB ZMIENNOPRZECINKOWYCH

```
    float fla = 1.0;
    float flb = 1.0;
    int i = 0;
    while (!(fla + flb == fla))
    {
        flb /= 10;
        i++;
    }
    printf("graniczna wartosc roznicy a i b kiedy a+b=a, float:
%20.15f\nDzielenie przez 10 zostalo wykonane %d razy\n",flb,i);
    i = 0;
    double dbc = 1.0;
    double dbd = 1.0;
    while (!(dbc + dbd == dbc))
    {
        dbd /= 10;
        i++;
    }
    printf("graniczna wartosc roznicy a i b kiedy a+b=a, double:
%20.15lf\nDzielenie przez 10 zostalo wykonane %d razy\n",dbd,i);
```

```
graniczna wartosc roznicy a i b kiedy a+b=a, float:    0.000000009999999
Dzielenie przez 10 zostalo wykonane 8 razy
graniczna wartosc roznicy a i b kiedy a+b=a, double:    0.000000000000000
Dzielenie przez 10 zostalo wykonane 16 razy
```