Anna Jasielec

Podstawy programowania, grupa nr 4

Sprawozdanie z laboratorium nr 4

Cel laboratorium: Opanowanie tworzenia prostych programów w C zapisujących wartości zmiennych różnych typów oraz wykonujących podstawowe operacje na zmiennych.

Przebieg zajęć:

- otworzenie programu zmienne.c,
- kompilacja i sprawdzanie działanie programu,
- odkomentywywanie, analiza przykładów i wykonywanie zadań:

Zadanie 1. Definiowanie zmiennych, inicjowanie i wypisywanie ich wartości

```
50 int liczba_calkowita1;
51 long int liczba_calkowita2;
52 short int liczba_calkowita3;
53 unsigned int liczba_calkowita4;
54 float liczba_rzeczywista1;
55 double liczba_rzeczywista2;
56 char znak1;
57
58 liczba_calkowita1 = 2;
59 liczba_calkowita2 = 455;
60 liczba_calkowita3 = 90;
61 liczba_calkowita4 = 42;
62 liczba_rzeczywista1 = 21.37;
63 liczba_rzeczywista2 = 3.14159;
64 znak1 = 'D';
65
66 liczba_calkowita1 = 12;
68 printf("\nwartość zmiennej  liczba_calkowita1= %d, wartość zmiennej liczba_rzeczywista1=%d, wartosc zmiennej znak1= %c\n",
69          liczba_calkowita1, liczba_rzeczywista1, znak1);
```

```
wartość zmiennej  liczba_calkowita1= 12, wartość zmiennej liczba_rzeczywista1=65,
 wartosc zmiennej znak1= D
```

Zadanie 2. Proste operacje na zmiennych.
- sprawdzenie zachowania kolejności wykonywania działań,

```
87
88 int wynik1;
89 wynik1 = (l_cal1 + 2) * 23;
90 float wynik2 = (l_rze1 - 1) / 23;
91     printf("\n wynik1 = %d, wynik2 = %f\n", wynik1, wynik2);
92
93
```

```
wynik1 = 322, wynik2 = 0.885652
```

- inkrementacja, dekrementacja, preinkrementacja, predekrementacja,

```
107 wynik1 = l_cal2 ++;
108 wynik2 = -- l_rze2;
109 printf("\n wynik1 = %d, wynik2 = %f\n", wynik1, wynik2);
```

```
wynik1 = 455, wynik2 = 2.141590
```

- dzielenie całkowite i modulo(%),

```
127 int x=37;
128 wynik1 = x/l_cal1;
129 int wynik3 = x%l_cal1;
130 printf("\n wynik1 = %d, wynik3 = %d\n", wynik1, wynik3);
131
```

```
wynik1 = 3, wynik3 = 1
```

Zadanie 3. Definiowanie stałych symbolicznych.

```
150 #define STO 100
151 int def = STO;
152  printf("\ndef = %d, stała STO = %d\n", def, STO);
```

```
def = 100, stała STO = 100
```

Zadanie 4. Definiowanie z float i double, sprawdzanie dokładności matematycznej wartości.

```
float f1 = 1.0f/7.0f;  // zapis stałej float z literką f
float f2 = 1.0f/7.0;
double d1 = 1.0/7.0f;
double d2 = 1.0/7.0;

 printf("\nliczby zmiennoprzecinkowe:\n");
 printf("float  -  f1 = %f  (dokładnie: %20.15f)\n", f1, f1);
 printf("float  -  f2 = %f  (dokładnie: %20.15f)\n", f2, f2);
 printf("double -  d1 = %lf  (dokładnie: %20.15lf)\n", d1, d1);
 printf("double -  d2 = %lf  (dokładnie: %20.15lf)\n", d2, d2);
 int s = 1/7;
 printf("(liczba całkowita s = %d - dzielenie całkowite)\n", s);
 int t = 1.0/7.0;
 printf("(liczba całkowita t =%d - obcięcie przy konwersji)\n", t);
```

```
liczby zmiennoprzecinkowe:
float  -  f1 = 0.142857  (dokładnie:    0.142857149243355)
float  -  f2 = 0.142857  (dokładnie:    0.142857149243355)
double -  d1 = 0.142857  (dokładnie:    0.142857142857143)
double -  d2 = 0.142857  (dokładnie:    0.142857142857143)
(liczba całkowita s = 0 - dzielenie całkowite)
(liczba całkowita t =0 - obcięcie przy konwersji)

float  -  f1 = 1.0f/7.0f = 1.428571e-01  (dokładnie: 1.428571492433548e-01)
double -  d1 = 1.0/7.0   = 1.428571e-01  (dokładnie: 1.428571428571428e-01)
```

Zadanie 5. Konstruowanie złożonych wyrażeń logicznych.

```
m = 28;
n = 14;
o = 549;
 p = m>n;
 q = n<o;
 r = o<m;

_Bool b1 = (p||q) && r;
_Bool b2 = (p && r) || (q&&r);
printf("\nWartosc wyrazenia (p lub q) i r wynosi %d\n", b1);
printf("\nWartosc wyrazenia (p i r) lub (q i r) wynosi %d\n", b2);


_Bool b3 = (p&&q) || r;
_Bool b4 = (p||r) && (q||r);
printf("\nWartosc wyrazenia (p i q) lub r wynosi %d\n", b3);
printf("\nWartosc wyrazenia (p lub r) i (q lub r) wynosi %d\n", b4);
```

```
Wartosc wyrazenia (p lub q) i r wynosi 0

Wartosc wyrazenia (p i r) lub (q i r) wynosi 0

Wartosc wyrazenia (p i q) lub r wynosi 1

Wartosc wyrazenia (p lub r) i (q lub r) wynosi 1
```

- (p lub q) i r =? (p i r) lub (q i r) jest równoważne, (p i q) lub r =? (p lub r) i (q lub r) jest równoważne,
- w przypadku usuwania kolejnych nawiasów kolejność jest od lewej do prawej, wynik się zmienia.

Zadanie 6. Niejawne i jawne konwersje typów.

```
int n = 2137;
double d;
double e;
float f = 1.0/6.0;

e = f;
d = 15 + 5.5 * n;
n = f + 10.546;

printf("n = %d, f = %f, d = %lf, e = %le\n",
        n, f, d, e);
```

```
n = 10, f = 0.166667, d = 11768.500000, e = 1.666667e-01
```

```
f =  (double) n/3;
float f1 =  n/3;
float f2 =  n/3.0;
float f3 = (double) (n/3);

printf("f = %lf, f1 = %f, f2 = %f\n",
        f, f1, f2);
```

```
f = 3.333333, f1 = 3.000000, f2 = 3.333333, f3 = 3.000000
```

- kompilator niejawnie skonwertował wartości zmiennych na inne typy.

Zadanie 7. Definiowanie ułamków, sprawdzanie roli nawiasów w definicji.

```
#define JednaTrzecia 1.0/3.0

e = (1.0/3.0) / JednaTrzecia;

printf("e = %lf\n", e);
```

```
e = 0.111111
```

```
#define JednaTrzecia (1.0/3.0)

e = (1.0/3.0) / JednaTrzecia;

printf("e = %lf\n", e);
```

```
e = 1.000000
```

- gdy nie użyłam nawiasu podczas definiowania stałej JednaTrzecia kompilator wykonywał równanie od lewej do prawej (1/3 podzielił na 1, następnie podzielił na 3), stąd wynik e = 0.111111.

- własne przykłady:

```
#define JednaCzwarta 1.0/4.0

e = (22+5) / JednaCzwarta;

printf("e = %lf\n", e);
```

```
e = 6.750000
```

```
#define JednaCzwarta (1.0/4.0)

e = (22+5) / JednaCzwarta;

printf("e = %lf\n", e);
```

```
e = 108.000000
```

Zadanie 8. Analizowanie problemów związanych z precyzją liczb zmiennoprzecinkowych.

- Wynikiem działania (fx1+fx2)-fx1 powinna być zmienna fx2, jednak tak nie jest, ponieważ procesor nie potrafi dokładnie reprezentować takich liczb w systemie binarnym. Dzieje się tak nawet wtedy kiedy nadamy zmiennej typ *double*.

- Liczba przez którą dzielimy jest za blisko 0. Aby naprawić ten błąd należało przyjąć tolerancję 1e-12. Poniżej niej dwie liczby zmiennoprzecinkowe są uznawane za równe.

Wnioski:

- *int* definiuje wartość całkowitą, *float* wartość zmiennoprzecinkową, *char(‘ ’)* pojedynczy znak, *short* lub *long* określa nam zakres wartości jakie zmienna przyjmie, używając *unsigned* zmienna będzie mogła być tylko dodatnia, ale zwiększy się jej dodatni zakres.
- Kompilator kompiluje działania zgodnie z zasadą kolejności wykonywania działań.
- Inicjując zmienną z użyciem inkrementacji najpierw przypiszemy do zmiennej_a wartość zmiennej_b, a następnie podniesiemy wartość zmiennej_b o 1, używając preinkrementacji przypisujemy zmiennej_a podniesioną o 1 wartość zmiennej_b. Dekrementacja i predekrementacja działają podobnie, z tą różnicą, że zmiejszamy wartość o 1.
- Gdy dzielimy przez siebie dwie zmienne typu *int* wynik będzie liczbą całkowitą.
- Modulo (%) pokaże nam resztę z dzielenia dwóch liczb całkowitych.
- Gdy użyjemy *#define symbol stała* kompilator zastąpi wszystkie użycia symbolu w programie na określoną stałą. Przy definiowaniu ułamków należy pamiętać o nawiasach.
- Zmienna typu *double* będzie miała większą dokładność niż zmienna typu *float*.
- *_Bool* to typ zmiennej, który przechowywuje dane prawda(1) i fałsz(0).
- && oznacza i, || oznacza lub, kompilator przy kompilowaniu złożonych wyrażeń logicznych uwzględnia nawiasy.
- Jeżeli przypiszemy do zmiennej_a pewnego typu, wartość zmiennej_b innego typu to kompilator niejawnie konwertuje tą wartość na typ zmiennej_a.
- Chcąc zmienić typ zmiennej możemy użyć konwersji jawnej, czyli: *zmienna = (typ_zmiennej) wartość_zmiennej.*