# Język C – zajęcia nr 10

## *I. Operatory języka C – ciąg dalszy*

### *1. Operator sizeof*

Użycie operatora sizeof może mieć formę:

**sizeof** *zmienna*

gdzie *zmienna* jest nazwą zmiennej dowolnego typu lub formę:

**sizeof (** *typ* **)**

gdzie *typ* jest nazwą typu.

Wartością wyrażeń z operatorem **sizeof** jest liczba bajtów przeznaczonych w pamięci dla podanego jako argument obiektu lub typu.

**Zinterpretuj kod źródłowy programu:**

```
#include <stdio.h>
int main()
{
    char a;
    int b;
    long int c;
    float d;
    double e;
    long double f;
    int g[12];
    printf("\n%d %d %d %d %d %d %d",sizeof a, sizeof b,
        sizeof c, sizeof d, sizeof e, sizeof f, sizeof g);
    printf("\n%d %d",sizeof(int), sizeof(double));
    getch();
}
```

```
1 2 4 4 8 10 24

2 8
```

*Uwaga: wyniki (rozmiary typów) mogą być zależne od użytego kompilatora!*

## *2. Operator rzutowania*

Jawne wymuszenie konwersji typów dokonywane jest poprzez operator rzutowania mający postać:

**( *typ* )**

gdzie ***typ*** jest nazwą typu.

Wyrażenie:

**( *typ* ) *arg***

ma wartość, jaka powstaje z wartości argumentu ***arg*** przekształconej do typu ***typ***.

**Wprowadź i uruchom program, zinterpretuj wyniki:**

```
#include <stdio.h>
#include <math.h>
int main()
{
  float x;
  scanf("%f",&x);
  printf("\n%f",sqrt(x));
  printf("\n%d",(int)sqrt(x));
  getch();
}
```

```
18.4                                KLAWIATURA  18.4↵

4.289522
4
```

## *3. Operatory bitowe*

Do wykonywania operacji na bitach binarnego przedstawienia liczb całkowitych w pamięci komputera wykorzystywane są operatory:

**<<** Przesunięcie w lewo.

Wartością wyrażenia **a<<b** jest liczba całkowita, której reprezentacja dwójkowa ma układ bitów argumentu **a** przesuniętych w lewo o **b** pozycji binarnych. Zwalniane bity z prawej strony uzupełniane są zerami.

**>>** Przesunięcie w prawo.

Wartością wyrażenia **a>>b** jest liczba całkowita, której reprezentacja dwójkowa ma układ bitów argumentu **a** przesuniętych w prawo o **b** pozycji binarnych. Zwalniane bity z lewej strony uzupełniane sa zerami dla typu **`unsigned`** oraz wartością bitu znakowego dla typu **`signed`**.

**|** Bitowa alternatywa.

Wartością wyrażenia **a|b** jest liczba całkowita, która powstaje przez wykonanie niezależnych operacji alternatywy na korespondujących bitach obydwu argumentów.

**&** Bitowa koniunkcja.

Wartością wyrażenia **a&b** jest liczba całkowita, która powstaje przez wykonanie niezależnych operacji koniunkcji na korespondujących bitach obydwu argumentów.

**^** Bitowa różnica symetryczna.

Wartością wyrażenia **a^b** jest liczba całkowita, która powstaje przez wykonanie niezależnych operacji różnicy symetrycznej na korespondujących bitach pierwszego i drugiego argumentu.

**~** Bitowa negacja.

Wartością wyrażenia **~b** jest liczba całkowita, która powstaje przez wykonanie niezależnych operacji negacji na poszczególnych bitach argumentu.

Jeżeli wartości zmiennych **`char a,b;`** mają układ bitów:

```
a  00010101
b  11001100
```

to wyniki operacji bitowych mają układy bitów:

```
a<<2  01010100
a>>3  00000010
a|b   11011101
a&b   00000100
a^b   11011001
~a    11101010
```

**Wprowadź i uruchom program, zinterpretuj wyniki:**

```
#include <stdio.h>
int main()
{
int n=100;
  printf(„%d\n”,n<<2);   // efekt mnozenia przez 4
  printf(„%d\n”,n>>3);   // efekt trzykrotnego dzielenia
                         // calkowitego przez 2
  getch();
}
```

**Zmodyfikuj powyższy program, uzasadnij wyniki:**

```
#include <stdio.h>
int main()
{
int n=100;
  printf(„%d\n”,n<<=2);
  printf(„%d\n”,n>>3);
  getch();
}
```

## 4. Operator połączenia

Para wyrażeń może być rozdzielona operatorem połączenia **,** . Wyrażenie:

*wyr1* **,** *wyr2*

w którym *wyr1* i *wyr2* są dowolnymi wyrażeniami ma wartość prawego wyrażenia *wyr2*. Operator połączenia pozwala umieścić dwa lub więcej wyrażeń w miejscu, gdzie może wystąpić jedno wyrażenie lub jedna instrukcja.

```
if(tab[k]>tab[k+1])
{
    x=tab[k];
    tab[k]=tab[k+1];
    tab[k+1]=x;
}

if(tab[k]>tab[k+1]) x=tab[k], tab[k]=tab[k+1], tab[k+1]=x;
```

**Uwaga:** Przecinki oddzielające wyrażenia na liście argumentów wywołania funkcji lub występujące na liście wartości inicjalizatora **nie są operatorami połączenia**.

**Priorytety i wiązania operatorów (wykaz kompletny):**

| Priorytet | Operator | Wiązanie | Nazwa operatora |
|---|---|---|---|
| 1 | **( )** | Lewostronne | Wywołanie funkcji |
| | **[ ]** | Lewostronne | Indeksowanie |
| | **.** | Lewostronne | Wybór składowej |
| | **->** | Lewostronne | Wybór wskaźnikowy składowej |
| 2 | **+** | Prawostronne | Jednoargumentowy plus |
| | **-** | Prawostronne | Jednoargumentowy minus |
| | **~** | Prawostronne | Bitowa negacja |
| | **!** | Prawostronne | Logiczna negacja |
| | **++** | Prawostronne | Inkrementacja |
| | **--** | Prawostronne | Dekrementacja |
| | ***** | Prawostronne | Wyłuskanie |
| | **&** | Prawostronne | Pobranie adresu |
| | **sizeof** | Prawostronne | Rozmiar |
| | **( )** | Prawostronne | Rzutowanie |
| 3 | ***** | Lewostronne | Mnożenie |
| | / | Lewostronne | Dzielenie |
| | **%** | Lewostronne | Modulo |
| 4 | + | Lewostronne | Dodawanie |
| | - | Lewostronne | Odejmowanie |
| 5 | << | Lewostronne | Bitowe przesunięcie w lewo |
| | >> | Lewostronne | Bitowe przesunięcie w prawo |
| 6 | < | Lewostronne | Relacja mniejszy |
| | <= | Lewostronne | Relacja mniejszy lub równy |
| | > | Lewostronne | Relacja większy |
| | >= | Lewostronne | Relacja większy lub równy |
| 7 | == | Lewostronne | Równy |
| | != | Lewostronne | Różny |
| 8 | & | Lewostronne | Bitowa koniunkcja |
| 9 | ^ | Lewostronne | Bitowa różnica symetryczna |
| 10 | \| | Lewostronne | Bitowa alternatywa |
| 11 | && | Lewostronne | Logiczna koniunkcja |
| 12 | \|\| | Lewostronne | Logiczna alternatywa |
| 13 | **?:** | Prawostronne | Warunek |
| 14 | = | Prawostronne | Przypisania |
| | += | Prawostronne | Złożony przypisania |
| | -= | Prawostronne | Złożony przypisania |
| | *= | Prawostronne | Złożony przypisania |
| | /= | Prawostronne | Złożony przypisania |
| | %= | Prawostronne | Złożony przypisania |
| | &= | Prawostronne | Złożony przypisania |
| | \|= | Prawostronne | Złożony przypisania |
| | <<= | Prawostronne | Złożony przypisania |
| | >>= | Prawostronne | Złożony przypisania |
| | ^= | Prawostronne | Złożony przypisania |
| 15 | **,** | Lewostronne | Połączenie |

**Uwaga:** Wiązanie prawostronne mają jedynie operatory: jednoargumentowe, warunku i przypisania. Pozostałe operatory mają wiązanie lewostronne.

**Uwaga:** Operatory: *wybór składowej i wybór wskaźnikowy składowej* zostaną omówione w dalszej części kursu.

## Konwersje arytmetyczne

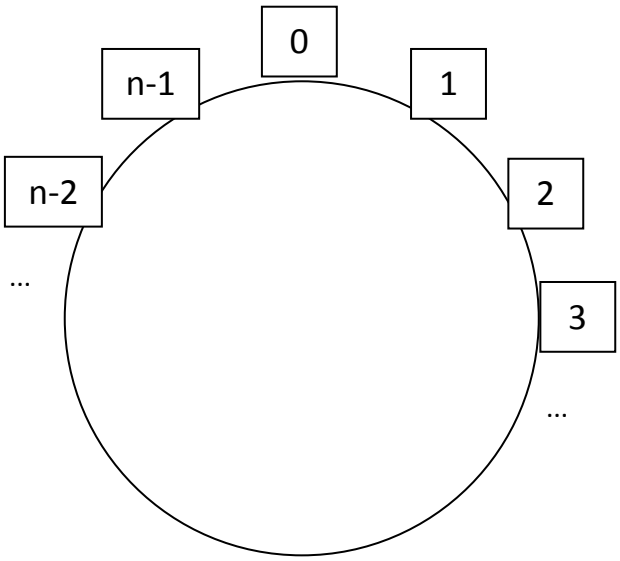
Wyrażenia arytmetyczne mogą być konstruowane z obiektów różnych typów. Do właściwego wyliczenia wartości całego wyrażenia konieczne jest dokonanie przekształceń, czyli **konwersji arytmetycznej** wszystkich elementów wyrażenia do jednego typu.

- **Standardowa** konwersja arytmetyczna wykonywana jest automatycznie (zajmuje się tym kompilator) według zasady, że typ *młodszy* przekształcany jest do typu *starszego*. Typy arytmetyczne w kolejności starszeństwa to: **`char, int, long, float, double, long double`**.
- **Jawna** konwersja zachodzi w przypadku użycia **operatora rzutowania ( *typ* )**.
- **Wymuszona** konwersja związana jest z operatorem przypisania **=** lub **op=** . Typ prawego argumentu tego operatora przekształcany jest do typu lewego argumentu. Podobna konwersja zachodzi podczas przekazywania wartości argumentów przy wywołaniu funkcji. Typ argumentu aktualnego jest przekształcany do typu argumentu formalnego określonego w deklaracji funkcji.

**Zadania:**

1. Zmienna n typu int ma wartość 100, wyznacz wartość n po wykonaniu instrukcji:
   n=n<<3;
   n>>=3;
   n=––n>>1;
   n=n>>1>>1;

2. C10-1. **Eliminacja Rycerzy Okrągłego Stołu.** Program czyta liczbę rycerzy (liczbę miejsce przy stole) *n* (zakładamy, że *n* <= 30) oraz krok eliminacji *k*. Poczynając od pierwszego rycerza następuje odliczanie *k* zajętych miejsc (rycerzy) wokół stołu (puste miejsca nie są liczone), po czym eliminuje się rycerza na którego „wypadła” liczba *k*. Kolejny cykl odliczania rozpoczyna się od następnego rycerza po wyeliminowanym. W ten sposób cyklicznie eliminuje się co *k*-tego rycerza aż do momentu, gdy przy stole pozostanie jeden tylko rycerz. Program wyświetla stan opisanego procesu po każdym cyklu eliminacji.




Przykład sposobu wyświetlenia wyników (*n*=8, *k*=3):

```
1 1 1 1 1 1 1 1
----------------------
1 1 0 1 1 1 1 1
1 1 0 1 1 0 1 1
0 1 0 1 1 0 1 1
0 1 0 1 0 0 1 1
0 0 0 1 0 0 1 1
0 0 0 1 0 0 1 0
0 0 0 0 0 0 1 0
```

Kod źródłowy programu C10-1 wraz z oknem zawierającym efekty przykładowego uruchomienia tego programu, należy zapisać w **jednym** dokumencie (**format Word**) o nazwie **C10.doc** i przesłać za pośrednictwem platformy Moodle w wyznaczonym terminie.

**Przygotuj się do sprawdzianu wiadomości ogólnych i praktycznych na następnych zajęciach.**