

Podstawy Programowania

Wykład nr 3: Operatory relacyjne, logiczne i bitowe. Instrukcje sterujące.

dr hab. inż. Dariusz Dereniowski

Katedra Algorytmów i Modelowania Systemów
Wydział ETI, Politechnika Gdańska

Typy danych i kwalifikatory

Podstawowe typy danych w C:

- **char** – jeden bajt, typ znakowy
- **int** – typ całkowity, standard ANSI określa rozmiar na co najmniej dwa bajty (obecnie w większości kompilatorów cztery bajty)
- **float** – typ zmiennoprzecinkowy pojedynczej precyzji
- **double** – typ zmiennoprzecinkowy podwójnej precyzji

Kwalifikatory

- **short** (krótki, nie dłuższy niż int)
- **long** (długi, przynajmniej 4 bajty)
- **signed** (liczby ze znakiem)
- **unsigned** (liczby bez znaku)

Stałe

- Każda stała jest jakiegoś typu (np. występująca w wyrażeniach).
- Stałe całkowite, np.:
 - 1234 domyślnie typu `int`
 - 1234L typu `long int`
 - 1234U `unsigned int`
 - 1234UL `unsigned long int`
 - 012 system ósemkowy = 10
 - 0x12 system szesnastkowy = 18
- Stałe zmiennoprzecinkowe (domyślnie typu `double`), np.:
 - 314.15
 - 3.1415e2 (lub -3.1415E2) – notacja wykładnicza
 - 31415e-2
 - (użycie literki F lub L zmienia typ na `float` lub `long double`), np. 10.0F, 10.0L
- Stałe znakowe, np. 'a', 'b', '0',
- Stałe łańcuchowe, np. "ala ma psa"
- Znaki specjalne, np.: `\n` (nowa linia), `\r` (powrót karetki), `\t` (tabulacja), `\b` (backspace), `\?`, `\'`, `\"`, `\000` liczba ósemkowa, `\x000` liczba szesnastkowa

Kwalifikatory – przykłady

```
char c; /* zmienna z zakresu od -128 do 127 */
signed char c; /* równoważne powyższemu */
unsigned char c; /* zmienna z zakresu od 0 do 255 */

short int s; /* co najmniej 16 bitów, nie dłuższy niż int */
short s; /* równoważne powyższemu */
signed short s; /* równoważne powyższemu */

unsigned short us; /* jak short, lecz bez znaku */

int i;
signed int i; /* równoważne powyższemu */

long int l; /* co najmniej 32 bity, nie krótszy niż int */
long l; /* równoważne powyższemu */
signed long l;

long long int ll; /* co najmniej 64 bity */
long long ll; /* równoważne powyższemu */
signed long long ll; /* równoważne powyższemu */

unsigned long long ull; /* jak long long, lecz bez znaku */

float f;
double d;
long double ld;
```

Kwalifikator `const`

- Wartość zmiennej poprzedzonej kwalifikatorem `const` nie może być zmieniona w trakcie działania programu.
- Elementem dobrego stylu programowania jest używanie tego kwalifikatora w stosunku do takich zmiennych.
- Przykłady:

```
const double pi=3.14159;  
const int liczba_miesiecy=12;
```

Operator sizeof

Operator `sizeof` pozwala uzyskać rozmiar (w bajtach) danego typu. Pamiętaj, że rozmiary zmiennych zależą od konkretnej architektury. Można użyć poniższego programu, aby dowiedzieć się jakie są rozmiary poszczególnych typów na konkretnym komputerze.

```
#include <iostream>
using namespace std;
int main() { /* Uwaga: endl powoduje wypisanie znaku \n na ekran */
    cout << "Rozmiar typu char: " << sizeof(char) << endl;
    cout << "Rozmiar typu short: " << sizeof(short) << endl;
    cout << "Rozmiar typu int: " << sizeof(int) << endl;
    cout << "Rozmiar typu long: " << sizeof(long) << endl;
    cout << "Rozmiar typu long long: " << sizeof(long long) << endl;
    cout << "Rozmiar typu float: " << sizeof(float) << endl;
    cout << "Rozmiar typu double: " << sizeof(double) << endl;
    cout << "Rozmiar typu long double: " << sizeof(long double) << endl;
    return 0;
}
```

```
Rozmiar typu char: 1
Rozmiar typu short: 2
Rozmiar typu int: 4
Rozmiar typu long: 4
Rozmiar typu long long: 8
Rozmiar typu float: 4
Rozmiar typu double: 8
Rozmiar typu long double: 12
```

Przykładowe zakresy danych

W oparciu o powyższe rozmiary typów (na konkretnym komputerze, na którym uruchomiono powyższy program!) otrzymujemy następujące zakresy typów:

```
Rozmiar typu char: 1
Rozmiar typu short: 2
Rozmiar typu int: 4
Rozmiar typu long: 4
Rozmiar typu long long: 8
Rozmiar typu float: 4
Rozmiar typu double: 8
Rozmiar typu long double: 12
```

Typ	Zakres
char	od -128 do 127
unsigned char	od 0 do 255
short	od -32768 do 32767
unsigned short	od 0 do 65535
int	od -2147483648 do 2147483647
unsigned int	od 0 do 4294967295
long	od -2147483648 do 2147483647
float	$\pm 3.4 \cdot 10^{\pm 38}$
double	$\pm 1.7 \cdot 10^{\pm 308}$
long double	$\pm 1.2 \cdot 10^{\pm 4932}$

Przekroczenie zakresu typu

Uwaga: Należy mieć świadomość tego, że typy mają ograniczony zakres i dobierać typy szacując maksymalną wartość jaką powinny przechować w trakcie działania programu.

Przykład:

```
#include <iostream>
using namespace std;

/* Program ilustrujący przekroczenie zakresu typu */
int main() {
    short s;
    unsigned int ui = 0;

    s = 128*256;
    cout << s << endl;

    ui = ui - 1;
    cout << ui << endl;
    return 0;
}
```

```
-32768
4294967295
```


Operatory logiczne

Operatory logiczne:

- koniunkcja: `a && b`
- alternatywa: `a || b`
- negacja: `!a`

Wartość wyrażenia logicznego jest typu `int`.

Tabela wartości wyrażeń logicznych:

a	b	koniunkcja	alternatywa	negacja a
fałsz	fałsz	fałsz	fałsz	prawda
fałsz	prawda	fałsz	prawda	prawda
prawda	fałsz	fałsz	prawda	fałsz
prawda	prawda	prawda	prawda	fałsz

Operator warunkowy (`?:`). Wyrażenie `a ? b : c` realizowane jest następująco:

- Obliczana jest wartość `a`.
- Jeśli `a \neq 0`, to obliczana jest wartość `b`, natomiast `c` nie jest obliczane.
- Jeśli `a = 0`, to obliczana jest wartość `c`, natomiast `b` nie jest obliczane.

```
#include <iostream>
using namespace std;

/* Program ilustrujący operatory logiczne */
int main() {
    int x = sizeof(int);
    int y = (x>4 ? 1 : 0);

    y == 0 ? (cout << "int ma 4 bajty\n") : (cout << "int ma > 4 bajty");
    return 0;
}
```

Instrukcja wyboru switch

```
switch (wyrażenie)
{
    case wartosc1 : instrukcja1;
        break;
    case wartosc2 : instrukcja2;
        break;
    case wartosc3 : instrukcja3;
        break;
    ...
    case wartoscN : instrukcjaN;
        break;
    default : instrukcjaD;
}
```

Instrukcja switch – wyjaśnienia

- Realizacja instrukcji rozpoczyna się od obliczenia wartości wyrażenia umieszczonego w nawiasach po słowie switch.
- Wartość wyrażenia musi być typu porządkowego (char, int).
- Jeśli jego wartość odpowiada którejś z wartości podanej w jednej z etykiet wartosc1,...,wartoscN, wówczas wykonywane są instrukcje począwszy od tej etykiety.
- Wykonywanie tych instrukcji kończy się po napotkaniu instrukcji break. Działanie instrukcji switch zostaje wówczas zakończone.
- Etykiety warunków muszą być wartościami liczbowymi stałymi (np. 9, 'a').
- Jeśli wartość wyrażenia nie zgadza się z żadną z wartości wartosc1,...,wartoscN, wówczas wykonywane są instrukcje umieszczone po etykiecie default.
- Etykieta default może być umieszczona w dowolnym miejscu instrukcji switch, nawet na samym jej początku.
- Etykietę default można pominąć. Wówczas, jeśli w zbiorze etykiet case nie ma żadnej etykiety równej wartości wyrażenia, instrukcja switch nie będzie wykonana.
- Uwaga! Instrukcje występujące po etykiecie case nie muszą kończyć się instrukcją break. Jeśli jej nie umieścimy, to będą wykonywane instrukcje umieszczone pod następną etykietą case.

Instrukcja switch – przykład

```
#include <iostream>
using namespace std;

/* Program ilustrujący użycie instrukcji switch */
int main() {
    int ocena;

    cin >> ocena;

    switch ( ocena ) {
        case 2:
            cout << "<50%\n" ;
            break;
        case 3:
            cout << "50%–70%\n" ;
            break;
        case 4:
            cout << "70%–90%\n" ;
            break;
        case 5:
            cout << "90%–100%\n" ;
            break;
    }
    return 0;
}
```

Instrukcja while

```
while (wyrażenie)  
    instrukcja;
```

1. Następuje bliczenie wartości wyrażenia w nawiasie.
2. Jeśli wartość ta jest **zerowa**, to następuje zakończenie wykonywania instrukcji while.
3. Jeśli wartość ta jest **niezerowa**, to następuje wykonanie instrukcji, po czym następuje powrót do pkt.1.

Instrukcja while – przykłady

```
#include <iostream>
using namespace std;

/* wypisanie liczb podzielnych przez 3 z zakresu [1,n],
   gdzie n jest wartością podaną przez użytkownika
   — wersja błędna!! */
int main() {
    int n, i=3;

    cin >> n;
    while ( i <= n ) {
        if ( i % 3 == 0 )
            cout << i;
    }
    return 0;
}
```

Uwaga: program działa błędnie: jeśli podamy na wejściu liczbę n większą lub równą 3, to program nigdy się nie zakończy i nie wypisze żadnego znaku na ekran.

Instrukcja while – przykłady

Przykład poprawnie działającego programu:

```
#include <iostream>
using namespace std;

/* wypisanie liczb podzielnych przez 3 z zakresu [1,n],
   gdzie n jest wartością podaną przez użytkownika.*/
int main() {
    int n, i=3;

    cin >> n;
    while ( i <= n ) {
        if ( i % 3 == 0 )
            cout << i << endl;
        i = i+1;
    }
    return 0;
}
```

Instrukcja do-while

```
do  
  instrukcja  
while (wyrażenie);
```

1. Wykonywana jest instrukcja.
2. Następnie obliczana jest wartość wyrażenia w nawiasie.
3. Jeśli jego wartość jest **zerowa**, to następuje zakończenie wykonywania instrukcji do-while. Jeśli jego wartość jest **niezerowa**, to następuje powrót do pkt.1.

Instrukcja do-while – przykład.

```
#include <iostream>
using namespace std;

/* wypisanie liczb podzielnych przez 3 z zakresu [1,n],
   gdzie n jest wartością podaną przez użytkownika.*/
int main() {
    int n, i=3;

    cin >> n;
    if ( n < 3 )
        return 0;
    do {
        if ( i % 3 == 0 )
            cout << i << endl;
        i = i+1;
    } while ( i <= n );
    return 0;
}
```

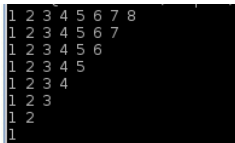
Instrukcja break

- Instrukcję **break** można używać w instrukcjach **for**, **while** oraz **do-while**.
- Jej wykonanie powoduje natychmiastowe zakończenie pętli, w której się znajduje.
- W przypadku pętli zagnieżdżonych, następuje zakończenie petli, w której została wykonania.

```
#include <iostream>
using namespace std;

/* Ilustracja działania instrukcji
   break w zagnieżdżonej pętli.*/
int main() {
    int i, j;

    for ( i=1; i <= 10; i+=1 ) {
        for ( j=1; j <= 10; j+=1 ) {
            if ( i + j >= 10 )
                break;
            cout << j << ' ';
        }
        cout << endl;
    }
    return 0;
}
```



```
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Instrukcja continue

- Instrukcję **continue** można używać w instrukcjach **for**, **while** oraz **do-while**.
- Jej wykonanie powoduje przerwanie bieżącej iteracji pętli, w której się znajduje.
- W przypadku pętli zagnieżdżonych, następuje przerwanie petli, w której została wykonania.
- W przeciwieństwie do instrukcji **break** nie następuje zakończenie wykonywania pętli.

Instrukcja 'zakazana': goto

```
goto etykieta;
```

- Instrukcja wykonuje skok do instrukcji opatrzonej **etykieta**.
- **etykieta** jest literałem, po którym następuje dwukropek.
- Etykieta musi znajdować się w "bieżącym zakresie ważności".
- Należy unikać używania tej instrukcji, gdyż utrudnia ona czytanie i analizę kodu.

Wskaźniki – bardzo krótkie wprowadzenie

- Dla każdego typu podstawowego (np. `int`) można zadeklarować wskaźnik na taki typ, czyli zadeklarować zmienną, która jest dedykowana do zapamiętywania adresu (w pamięci operacyjnej) zmiennej (np. typu `int`).
- Zmienną wskaźnikową typu `typ` deklarujemy: `typ *zmienna;`
- Przykład:
`int i;`
`int *wsk;`
- Dla dowolnej zmiennej, operator `&` zwraca adres tej zmiennej. Kontynuując przykład:
`wsk = &i;`
- Powyższe przypisanie powoduje, że zmienna `wsk` przechowuje adres zmiennej `i`.
- Operator `*` zastosowany do zmiennej wskaźnikowej powoduje odwołanie się do wartości **wskazywanej** przez tą zmienną (a nie do adresu przechowywanego we wskaźniku). Kontynuując przykład:
`*wsk = 1;`
- Powyższe przypisanie powoduje umieszczenie liczby 1 pod adresem przechowywanym w zmiennej `w`. Jako że zmienna `wsk` przechowuje adres zmiennej `i`, przypisanie to zmienia wartość zmiennej `i` (po tym przypisaniu `i` jest równe 1)

Styl programowania – przykład

```
Sum_Dod = 0; Licz_Dod = 0;
Sum_Poz = 0; Licz_Poz = 0;
for (i = 1; i <= N; i = i+1)
    if (a[i] > 0) {
        Sum_Dod = Sum_Dod + a[i];
        Licz_Dod = Licz_Dod + 1;
    } else {
        Sum_Poz = Sum_Poz + a[i];
        Licz_Poz = Licz_Poz + 1;
    }
```

```
Sum_Dod = 0; Licz_Dod = 0;
Sum_Poz = 0; Licz_Poz = 0;
i = 1;
et1: if (a[i] > 0) goto et2;
    Sum_Poz = Sum_Poz + a[i];
    Licz_Poz = Licz_Poz + 1;
    goto et3;
et2: Sum_Dod = Sum_Dod + a[i];
    Licz_Dod = Licz_Dod + 1;
et3: i = i + 1;
    if (i <= N) goto et1;
```

```
Sum_Dod = 0; Licz_Dod = 0; Sum_Poz
    = 0; Licz_Poz = 0; i = 1;
et1: if (a[i] > 0) goto et2;
    Sum_Poz = Sum_Poz + a[i];
    Licz_Poz = Licz_Poz + 1; goto et3;
et2: Sum_Dod = Sum_Dod + a[i];
    Licz_Dod = Licz_Dod + 1;
et3: i = i + 1; if (i <= N) goto
    et1;
```

Zalecenia:

- Stosujemy konsekwentnie wcięcia.
- Unikamy stosowania instrukcji `goto`.
- Nazwy zmiennych powinny sugerować ich przeznaczenie.
- Piszemy jedną instrukcję w jednej linii.

Efekty uboczne

Zasada: wartość zmiennej, która ulega modyfikacji w obrębie wyrażenia, może być pobrana (użyta) w tym wyrażeniu co najwyżej jeden raz! Dotyczy to przede wszystkim operatorów inkrementacji i dekrementacji.

Przykłady błędnych instrukcji:

- `a = (b++) + b;` Wartość zmiennej `b` nie jest jednoznaczna w kontekście operatora dodawania.
- `f() + g();` Nieokreślona kolejność wywołania funkcji `f` i `g`.
- `f(i,i=3);` Niejednoznaczna wartość pierwszego argumentu.