Różności w C++

Marek Pudełko

Kodowanie i reprezentacja liczb

Kodowanie uzupełnieniowe U1 i U2

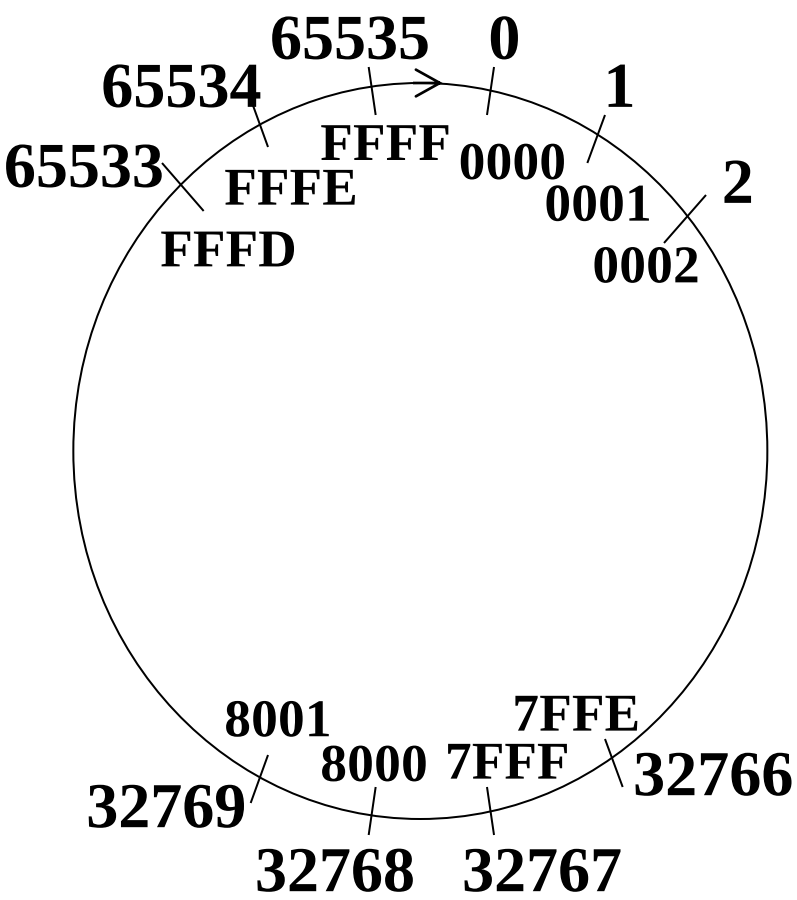
Bity liczby numerujemy od 0 do k –1
Bitom nr j, j < k–1 przyporządkowuje się wagi $w_j = 2^j$
Najstarszej pozycji binarnej przyporządkowuje się wagę ujemną:

$w_{k-1} = -2^{k-1} + 1$ **w kodzie U1,**

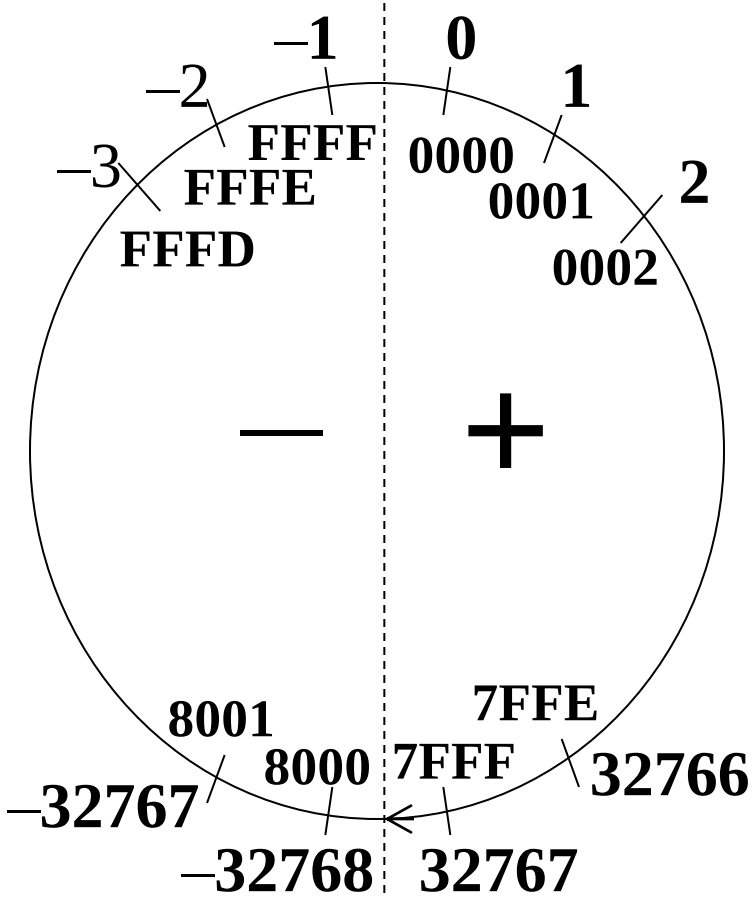
$w_{k-1} = -2^{k-1}$ **w kodzie U2.**

	U1	U2
Zakres	$-2^{k-1}+1$ do $2^{k-1}-1$	-2^{k-1} do $2^{k-1}-1$
Zero	**dwa**: **+0** oraz **–0**	**jedno**
Zmiana znaku	**zanegowanie wszystkich bitów**	**zanegowanie bitów i dodanie jedynki**

Interpretacja graficzna



Kod naturalny



Kod U2

Szesnastobitowy przykład

U1	Kod 16–bitowy	U2
32 767	**0** 111 1111 1111 1111	32 767
32 766	**0** 111 1111 1111 1110	32 766
. . .	. . .	. . .
+2	**0** 000 0000 0000 0010	+2
+1	**0** 000 0000 0000 0001	+1
+0	**0** 000 0000 0000 0000	0
–0	**1** 111 1111 1111 1111	–1
–1	**1** 111 1111 1111 1110	–2
. . .	. . .	. . .
–32 766	**1** 000 0000 0000 0001	–32 767
–32 767	**1** 000 0000 0000 0000	–32 768

Kodowanie +N

Do rejestru wpisuje się liczbę naturalną o N większą.

Zwykle $N=2^{k-1}$ lub $N=2^{k-1}-1$ (k – liczba bitów)

N=32767	16–bitowy kod +N	N=32768
–32 767	**0000 0000 0000 0000**	–32 768
–32 766	**0000 0000 0000 0001**	–32 767
. . .	. . .	. . .
– 2	**0111 1111 1111 1101**	–3
– 1	**0111 1111 1111 1110**	–2
0	**0111 1111 1111 1111**	–1
1	**1000 0000 0000 0000**	0
2	**1000 0000 0000 0001**	+1
. . .	. . .	. . .
32 767	**1111 1111 1111 1110**	32 766
32 768	**1111 1111 1111 1111**	32 767

Kod BCD i BCD+3

Cyfry liczby dziesiętnej koduje się dwójkowo na 4 bitach

	0000	0001	0010	0011	0100	0101	0110	0111
BCD	0	1	2	3	4	5	6	7
BCD+3	-	-	-	0	1	2	3	4

	1000	1001	1010	1011	1100	1101	1110	1111
BCD	8	9	-	-	-	-	-	-
BCD+3	5	6	7	8	9	-	-	-

Liczba dziesiętna: **1 9 2 5**

Kod BCD: **0001 1001 0010 0101**

Kod BCD+3: **0100 1100 0101 1000**

Typy całkowite w C++

Kod	Typ danych	Zakres	Rozmiar
	unsigned **char**	**0 ÷ 255**	**1 bajt**
U2	signed **char**	**–128 ÷ 127**	
	unsigned **short**	**0 ÷ 65 535**	**2 bajty**
U2	**short**	**–32 768 ÷ 32 767**	
	unsigned (1)	**0 ÷ 65 535**	**2 bajty**
	unsigned (2)	**0 ÷ 4 294 967 296**	**4 bajty**
U2	**int** (1)	**–32 768 ÷ 32 767**	**2 bajty**
U2	**int** (2)	**–2 147 483 648 ÷ 2 147 483 647**	**4 bajty**
	unsigned **long**	**0 ÷ 4 294 967 296**	**4 bajty**
U2	**long**	**–2 147 483 648 ÷ 2 147 483 647**	

(1) Borland C++, **(2) MS Visual C++ i Java**

Standard IEEE 754

1 bit	**e** bitów	**m** bitów
s	**E**	**f**
znak	wykładnik w kodzie +N	ułamek (**0≤ f <1**)

F – wartość liczby w rejestrze

$F=(-1)^s(1+f)2^E$ gdy $-N < E < 2^e - N - 1$

$F=(-1)^s(0+f)2^{1-N}$ gdy $E = -N$

Uwaga:
Gdy wykładnik zawiera same zera, to E= –N.
Gdy wykładnik zawiera same jedynki to $E= 2^e - N - 1$.

Obiekty IEEE 754

Wykładnik	Kod binarny s E f	Obiekt
$E = -N$	s 00...00 000...000	± 0
$E = -N$	s 00...00 000...001	$(-1)^s 2^{1-N-m}$
$E = -N$	s 00...00 *bbb...bbb*	$(-1)^s (0 + f) 2^{1-N}$
$E = -N+1$	s 00...01 000...000	$(-1)^s 2^{1-N}$
$-N < E < 2^e - N - 1$	s *ee...ee* *bbb...bbb*	$(-1)^s (1 + f) 2^E$
$E = 2^e - N - 2$	s 11...10 111...111	$(-1)^s (1 - 2^{-m-1}) 2^{2^e - N - 1}$
$E = 2^e - N - 1$	s 11...11 000...000	$\pm\infty$
$E = 2^e - N - 1$	s 11...11 *bbb...bbb*	NaN

ee...ee, bbb...bbb – **nie wszystkie bity równe zeru**

Formaty IEEE 754

Typ IEEE	SINGLE	DOUBLE	**DOUBLE EXT.**
Typ w C i C++	**float**	**double**	**long double**
Liczba bajtów	**4**	**8**	**10** (≥10)
Liczba bitów: **n**	**32**	**64**	**80** (≥79)
Liczba bitów: **m**	**23**	**52**	**64** (≥64)
Wykładnik: **e**	**8**	**11**	**15** (≥15)
Kodowanie: **N**	**127**	**1023**	**16383** (≥16383)
Zakres wykładnika	**[–126,+127]**	**[–1022,+1023]**	**[–(N–1),+N]**
Precyzja dziesiętna	**7 cyfr**	**15 cyfr**	**20 cyfr**
Zakres wartości	$\pm 3{,}4 \cdot 10^{38}$	$\pm 1{,}8 \cdot 10^{308}$	$\pm 1{,}19 \cdot 10^{4932}$

Przykłady

32 bitowa reprezentacja float liczby –1500,6875

$-1500{,}6875_{(10)} = -10111011100{,}1011_{(2)}$

1 10001001 01110111001011000000000

–1 **E=137–127** **f=0,01110111001011 =**

E=10 **N** **= 0,46551513671875**

$F=(-1)(1+0{,}01110111001011)\cdot 2^{10} =$

$= -1{,}01110111001011\cdot 2^{10} = -10111011100{,}1011 =$

$= -1500{,}6875$

Lub na liczbach dziesiętnych

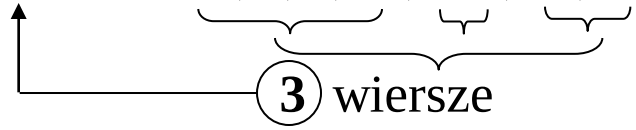
$F =(-1)(1+0{,}46551513671875)2^{10} =$

$= -1{,}46551513671875\cdot 1024 = -1500{,}6875$

Operatory

Inicjowanie tablic c.d.

```
int L[ ][4]={{5,7,3,9},{0},{0,8}},
```




```
M[5][4]={{1},{0,2}};
```

L

5	7	3	9
0	0	0	0
0	8	0	0

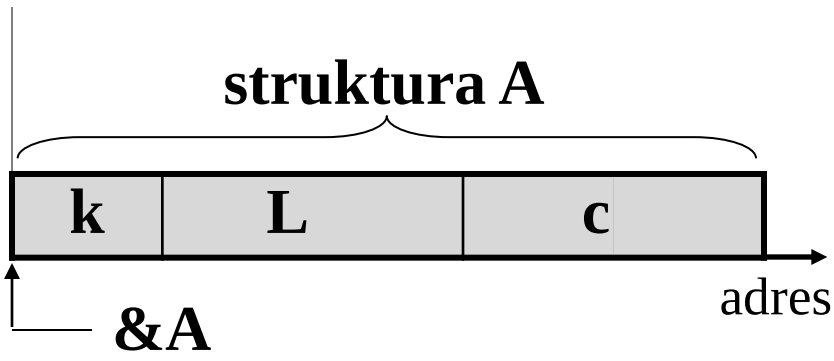
M

1	0	0	0
0	2	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Struktury i unie

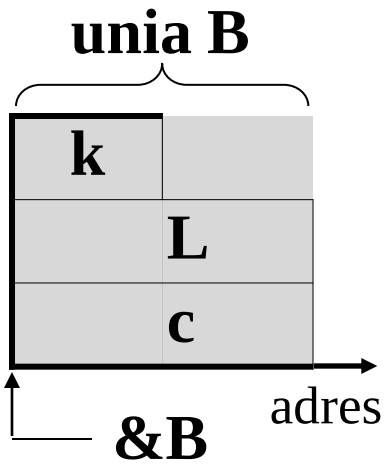
```
struct ST1{
    short k;
    long L;
    unsigned char c[4];
};
```

struct **ST1 A;**




```
union UN1{
    short k;
    long L;
    unsigned char c[4];
};
```

union **UN1 B;**



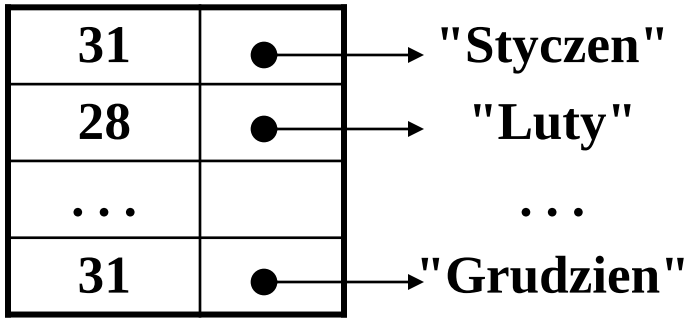
Przykłady tablic struktur

```
struct Rzym {
     char tekst[4];
     short wartosc;
};
```

"M"	1000
"CM"	900
. . .	. . .
"I"	1

```
struct Rzym TR[ ]={{"M",1000}, {"CM",900}, {"D",500},
    {"CD",400}, {"C",100}, {"XC",90}, {"L",50}, {"XL",40},
    {"X",10}, {"IX",9}, {"V",5}, {"IV",4}, {"I",1}};
```

```
struct Miesiace {
     short L_dni;
     char *nazwa;
};
```

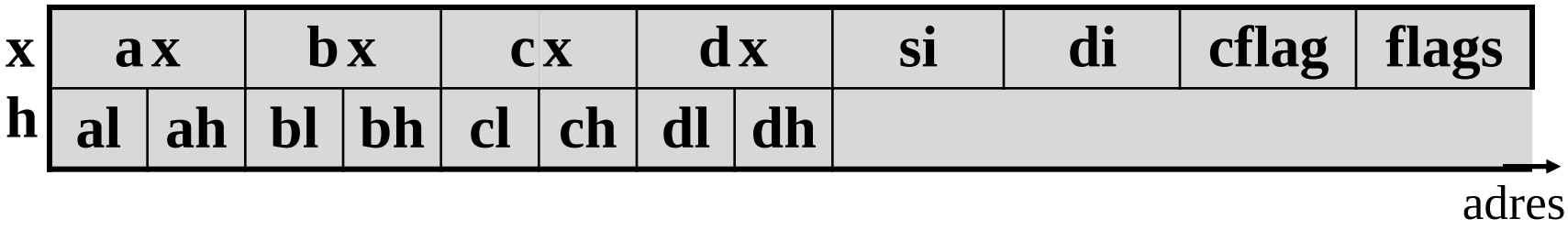



```
struct Miesiace TM[ ]={{31,"Styczen"}, {28,"Luty"},
    {31,"Marzec"}, {30,"Kwiecien"}, {31,"Maj"}, . . .
    . . . {30,"Listopad"}, {31,"Grudzien}};
```

Przykład unii struktur

```
union REGS {
   struct {unsigned short ax, bx, cx, dx, si, di, cflag, flags;} x;
   struct {unsigned char al, ah, bl, bh, cl, ch, dl, dh; } h;
};
```

union **REGS R1;**



R1.x.ax = n; // *niech* **n = 2*256 + 10 = 522**

b0 = R1.h.al; // **b0 = 10** (młodszy bajt n)

b1 = R1.h.ah; // **b1 = 2** (starszy bajt n)

Zmienne wyliczeniowe

enum *Nazwa* **{wyliczenie, wyliczenie, ... wyliczenie};**

Wyliczenie ma postać:

identyfikator
identyfikator = inicjator

Przykłady:

```
enum {False, True};                    // False=0, True=1
enum Boolean {False, True};
enum Boolean L1, L2;                   // zmienne typu Boolean
enum Dni {Pn, Wt, Sr, Cz, Pt, So, Ni};
enum Fig {Kolo, Trojkat=3, Prostokat};
                                       // Kolo = 0, Trojkat = 3, Prostokat = 4
```

Definicje identyfikatorów typów

typedef double Wiersz[20], Kolumna[20], *Wskaznik;

Przykłady:

Wiersz A, B[5];	**double A[20], B[5][20];**
Kolumna C, D[8];	**double C[20], D[8][20];**
Wskaznik E, F[12];	**double *E, *F[12];**

Są dwa główne powody używania specyfikacji **typedef**.

1. **Parametryzacja programu.**
2. **Poprawa czytelności programu.**

Operatory

Poziom	Prio-rytet	Łącz-ność	Operatory języka C i C++	Operatory C++
Kwalifikatory	0	L		:: *Klasa*::
Specjalne	1	L	[] . –> () $_{\nabla}$++ $_{\nabla}$––	*Typ*( )
1-argumentowe	2	P	+ – ++$_{\nabla}$ ––$_{\nabla}$! ~ * & (*Typ*) sizeof	new delete
Komponenty	3	L		.* –>*
Arytme-	4	L	* / %	
tyczne	5	L	+ –	
Przesuwanie	6	L	<< >>	
Porównania	7	L	< <= > >=	
Przyrównania	8	L	== !=	
Bitowe	9	L	&	
	10	L	^	
	11	L	|	
Logiczne	12	L	&&	
	13	L	||	
Wyboru	14	P	$_{\nabla}$?$_{\nabla}$:$_{\nabla}$	
Przypisania	15	P	= *= /= %= += –= <<= >>= &= ^= |=	
Połączenia	16	L	,	

Operatory inkrementacji ++

Są 2 operatory: następnikowy (k++) i poprzednikowy (++k).
Oba zwiększają argument o 1, ale **wyniki dają różne**!!!

```
int i, k=3;
double A[10]={0};      // 10 elementów wyzerowanych
A[k++]=5;              // (początkowe k=3, zatem) A[3]=1, k=4
A[++k]=8;              // (końcowe k=5, zatem) A[5]=8, k=5
k=0;
for(i=0; i<10; i++) A[i]=k++;
// A[0]=0, A[1]=1, A[2]=2, A[3]=3, ..., A[9]=9
k=0;
for(i=0; i<10; i++) A[i]=++k;
// A[0]=1, A[1]=2, A[2]=3, A[3]=4, ..., A[9]=10
```

Rzutowanie typu

```
int k=11, n=4;
double x, y;
x=k/n;          // x=2.0, bo wykonano dzielenie całkowite
x=(double)k/n;      // x=2.75, bo (double)k=11.0

void *p;
p=&x;               // p nie wskazuje na double
y=*(double*)p;      // y=x;
p=&k;               // p nie wskazuje na int
y=*(int*)p;         // y=k;
```

Operator sizeof

```
double x, Y[50], A[40][8], B[]={5,7,2,4,1,0,7,6,3,2,9,8,6,4};
double *p, (*R)[4];
const int n=sizeof(B)/sizeof(B[0]);   // Rozmiar tablicy B
int K[n], M[sizeof(B)/sizeof(B[0])];
          // Tablice K i M mają taki sam rozmiar jak tablica B
```

Prawdziwe sa następujące równości:

```
sizeof(x) == sizeof(double)          // rozmiar zmiennej x
sizeof(Y[0]) == sizeof(double)       // rozmiar zmiennej Y[0]
sizeof(Y) == 50*sizeof(double)       // rozmiar tablicy Y
sizeof(A) == 40*8*sizeof(double)     // rozmiar tablicy A
sizeof(A[0]) == 8*sizeof(double)     // rozmiar wiersza
sizeof(*p) == sizeof(double)         // rozmiar zmiennej *p
sizeof(R) == sizeof(void*)           // rozmiar wskaźnika
sizeof(*R) == 4*sizeof(double)       // rozmiar tablicy
```

Przesunięcia bitowe << >>

int k = 0x335F;	00110011 01011111	k = 13 151
k = k<<3;	10011010 11111000	k = –25 864
k = k>>3; albo	00010011 01011111 11110011 01011111	(?) k = 4 959 (?) k = –3 233
unsigned n=13151	00110011 01011111	n = 13 151
n = n<<3;	10011010 11111000	n = 39 672
n = n>>3	00010011 01011111	n = 4 959
n = (n>>3)<<3	00110011 01011000	n = 13 144
n = 3<<4	00000000 00110000	n = $3*2^4$
n = 13151>>4	00000011 00110101	n = $13151/2^4$

Porównania i przyrównania

Porównania: **< <= > >=** **(< ≤ > ≥)**

Przyrównania: **== !=** **(= ≠)**

$$x < y = \begin{cases} 0 & \text{gdy relacja jest fałszywa,} \\ 1 & \text{gdy relacja jest prawdziwa} \end{cases}$$

Przykłady:

Wyrażenie	Wynik	Uwagi
3.1 < 3	**0**	**Fałsz**
5 == 5	**1**	**Prawda**
5 == 5 == 5	**0**	**(5 == 5) == 5** czyli **1 == 5**
6 > 4 > 2	**0**	**(6 > 4) > 2** czyli **1 > 2**
1 == 5 > 3	**1**	**1 == (5 > 3)** czyli **1 == 1**

Bitowe: and, ex-or, or

A	B	A & B	A ^ B	A | B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

Przykłady:

j =	**01000011**	**11000011**	**= 0x43C3 = 17 347**
k =	**00000010**	**01110110**	**= 0x0276 = 630**
j & k =	**00000010**	**01000010**	**= 0x0242 = 578**
j ^ k =	**01000001**	**10110101**	**= 0x41B5 = 16 821**
j | k =	**01000011**	**11110111**	**= 0x43F7 = 17 399**

k & 1<<n - testuje *n*-ty bit w słowie *k*
k ^ 1<<n - daje wartość *k* z zanegowanym *n*-tym bitem
k & ~(1<<n) - daje wartość *k* z wyzerowanym *n*-tym bitem

Logiczne: and, or

$$x \&\& y = \begin{cases} 0 & \text{gdy } x = 0 \text{ lub } y = 0, \\ 1 & \text{gdy } x \neq 0 \text{ i } y \neq 0. \end{cases}$$

$$x \| y = \begin{cases} 0 & \text{gdy } x = 0 \text{ i } y = 0, \\ 1 & \text{gdy } x \neq 0 \text{ lub } y \neq 0. \end{cases}$$

Najpierw opracowywany jest lewy argument.
Prawy argument opracowuje się tylko wtedy, gdy od niego zależy wynik operacji.

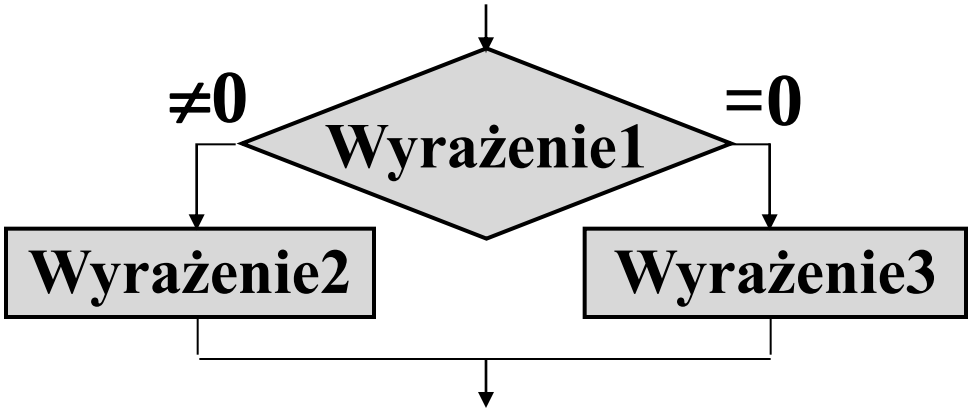
Np. w wyrażeniu

fp && fscanf(fp, "%d", &n)

funkcja *fscanf* jest wywoływana tylko wtedy, gdy fp≠0 (jeśli fp=0, to wynikiem operatora && jest tu 0).

Operator warunkowy

Wyrażenie1 ? Wyrażenie2 : Wyrażenie3;



Przykłady:

MaxXY = x > y ? x : y;

Srednia = n > 1 ? S/n : S;

Max3 = x > y ? (z > x ? z : x) : (z > y ? z : y);

***(x < y ? &x : &y) = 0;** // Zeruje zmienną o mniejszej wartości

Operatory przypisania

A = Wyrażenie; // Wynik = (typ A)Wyrażenie
A @= Wyrażenie; // A=(A)@(Wyrażenie)
// gdzie @= * / % + – << >> & ^ |

Przykłady:

x = k = y = 3.9; // y = 3,9; k = 3; x = 3,0;

k = 3,9
x = 3
3,0

x *= n+1; // x = x*(n+1);
k ^= 1<<n; // negowanie n–tego bitu w słowie k
k |= 1<<n; // ustawienie n–tego bitu w słowie k
k &= ~(1<<n); // zerowanie n–tego bitu w słowie k

Operator połączenia

Wyrażenie1 , Wyrażenie2

Wykonuje się Wyrażenie1 i potem Wyrażenie2.
Wynikiem jest wartość Wyrażenia2.

Przykłady:

```
k = (n=0, n=6, 1);          // k = 1
x = A[i, j];                // x = A[ j]
for(max=A[0], i=1; i<n; i++)
                    if(A[i]>max) max=A[i];
for(i=0, j=n–1; i<j; i++, j––)
                    z=A[i], A[i]=A[j], A[j]=z;
```