Reprezentacja symboli w komputerze.

Liczby całkowite i zmiennoprzecinkowe.

Ostatnia aktualizacja: 29 września 2026

- **Bit** (***bi**nary **d**igit*) najmniejsza ilość informacji $\{0, 1\}$, wysokie/niskie napięcie
- **Kod binarny** - grupa bitów reprezentująca zbiór symboli

1b	$\rightarrow$	2 symbole $\{0, 1\}$
2b	$\rightarrow$	4 symbole $\{00, 01, 10, 11\}$
8b	$\rightarrow$	$256 = 2^8$ symboli
n bitów	$\rightarrow$	2^n symboli

- Do zakodowania n symboli potrzeba co najmniej $\lceil \log_2 n \rceil$ bitów
- Ile bitów potrzeba do zakodowania alfabetu polskiego?

Bajt (*byte*) $1\text{B} = 8\text{b}$

- wg. standardu C: najmniejsza porcja adresowalnej pamięci
- stała `CHAR_BIT` z pliku `limits.h`
- *Najmniejsza porcja danych, którą może „ugryźć" komputer.*
- utożsamiany ze znakiem (zmienna typu `char` ma rozmiar 1B).

Ile pamięci można zaadresować zależnie od szerokości słowa bitowego adresu?

$16\text{b} \rightarrow 2^{16}\text{B} = 64\text{KB}$
$32\text{b} \rightarrow 2^{32}\text{B} = 4294967296\text{B} \sim 4\text{GB}$
$64\text{b} \rightarrow 2^{64}\text{B} \approx 16\text{EB}$ (eksa bajty)

- układ SI: $1\text{k} = 10^3 = 1000 \neq 1024 = 2^{10} = 1K$
- przedrostki dwójkowe zgodnie z normą Międzynarodowej Komisji Elektrotechnicznej (IEC, 1998)

IEC		podstawa						SI	
nazwa	symbol	2	16		różnica	10		nazwa	symbol
kibi	Ki	2^{10}	$16^{2.5}$	400_{16}	2,40%	1 024	$> 10^3$	kilo	k
mebi	Mi	2^{20}	16^{5}	$10\ 0000_{16}$	4,86%	1 048 576	$> 10^6$	mega	M
gibi	Gi	2^{30}	$16^{7.5}$	$4000\ 0000_{16}$	7,37%	1 073 741 824	$> 10^9$	giga	G
tebi	Ti	2^{40}	16^{10}	$100\ 0000\ 0000_{16}$	9,95%	1 099 511 627 776	$> 10^{12}$	tera	T
pebi	Pi	2^{50}	$16^{12.5}$	$4\ 0000\ 0000\ 0000_{16}$	12,59%	1 125 899 906 842 624	$> 10^{15}$	peta	P
eksbi	Ei	2^{60}	16^{15}	$1000\ 0000\ 0000\ 0000_{16}$	15,29%	1 152 921 504 606 846 976	$> 10^{18}$	eksa	E
zebi	Zi	2^{70}	$16^{17.5}$	$40\ 0000\ 0000\ 0000\ 0000_{16}$	18,06%	1 180 591 620 717 411 303 424	$> 10^{21}$	zetta	Z
jobi	Yi	2^{80}	16^{20}	$1\ 0000\ 0000\ 0000\ 0000\ 0000_{16}$	20,89%	1 208 925 819 614 629 174 706 176	$> 10^{24}$	jotta	Y

https://pl.wikipedia.org/wiki/Przedrostek_dwjkowy

Operator `sizeof` zwraca rozmiar typu lub zmiennej w bajtach

Składnia

```
sizeof (typ)
sizeof zmienna
```

Przykład

```
int x = 10;
printf("%d\n", sizeof (int));
printf("%d\n", sizeof x);
```

Typowe użycie

```
int *x;
x = malloc(sizeof(int) * n);
```

```c
#include <stdio.h>
#include <limits.h>

struct s {
    int a;
    char b[100];
};

int main()
{
    int tab[5];

    printf("CHAR_BIT           = %d\n", CHAR_BIT);
    printf("sizeof (char)      = %d\n", sizeof (char));
    printf("sizeof (int)       = %d\n", sizeof (int));
    printf("sizeof (long)      = %d\n", sizeof (long));
    printf("sizeof (float)     = %d\n", sizeof (float));
    printf("sizeof (double)    = %d\n", sizeof (double));
    printf("sizeof (int *)     = %d\n", sizeof (int*));
    printf("sizeof (char *)    = %d\n", sizeof (char*));
    printf("sizeof (struct s) = %d\n", sizeof (struct s));
    printf("sizeof tab         = %d\n", sizeof tab);

    return 0;
}
```

☞ sizeof.c

```c
#include <stdio.h>
#include <limits.h>

struct s {
    int a;
    char b[100];
};

int main()
{
    int tab[5];

    printf("CHAR_BIT          = %d\n", CHA
    printf("sizeof (char)     = %d\n", size
    printf("sizeof (int)      = %d\n", size
    printf("sizeof (long)     = %d\n", sizeof (long));
    printf("sizeof (float)    = %d\n", sizeof (float));
    printf("sizeof (double)   = %d\n", sizeof (double));
    printf("sizeof (int *)    = %d\n", sizeof (int*));
    printf("sizeof (char *)   = %d\n", sizeof (char*));
    printf("sizeof (struct s) = %d\n", sizeof (struct s));
    printf("sizeof tab        = %d\n", sizeof tab);

    return 0;
}
```

Przykładowy wynik (może się różnić zależnie od architektury komputera)

```
CHAR_BIT          = 8
sizeof (char)     = 1
sizeof (int)      = 4
sizeof (long)     = 8
sizeof (float)    = 4
sizeof (double)   = 8
sizeof (int *)    = 8
sizeof (char *)   = 8
sizeof (struct s) = 104
sizeof tab        = 20
```

☞ sizeof.c

$$x_{n-1}x_n \ldots x_0 = \sum_{i=0}^{n-1} x_i a^i$$

a - baza (podstawa) systemu liczbowego

Dziesiętny

$46532_{(10)} = 4 \cdot 10^4 + 6 \cdot 10^3 + 5 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0$

Dwójkowy (binarny)

$10011_{(2)} = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 19_{(10)}$

Ósemkowy (oktalny)

$351_{(8)} = 3 \cdot 8^2 + 5 \cdot 8^1 + 1 \cdot 8^0 = 233_{(10)}$

Szesnastkowy (heksadecymalny)

$3\mathsf{FC}_{(16)} = 3 \cdot 16^2 + 15 \cdot 16^1 + 12 \cdot 16^0 = 1020_{(10)}$

A=10, B=11, C=12, D=13, E=14, F=15

$$1111110101111110_{(2)} = \mathsf{FD7E}_{(16)} = 176576_{(8)} = 64894_{(10)}$$

Binarny na szesnastkowy

1 cyfrze szesnastkowej odpowiadają 4 bity

1111	1101	0111	1110
F	D	7	E

Binarny na ósemkowy

1 cyfrze ósemkowej odpowiadają 3 bity

1	111	110	101	111	110
1	7	6	5	7	6

Liczby ósemkowe i szesnastkowe w C

```c
#include <stdio.h>

int main()
{
    int a = 10;         /* dec */
    int b = 010;        /* oct */
    int c = 0x10;       /* hex */

    printf("dec: %d %d %d\n", a, b, c);
    printf("oct: %o %o %o\n", a, b, c);
    printf("hex: %x %x %x\n", a, b, c);

    return 0;
}
```

☞ dox.c

Przeliczanie na inny system pozycyjny

Problem: wyrazić liczbę dziesiętną $x_{(10)}$ w innym systemie pozycyjnym o podstawie p

binarny $p = 2$

139 % 2	1
69 % 2	1
34 % 2	0
17 % 2	1
8 % 2	0
4 % 2	0
2 % 2	0
1 % 2	1

ósemkowy $p = 8$

139 % 8	3
17 % 8	1
2 % 8	2

szesnastkowy $p = 16$

139 % 16	11 B
8 % 16	8

$$139_{(10)} = 10001011_{(2)} = 213_{(8)} = 8\mathrm{B}_{(16)}$$

Algorithm Zapis liczby dziesiętnej w innej podstawie

Dane wejściowe: liczba całkowita $x \geqslant 0$ oraz podstawa systemu liczbowego p

Wynik: ciąg cyfr $\{t_{n-1}, \ldots, t_1, t_0\}$ reprezentujący liczbę x w systemie o podstawie p

1: $i \leftarrow 0$
2: **dopóki** $x \neq 0$ **wykonuj**
3: $\quad t_i \leftarrow x \bmod p$
4: $\quad x \leftarrow \lfloor \frac{x}{p} \rfloor$
5: $\quad i \leftarrow i + 1$
6: **zwróć** $\{t_{i-1}, \ldots, t_1, t_0\}$ ▷ w odwrotnej kolejności

```c
int zmien_podstawe(int x, int *t, int p)
{
    int i = 0;

    while (x != 0)
    {
        t[i] = x % p;
        x = x / p;
        i = i + 1;
    }
    return i;
}
```

☞ dec2all.c

Operator / dla liczb całkowitych dzieli bez reszty!

Naturalny kod binarny (NKB)

$$b_{n-1} \dots b_1 b_0 = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

- liczby całkowite bez znaku
- zakres $[0, 2^n - 1]$
- operacje arytmetyczne przeprowadza się podobnie jak w systemie dziesiętnym
- Typy całkowite bez znaku w C:

```
unsigned char
unsigned short          unsigned short int
unsigned int
unsigned long           unsigned long int
unsigned long long      unsigned long long int    /* standard C99 */
```

- John Napier, XVI w.

Typy całkowite bez znaku

typ	rozmiar [B]	zakres
unsigned char	1	$[0, 255]$
unsigned short int	2	$[0, 65535]$
unsigned int	4 (lub 2)	$[0, 4294967295]$
unsigned long int	8 (lub 4)	$[0, 18446744073709551615]$
unsigned long long int	8	$[0, 18446744073709551615]$

- Rozmiary typów zależą od implementacji, standard C gwarantuje minimalne rozmiary (np. `int` co najmniej 2B)
- Zakresy typów całkowitych są określone w `limits.h` (np. stała `UINT_MAX` określa maksymalną wartość typu `unsigned int`)

Specyfikator formatu printf/scanf

dec	oct	hex	typ
u	o	x	unsigned int
lu	lo	lx	unsigned long int
llu	llo	llx	unsigned long long int

`unsigned char` i `unsigned short` promowane do `unsigned int`

Przykład

```
unsigned int x = 4294967295;
unsigned long y = 4294967296;
printf("%u  \n", x);
printf("%d  \n", x);
printf("%o  \n", x);
printf("%x  \n", x);
printf("%u  \n", y);
printf("%lu \n", y);
```

```
4294967295
-1
37777777777
ffffffff
0
4294967296
```

☞ uint2.c

```c
#include <stdio.h>
#include <limits.h>

int main()
{
    printf("sizeof\n");
    printf("unsigned char          = %lu\n", sizeof (unsigned char));
    printf("unsigned short         = %lu\n", sizeof (unsigned short));
    printf("unsigned int           = %lu\n", sizeof (unsigned int));
    printf("unsigned long int      = %lu\n", sizeof (unsigned long int));
    printf("unsigned long long int = %lu\n", sizeof (unsigned long long int));
    printf("Zakres:\n");
    printf("UCHAR_MAX              = %u\n"  , UCHAR_MAX);
    printf("USHRT_MAX              = %u\n"  , USHRT_MAX);
    printf("UINT_MAX               = %u\n"  , UINT_MAX);
    printf("ULONG_MAX              = %lu\n" , ULONG_MAX);
    printf("ULLONG_MAX             = %llu\n", ULLONG_MAX);

    return 0;
}
```

☞ uint1.c

```c
#include <stdio.h>
#include <limits.h>

int main()
{
    printf("sizeof\n");
    printf("unsigned char
    printf("unsigned short
    printf("unsigned int
    printf("unsigned long in
    printf("unsigned long lo
    printf("Zakres:\n");
    printf("UCHAR_MAX
    printf("USHRT_MAX
    printf("UINT_MAX
    printf("ULONG_MAX
    printf("ULLONG_MAX

    return 0;
}
```

```
                = %u\n   , UCHAR_MAX);
                = %u\n"  , USHRT_MAX);
                = %u\n"  , UINT_MAX);
                = %lu\n" , ULONG_MAX);
                = %llu\n", ULLONG_MAX);
```

Wartości mogą się róznić zależnie od implementacji.

```
sizeof
unsigned char           = 1
unsigned short          = 2
unsigned int            = 4
unsigned long int       = 8
unsigned long long int  = 8
Zakres:
UCHAR_MAX               = 255
USHRT_MAX               = 65535
UINT_MAX                = 4294967295
ULONG_MAX               = 18446744073709551615
ULLONG_MAX              = 18446744073709551615
```

☞ uint1.c

U2, kod uzupełnień do dwóch

$$b_{n-1} \dots b_1 b_0 = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

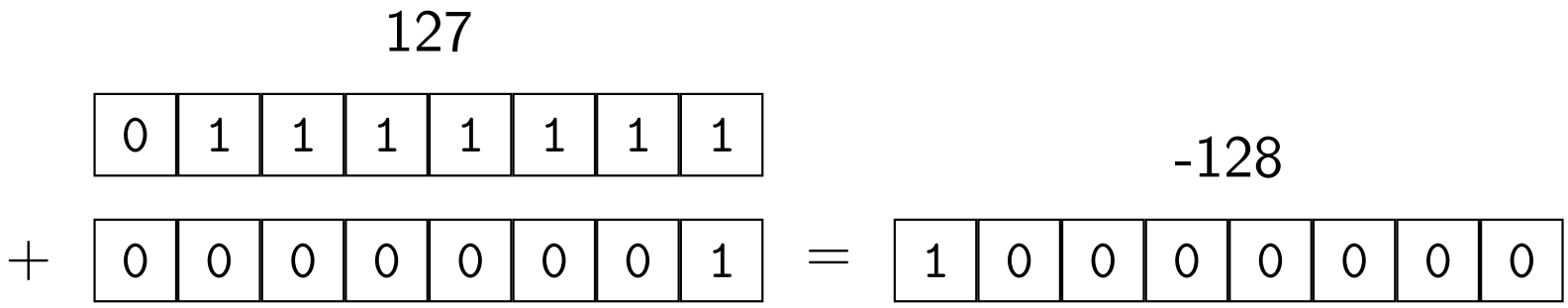
- liczby ze znakiem, najstarszy bit b_{n-1} odpowiada za znak
- zakres $\left[-2^{n-1}, 2^{n-1} - 1\right]$
- typy całkowite ze znakiem w C:

```
char
short               short int
int
long                long int
long long           long long int    /* standard C99 */
```

- można także użyć słowa `signed`,
  np.: `signed long int`

Porównanie reprezentacji NKB i U2

bity	NKB	U2
00000000	0	0
00000001	1	1
⋮		
01111110	126	126
01111111	127	127
10000000	128	-128
10000001	129	-127
⋮		
11111110	254	-2
11111111	255	-1



- **Nadmiar** (*overflow*) - przekroczenie zakresu wynikające ze skończonej liczby bitów reprezentujących liczbę.
- Zazwyczaj błąd nie jest sygnalizowany.
- Spowodowany przez dodawanie dużych liczb dodatnich (lub odejmowanie liczb ujemnych). W U2 widoczna zmiana znaku.
- Zapis liczby typu bardziej pojemnego do zmiennej typu mniej pojemnego (np. `float` do `char`) może spowodować nadmiar.
- Dodawanie liczby ujemnej i dodatniej nie powoduje nadmiaru.
- 4 czerwiec 1996, pierwszy lot rakiety Ariane 5 zakończony zniszczeniem w skutek nadmiaru, straty 370 mln $.

Typy całkowite ze znakiem

typ	rozmiar [B]	zakres
char	1	$[-128, 127]$
short int	2	$[-32768, 32767]$
int	4 (lub 2)	$[-2147483648, 2147483647]$
long int	8 (lub 4)	$[-9223372036854775808,$
long long int	8	$9223372036854775807]$

- Zakresy typów całkowitych są określone w `limits.h`
- Specyfikacja formatu `printf` i `scanf`:

specyfikator	typ
`d, i`	`int, char, short`
`ld, li`	`long int`
`lld, lli`	`long long int`

```
long int x = 4294967295;
printf("%d\n",  x);       /*  ZLE */
printf("%ld\n", x);       /*  OK  */
```

```c
#include <stdio.h>
#include <limits.h>

int main()
{
    printf("sizeof:\n");
    printf("char           = %lu\n", sizeof (char));
    printf("short          = %lu\n", sizeof (short));
    printf("int            = %lu\n", sizeof (int));
    printf("long int       = %lu\n", sizeof (long int));
    printf("long long int = %lu\n", sizeof (long long int));
    printf("Zakresy:\n");
    printf("CHAR_MIN       = %d\n"  , CHAR_MIN);
    printf("CHAR_MAX       = %d\n"  , CHAR_MAX);
    printf("SHRT_MIN       = %d\n"  , SHRT_MIN);
    printf("SHRT_MAX       = %d\n"  , SHRT_MAX);
    printf("INT_MIN        = %d\n"  , INT_MIN);
    printf("INT_MAX        = %d\n"  , INT_MAX);
    printf("LONG_MIN       = %ld\n" , LONG_MIN);
    printf("LONG_MAX       = %ld\n" , LONG_MAX);
    printf("LLONG_MIN      = %lld\n", LLONG_MIN);
    printf("LLONG_MAX      = %lld\n", LLONG_MAX);

    return 0;
}
```

☞ int1.c

```c
#include <stdio.h>
#include <limits.h>

int main()
{
    printf("sizeof:\n");
    printf("char          = %lu\n", sizeof(char));
    printf("short         = %lu\n", sizeof(short));
    printf("int           = %lu\n", sizeof(int));
    printf("long int      = %lu\n", sizeof(long int));
    printf("long long int = %lu\n", sizeof(long long int));
    printf("Zakresy:\n");
    printf("CHAR_MIN      = %d\n", CHAR_MIN);
    printf("CHAR_MAX      = %d\n", CHAR_MAX);
    printf("SHRT_MIN      = %d\n", SHRT_MIN);
    printf("SHRT_MAX      = %d\n", SHRT_MAX);
    printf("INT_MIN       = %d\n", INT_MIN);
    printf("INT_MAX       = %d\n", INT_MAX);
    printf("LONG_MIN      = %ld\n", LONG_MIN);
    printf("LONG_MAX      = %ld\n", LONG_MAX);
    printf("LLONG_MIN     = %lld\n", LLONG_MIN);
    printf("LLONG_MAX     = %lld\n", LLONG_MAX);

    return 0;
}
```

Wartości mogą się różnić zależnie od implementacji.

```
sizeof:
char          = 1
short         = 2
int           = 4
long int      = 8
long long int = 8
Zakresy:
CHAR_MIN      = -128
CHAR_MAX      = 127
SHRT_MIN      = -32768
SHRT_MAX      = 32767
INT_MIN       = -2147483648
INT_MAX       = 2147483647
LONG_MIN      = -9223372036854775808
LONG_MAX      = 9223372036854775807
LLONG_MIN     = -9223372036854775808
LLONG_MAX     = 9223372036854775807
```

☞ int1.c

Kod stałoprzecinkowy

$$3,14_{(10)} = 3 \cdot 10^0 + 1 \cdot 10^{-1} + 4 \cdot 10^{-2}$$
$$11,001_{(2)} = 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 3,125_{(10)}$$

Zapis zmiennoprzecinkowy

$$L = m \cdot p^c$$

c - cecha
p - podstawa (baza)
m - mantysa

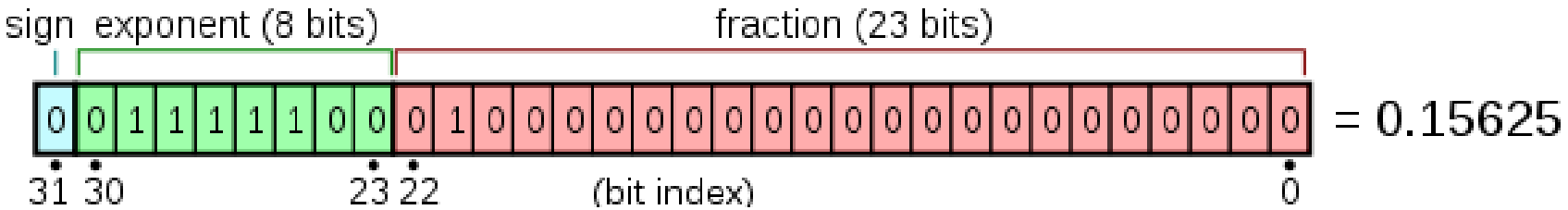
$$3,14 \cdot 10^0 = 0,0314 \cdot 10^2 = 314 \cdot 10^{-2}$$

Normalizacja: w celu uniknięcia wieloznaczności zapisu wymaga się, aby mantysa spełniała warunek:

$$1 \leqslant |m| < p$$

pojedyncza precyzja (*single precision*), `float`

$$x = (-1)^s \cdot 1.m \cdot 2^{(c-127)}$$



$s = 0$

$1.m = \left(1 + \sum_{i=1}^{23} b_{23-i} \cdot 2^{-i}\right) = 1 + 2^{-2} = 1.25$

$2^{(c-127)} = 2^{(124-127)} = 2^{-3} = 0.125$

$x = (-1)^s \cdot 1.m \cdot 2^{(c-127)} = 1 \cdot 1.25 \cdot 0.125 = 0.15625$

Podobne rozwiązanie stosował Konrad Zuse, 1936 r.

http://en.wikipedia.org/wiki/Single-precision_floating-point_format

typ	rozmiar	zakres	cyfry znaczące
`float`	32 b	$\pm 3.4 \cdot 10^{38}$	6
`double`	64 b	$\pm 1.8 \cdot 10^{308}$	15
`long double`	80 b	$\pm 1.2 \cdot 10^{4932}$	18

- Zakresy typów zmiennopozycyjnych są zdefiniowane w pliku nagłówkowym `float.h`
  np.: `FLT_MAX`, `DBL_MAX`
- Zakres `long double` zalezy od implementacji
  (w kompilatorze MS VC++ jest tożsamy z `double`)

Formatowenie liczb zmiennopozycyjnych

Formatowanie liczb zmiennopozycyjnych za pomocą `printf`

specyfikator	znaczenie	przykład
`f`	dziesiętne	`123.45678`
`e`, `E`	notacja naukowa	`1.2345678e+2`
`g`	krótki zapis (dziesiętny lub naukowy)	`123.4657`
`Lf`, `Le`, `Lg`	dla typu `long double`	1.2345e+400
`.2f`	dokładność 2 miejsc po przecinku	`123.46`

typ `float` jest promowany do `double`

Przykład

```
float  x = 10.14;
double y = 5e-3;

printf("%f  %e  %g\n", x, x, x);
printf("%f  %e  %g\n", y, y, y);
```

```
10.140000  1.014000e+01  10.14
0.005000  5.000000e-03  0.005
```

Wartości specjalne: NaN, Inf

Typ zmiennopozycyjny pozwala reprezentować specjalne wartości:

- `NaN` to wartość nieokreślona (*not a number*), np.: $\frac{0}{0}$, $\sqrt{-1}$
- `Inf` to nieskończoność, np. wynik dzielenia przez 0
- Wartości `NaN` i `Inf` są propagowane w dalszych obliczeniach i nie powodują błędu wykonania programu

Operacja	Wynik
$\pm x / \pm\infty$	0
$\pm\infty \cdot \pm\infty$	$\pm\infty$
$\pm x / 0$	$\pm\infty$
$\infty + \infty$	∞
$\pm 0 / \pm 0$	`NaN`
$\infty - \infty$	`NaN`
$\pm\infty / \pm\infty$	`NaN`
$\pm\infty \cdot 0$	`NaN`

Niedokładność reprezentacji

```c
#include <stdio.h>

int main()
{
    float x = 0.1;

    if (x == 0.1) printf ("OK, %f jest rowne 0.1\n", x);
    else printf("Nie OK, %f nie jest rowne 0.1\n", x);

    return 0;
}
```

☞ prec.c

```c
#include <stdio.h>

int main()
{
    float x = 0.1;

    if (x == 0.1) printf ("OK, %f jest rowne 0.1\n", x);
    else printf("Nie OK, %f nie jest rowne 0.1\n", x);

    return 0;
}
```

☞ prec.c

- Niektóre liczby nie będą dokładnie reprezentowane
- Liczb zmiennoprzecinkowych nie należy przyrównywać do dokładnych wartości. Lepiej użyć warunku $|a - b| < \epsilon$

```c
#include <stdio.h>
#include <math.h>
#define EPS 0.000001

int main()
{
    float x = 0.1;

    if (fabs(x-0.1) < EPS)
        printf("OK, %f jest rowne 0.1\n", x);
    else
        printf("Nie OK, %f nie jest rowne 0.1\n", x);

    return 0;
}
```

☞ prec2.c

```c
#include <stdio.h>
#include <float.h>

int main()
{
   printf("Limity typu float:\n");
   printf("sizeof(float)         : %lu\n", sizeof(float));
   printf("Najwieksza wartosc    : %e\n", FLT_MAX );
   printf("Najmniejsza dodatnia  : %e\n", FLT_MIN );
   printf("Epsilon maszynowy     : %e\n", FLT_EPSILON );
   printf("Cyfry znaczace        : %d\n", FLT_DIG );

   printf("\nLimity typu double:\n");
   printf("sizeof(double)        : %lu\n", sizeof(double));
   printf("Najwieksza wartosc    : %e\n", DBL_MAX );
   printf("Najmniejsza dodatnia  : %e\n", DBL_MIN );
   printf("Epsilon maszynowy     : %e\n", DBL_EPSILON );
   printf("Cyfry znaczace        : %d\n", DBL_DIG );

   printf("\nLimity typu long double:\n");
   printf("sizeof(long double)   : %lu\n", sizeof(long double));
   printf("Najwieksza wartosc    : %Le\n", LDBL_MAX );
   printf("Najmniejsza dodatnia  : %Le\n", LDBL_MIN );
   printf("Epsilon maszynowy     : %Le\n", LDBL_EPSILON );
   printf("Cyfry znaczace        : %d\n" , LDBL_DIG );

   return 0;
}
```

☞ float.c

```c
#include <stdio.h>
#include <float.h>

int main()
{
    printf("Limity typu float:\n");
    printf("sizeof(float)         : %lu\n", sizeo
    printf("Najwieksza wartosc    : %e\n", FLT_MA
    printf("Najmniejsza dodatnia  : %e\n", FLT_MI
    printf("Epsilon maszynowy     : %e\n", FLT_EP
    printf("Cyfry znaczace        : %d\n", FLT_DI

    printf("\nLimity typu double:\n");
    printf("sizeof(double)        : %lu\n", sizeo
    printf("Najwieksza wartosc    : %e\n", DBL_MA
    printf("Najmniejsza dodatnia  : %e\n", DBL_MI
    printf("Epsilon maszynowy     : %e\n", DBL_EP
    printf("Cyfry znaczace        : %d\n", DBL_DI

    printf("\nLimity typu long double:\n");
    printf("sizeof(long double)   : %lu\n", sizeo
    printf("Najwieksza wartosc    : %Le\n", LDBL_
    printf("Najmniejsza dodatnia  : %Le\n", LDBL_
    printf("Epsilon maszynowy     : %Le\n", LDBL_EPSILON );
    printf("Cyfry znaczace        : %d\n" , LDBL_DIG );

    return 0;
}
```

Wartości mogą się różnić zależnie od implementac

```
Limity typu float:
sizeof(float)         : 4
Najwieksza wartosc    : 3.402823e+38
Najmniejsza dodatnia  : 1.175494e-38
Epsilon maszynowy     : 1.192093e-07
Cyfry znaczace        : 6

Limity typu double:
sizeof(double)        : 8
Najwieksza wartosc    : 1.797693e+308
Najmniejsza dodatnia  : 2.225074e-308
Epsilon maszynowy     : 2.220446e-16
Cyfry znaczace        : 15

Limity typu long double:
sizeof(long double)   : 16
Najwieksza wartosc    : 1.189731e+4932
Najmniejsza dodatnia  : 3.362103e-4932
Epsilon maszynowy     : 1.084202e-19
Cyfry znaczace        : 18
```

☞ float.c

- **nadmiar** (*overflow*) - przekroczenie zakresu (`+Inf`, `-Inf`)
- **niedomiar** (*underflow*) - zaokrąglenie bardzo małej liczby do 0
- redukcja cyfr przy odejmowaniu bardzo bliskich sobie liczb
- dodawanie (lub odejmowanie) bardzo dużej i bardzo małej liczby powoduje utratę precyzji
- kolejność operacji może mieć wpływ na wynik
  $(a + b) + c \neq a + (b + c)$
- zaokrąglenia wynikające z reprezentacji binarnej, np. liczby 0.1 nie można dokładnie reprezentować w systemie binarnym
- 25 luty 1991 r., wojna w zatoce perskiej, awaria systemu antyrakietowego Patriot wywołana utratą precyzji (zegar rakiety tykał co 0.1 s.), zginęło 28 amerykańskich żołnierzy a 100 zostało rannych.

Problem: wyznaczyć sumę pierwszych n elementów szeregu harmonicznego

$$\sum_{i=1}^{n} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots \xrightarrow[n\to\infty]{} \infty$$

```c
#include <stdio.h>

int main()
{
    float x=0.0, y=0.0;
    int i=1, n;

    printf("n="); scanf("%d", &n);

    for (i=1; i <= n; i++)
        x = x + 1.0 / i;

    for (i=n; i >= 1; i--)
        y = y + 1.0 / i;

    printf("x=%f\ny=%f\n", x, y);
    return 0;
}
```

☞ szereg.c

```
n=66
x=4.774428
y=4.774427

n=600
x=6.974980
y=6.974978

n=2000
x=8.178369
y=8.178370

n=500000
x=13.690692
y=13.699607

n=1000000
x=14.357358
y=14.392652
```

- Typy całkowite: `char`, `short`, `int`, `long`
- Typy całkowite bez znaku: `unsigned`
- Typy zmiennopozycyjne: `float`, `double`
- Nadmiar, niedomiar i inne efekty wynikające z bitowej reprezentacji liczb
- `3/2` wynosi `1`, zaś `3/2.0` wynosi `1.5`
- Porównywanie wartości zmiennopozycyjnych `fabs(a-b)<EPS`

- Jerzy Wałaszek, *„Binarne kodowanie liczb”*, http://edu.i-lo.tarnow.pl/inf/alg/006_bin/index.php
- Thomas Huckle, *„Collection of Software Bugs”*, http://www5.in.tum.de/~huckle/bugse.html
- Piotr Krzyżanowski, Leszek Plaskota, Materiały do wykładu *„Metody numeryczne”*, Uniwersytet Warszawski, WMIiM, http://wazniak.mimuw.edu.pl/
- WikiBooks, *„Kursie programowania w języku C”*, Zaawansowane operacje matematyczne, http://pl.wikibooks.org/wiki/C/Zaawansowane_operacje_matematyczne/