Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

# Informatyka 1 (EMS1A1004)

Politechnika Białostocka - Wydział Elektryczny

Elektromobilność, semestr I, studia stacjonarne I stopnia

## Wykład nr 3

dr inż. Jarosław Forenc

# Plan wykładu nr 3

- Funkcje printf() i scanf()
- Specyfikatory formatu
- Reprezentacja liczb całkowitych w języku C
- Reprezentacja liczb zmiennoprzecinkowych w języku C

# Język C - Funkcja printf()

- Ogólna składnia funkcji printf()

```
printf("łańcuch_sterujący",arg1,arg2,...);
```

- W najprostszej postaci printf() wyświetla tylko tekst

```
printf("Witaj swiecie");
```

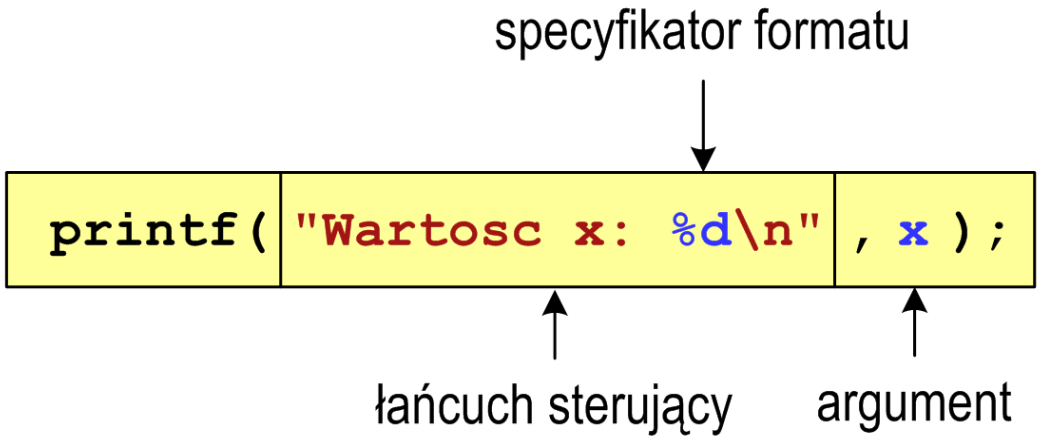
```
Witaj swiecie
```

- Do wyświetlenia wartości zmiennych konieczne jest zastosowanie specyfikatorów formatu, określających typ oraz sposób wyświetlania argumentów

```
%[znacznik][szerokość][.precyzja][modyfikator]typ
```

# Język C - Funkcja printf()

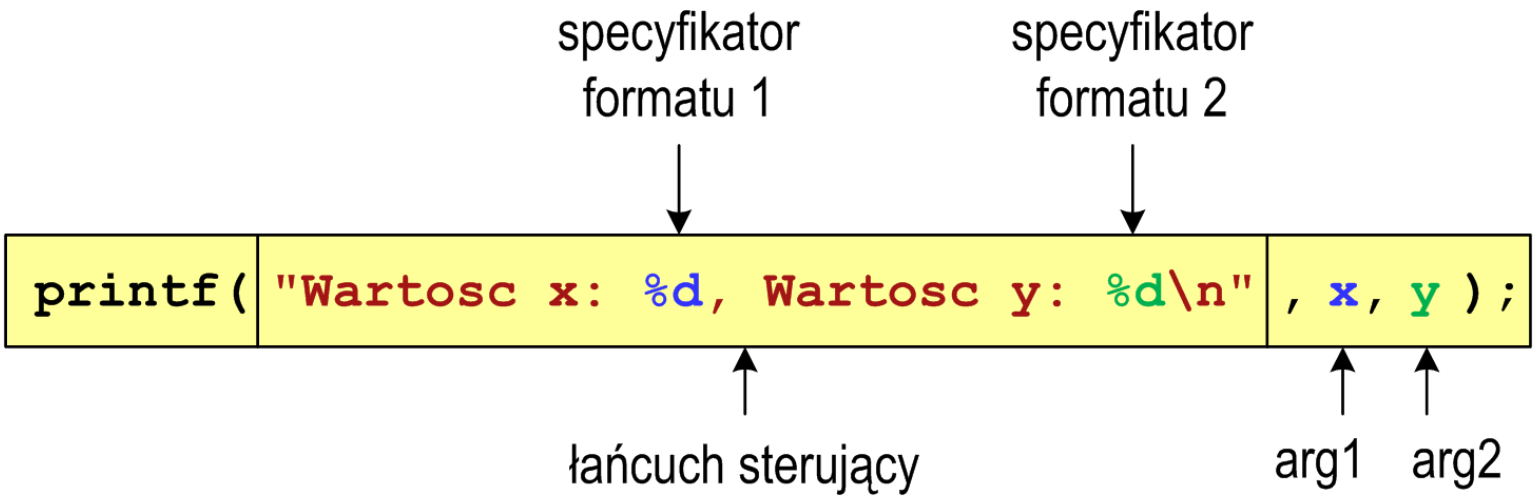
```
int x = 10;
printf("Wartosc x: %d\n", x);
```




```
Wartosc x: 10
```

# Język C - Funkcja printf()

```
int x = 10, y = 20;
printf("Wartosc x: %d, Wartosc y: %d\n", x, y);
```




```
Wartosc x: 10, Wartosc y: 20
```

# Język C - Specyfikatory formatu (printf)

| Typ w C | Specyfikator | Uwagi |
|---|---|---|
| `char` | `%c` | pojedynczy znak |
| | `%d` | kod ASCII znaku, liczba całkowita |
| `char *` | `%s` | łańcuch znaków, napis |
| `int` | `%d  %i` | liczba całkowita, dziesiętna |
| | `%o %O` | liczba całkowita, ósemkowa |
| | `%x  %X` | liczba całkowita, szesnastkowa |
| `float` `double` | `%f` | liczba rzeczywista |
| | `%e  %E` | liczba rzeczywista, format naukowy |
| | `%g  %G` | liczba rzeczywista (%f lub %e) |

# Język C - Funkcja printf()

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%d], y = [%f]\n", x, y);
```

```
x = [123], y = [1.234568]
```

```
printf("x = [], y = []\n", x, y);
```

```
x = [], y = []
```

```
printf("x = [%d], y = [%d]\n", x, y);
```

```
x = [123], y = [-536870912]
```

# Język C - Funkcja printf()

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%6d], y = [%12f]\n", x, y);
```

```
x = [   123], y = [    1.234568]
```

```
printf("x = [%6d], y = [%12.3f]\n", x, y);
```

```
x = [   123], y = [       1.235]
```

```
printf("x = [%6d], y = [%.3f]\n", x, y);
```

```
x = [   123], y = [1.235]
```

# Język C - Funkcja printf()

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%+6d], y = [%+12f]\n", x, y);
```

```
x = [  +123], y = [   +1.234568]
```

```
printf("x = [%-6d], y = [%-12f]\n", x, y);
```

```
x = [123   ], y = [1.234568    ]
```

```
printf("x = [%06d], y = [%012f]\n", x, y);
```

```
x = [000123], y = [00001.234568]
```

# Język C - Funkcja printf()

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%d], y = [%f]\n", x+321, y*25.5f);
```

```
x = [444], y = [31.481482]
```

```
printf("x = [%d], y = [%f]\n", 123, 2.0f*sqrt(y));
```

```
x = [123], y = [2.222222]
```

# Język C - Funkcja scanf()

- Ogólna składnia funkcji scanf()

```
scanf("specyfikatory",adresy_argumentów);
```

- Składnia specyfikatora formatu

```
%[szerokość][modyfikator]typ
```

- Argumenty są adresami obszarów pamięci, dlatego muszą być poprzedzone znakiem &

```
int x;
scanf("%d", &x);
```

# Język C - Funkcja scanf()

- Specyfikatory formatu w większości przypadków są takie same jak w przypadku funkcji printf()
- Największa różnica dotyczy typów float i double

| Typ w C | Specyfikator | Uwagi |
|---|---|---|
| `float` | `%f` | liczba rzeczywista |
| | `%e` `%E` | liczba rzeczywista, format naukowy |
| | `%g` `%G` | liczba rzeczywista (%f lub %e) |
| `double` | `%lf` | liczba rzeczywista |
| | `%le` `%lE` | liczba rzeczywista, format naukowy |
| | `%lg` `%lG` | liczba rzeczywista (%f lub %e) |

# Język C - Funkcja scanf()

```
int a, b, c;
scanf("%d %d %d", &a, &b, &c);
```

- Wczytywane argumenty mogą być oddzielone od siebie dowolną liczbą białych (niedrukowalnych) znaków: spacja, tabulacja, enter

```
15 20 -30
```

```
15□20□-30<enter>
```

```
   15    20     -30
```

```
□□15□□□20□□□□-30<enter>
```

```
15
20
-30
```

```
15<enter>
20<enter>
-30<enter>
```

# Liczby całkowite bez znaku w języku C

- Typy zmiennych całkowitych bez znaku stosowane w języku C:

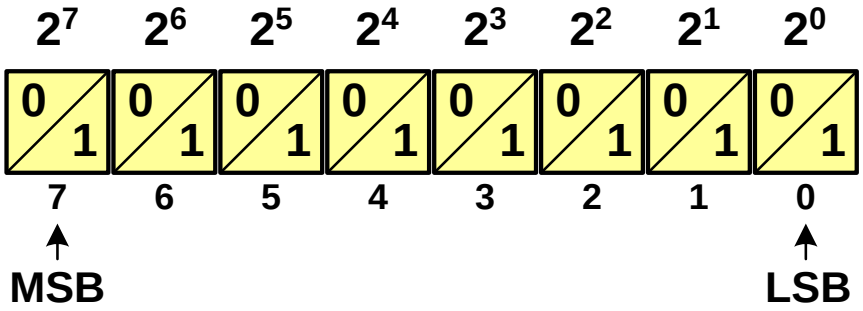
| Nazwa typu | Rozmiar (bajty) | Zakres wartości |
|---|---|---|
| unsigned char | 1 bajt | 0 … 255 |
| unsigned short int | 2 bajty | 0 … 65 535 |
| unsigned int | 4 bajty | 0 … 4 294 967 295 |
| unsigned long int | 4 bajty | 0 … 4 294 967 295 |
| unsigned long long int | 8 bajtów | 0 … 18 446 744 073 709 551 615 |

- W nazwach typów short i long można pominąć słowo int:

| | | |
|---|---|---|
| unsigned short int | → | unsigned short |
| unsigned long int | → | unsigned long |
| unsigned long long int | → | unsigned long long |

# Liczby całkowite bez znaku w języku C

- Typ unsigned char (1 bajt):

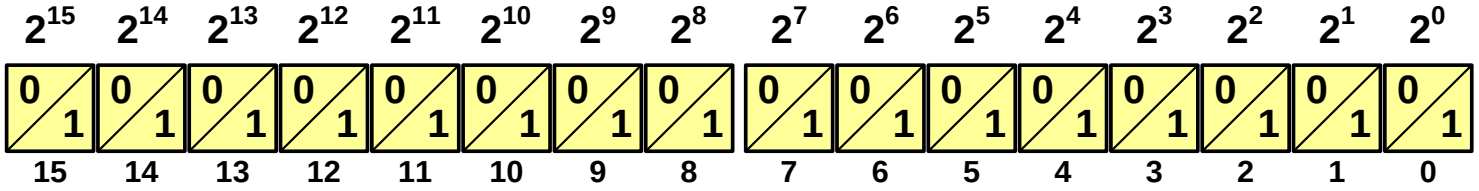



  - MSB (Most Significant Bit) - najbardziej znaczący bit, najstarszy bit, największa waga
  - LSB (Least Significant Bit) - najmniej znaczący bit, najmłodszy bit, najmniejsza waga

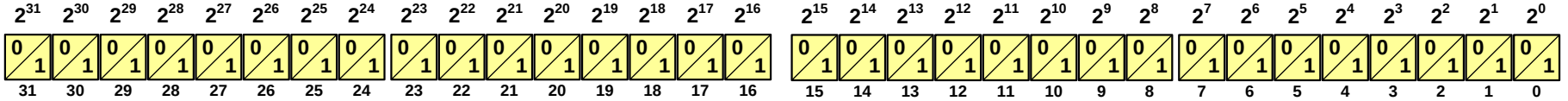
- Zakres wartości:
  - dolna granica: $0000\,0000_{(2)} = 00_{(16)} = 0_{(10)}$
  - górna granica: $1111\,1111_{(2)} = FF_{(16)} = 255_{(10)}$

# Liczby całkowite bez znaku w języku C

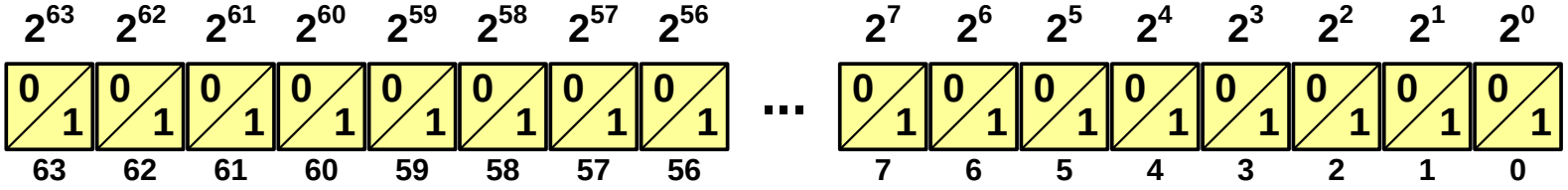
- Typ unsigned short int (2 bajty):




- Typy unsigned int (4 bajty) i unsigned long int (4 bajty):




- Typ unsigned long long int (8 bajtów):



# Liczby całkowite bez znaku w języku C

```
unsigned short int:      65535 0 1
unsigned int:            4294967295 0 1
unsigned long int:       4294967295 0 1
unsigned long long int: 18446744073709551615 0 1
```

```
#include <stdi

int main()    /* przepełnienie zmiennej, ang. integer overflow */
{
   unsigned short int      usi = 65535;
   unsigned int             ui = 4294967295;
   unsigned long int       uli = 4294967295;
   unsigned long long int ulli = 18446744073709551615;

   printf("unsigned short int:     %hu %hu %hu\n",usi,usi+1,usi+2);
   printf("unsigned int:           %u %u %u\n",ui,ui+1,ui+2);
   printf("unsigned long int:      %lu %lu %lu\n",uli,uli+1,uli+2);
   printf("unsigned long long int: %llu %llu %llu\n",
                                    ulli,ulli+1,ulli+2);
   return 0;
}
```

# Liczby całkowite bez znaku w języku C

```
unsigned short int:      1 0 65535
unsigned int:            1 0 4294967295
unsigned long int:       1 0 4294967295
unsigned long long int: 1 0 18446744073709551615
```

```
#include <stdi

int main()    /* przepełnienie zmiennej, ang. integer overflow */
{
   unsigned short int      usi = 1;
   unsigned int             ui = 1;
   unsigned long int       uli = 1;
   unsigned long long int ulli = 1;

   printf("unsigned short int:      %hu %hu %hu\n",usi,usi-1,usi-2);
   printf("unsigned int:            %u %u %u\n",ui,ui-1,ui-2);
   printf("unsigned long int:       %lu %lu %lu\n",uli,uli-1,uli-2);
   printf("unsigned long long int: %llu %llu %llu\n",
                                     ulli,ulli-1,ulli-2);
   return 0;
}
```

# Liczby całkowite ze znakiem - kod U2 w języku C

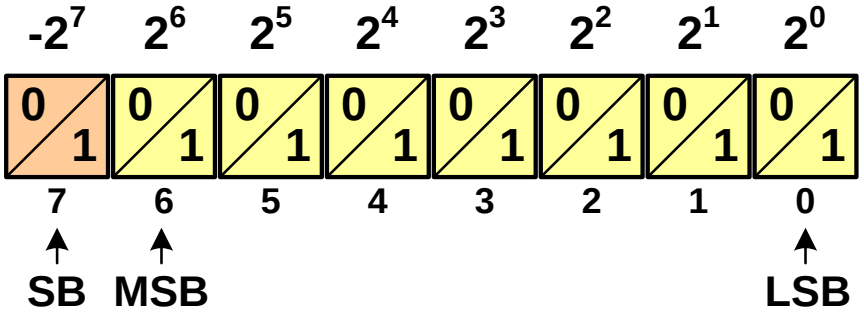
- Typy zmiennych całkowitych ze znakiem stosowane w języku C:

| Nazwa typu | Rozmiar (bajty) | Zakres wartości |
|---|---|---|
| char | 1 bajt | -128 … 127 |
| short int | 2 bajty | -32 768 … 32 767 |
| int | 4 bajty | -2 147 483 648 … 2 147 483 647 |
| long int | 4 bajty | -2 147 483 648 … 2 147 483 647 |
| long long int | 8 bajtów | -9 223 372 036 854 775 808 … 9 223 372 036 854 775 807 |

- Przed nazwą każdego z powyższych typów można dodać signed
  signed char, signed short int, signed int …
- W nazwach typów short i long można pominąć słowo int:
  short int → short, long int → long, long long int → long long

# Liczby całkowite ze znakiem - kod U2 w języku C

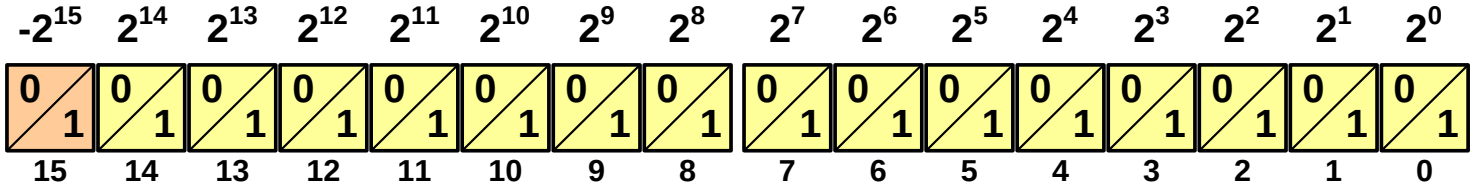
- Typ char / signed char (1 bajt):



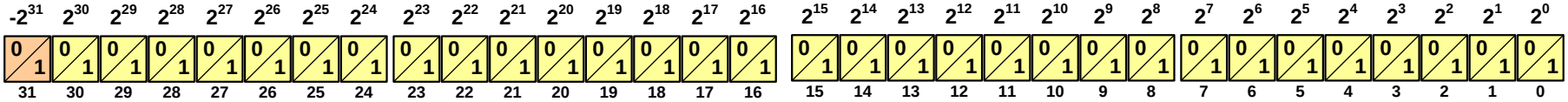

- Zakres wartości:
  - dolna granica: $1000\,0000_{(2)} = -128_{(10)}$
  - górna granica: $0111\,1111_{(2)} = 127_{(10)}$
  - inne wartości: $1111\,1111_{(2)} = -1_{(10)}$
    $0000\,0000_{(2)} = 0_{(10)}$

# Liczby całkowite bez znaku w języku C

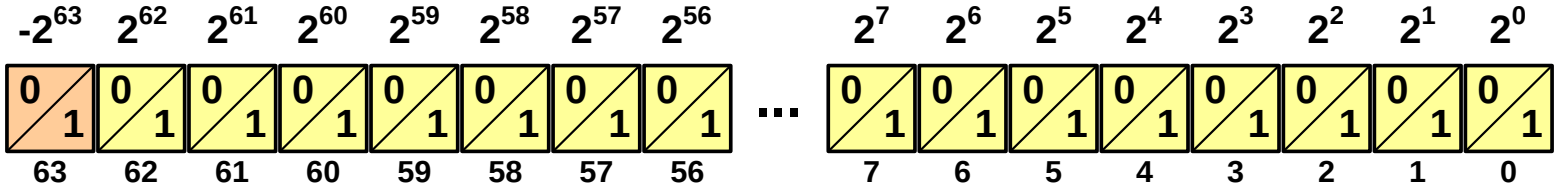
- Typ short / signed short int (2 bajty):




- Typy int / signed int (4 bajty) i long / signed long int (4 bajty):




- Typ long long int / signed long long int (8 bajtów):



# Liczby całkowite ze znakiem - kod U2 w języku C

```
short int:      32767 -32768 -32767
int:            2147483647 -2147483648 -2147483647
long int:       2147483647 -2147483648 -2147483647
long long int: 9223372036854775807 -9223372036854775808
```

```c
#include

int main()   /* przepełnienie zmiennej, ang. integer overflow */
{
   short int      si = 32767;
   int             i = 2147483647;
   long int       li = 2147483647;
   long long int lli = 9223372036854775807;

   printf("short int:      %hd %hd %hd\n",si,si+1,si+2);
   printf("int:            %d %d %d\n",i,i+1,i+2);
   printf("long int:       %ld %ld %ld\n",li,li+1,li+2);
   printf("long long int: %lld %lld\n",lli,lli+1);

   return 0;
}
```

# Zapis zmiennoprzecinkowy liczby rzeczywistej

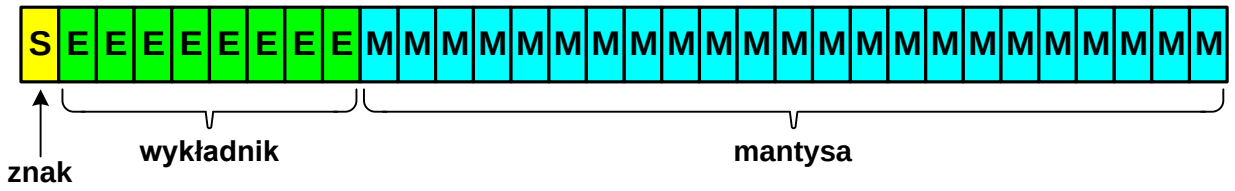
- Postać zmiennoprzecinkowa zapisu liczby: $1{,}2 \cdot 10^{13}$ $-4{,}53 \cdot 10^{-12}$
- Elementy zapisu:

$$L = (-1)^S \cdot M \cdot B^E$$

gdzie:

S - znak liczby (ang. sign), przyjmuje wartość 0 lub 1
M - mantysa (ang. mantissa), liczba ułamkowa
B - podstawa systemu liczbowego (ang. base)
E - wykładnik (ang. exponent), cecha, liczba całkowita

- W systemie binarnym




- Podstawa systemu jest stała: B = 2
- BIAS - przesunięcie wykładnika:
127 (format 32-bit.), 1023 (format 64-bit.)

$$L = (-1)^S \cdot M \cdot 2^{E-BIAS}$$

# Postać znormalizowana zapisu liczby

- Tę samą liczbę można zapisać w różnych sposób

$$243 \cdot 10^1 = 24{,}3 \cdot 10^2 = 2{,}43 \cdot 10^3 = 0{,}243 \cdot 10^4$$

- W postaci znormalizowanej mantysa spełnia nierówność:

$$B > |M| \geq 1$$

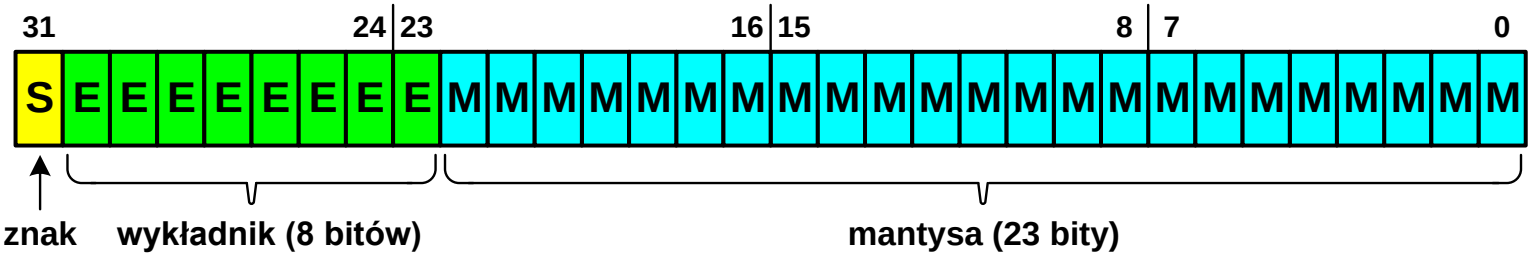
$2{,}43 \cdot 10^3$ - to jest postać znormalizowana, gdyż: $10 > |2{,}43| \geq 1$

$0{,}243 \cdot 10^4$ - to nie jest postać znormalizowana

$24{,}3 \cdot 10^2$ - to nie jest postać znormalizowana

# Standard IEEE 754 - liczby 32-bitowe

- Liczba pojedynczej precyzji przechowywana jest na 32 bitach:




- Bit znaku: 0 - liczba dodatnia, 1 - liczba ujemna
- Wykładnik zapisywany jest na z nadmiarem o wartości 127 i przyjmuje wartości od -127 do 128
- Mantysa w większości przypadków jest znormalizowana
- Mantysa zawiera się w przedziale 1 i 2 ($2>|M|\geq 1$), jej pierwszy bit jest zawsze równy 1 i nie jest zapamiętywany
- Bit ten jest automatycznie uwzględniany podczas wykonywania obliczeń

# Standard IEEE 754 - liczby 32-bitowe

- Przykład:
  - obliczmy wartość dziesiętną liczby zmiennoprzecinkowej

$$0100001011\ 0010000000\ 0000000000\ 00_{(\text{IEEE}\,754)} = ?_{(10)}$$

  - dzielimy liczbę na części

$$\underbrace{0}_{\text{S}-\text{bit znaku}}\ \underbrace{10000101}_{\text{E}-\text{wykladnik}}\ \underbrace{10010000000000000000000}_{\text{M}-\text{mantysa (tylko czesc ulamkowa)}}$$

  - określamy znak liczby

$$\text{S} = 0 \quad -\text{liczba dodatnia}$$

  - obliczamy wykładnik pamiętając, że w reprezentacji 32-bitowej nadmiar wynosi 127

$$\text{E} = 10000101_{(2)} = 128 + 4 + 1 = 133 - \underbrace{127}_{\text{nadmiar}} = 6_{(10)}$$

# Standard IEEE 754 - liczby 32-bitowe

- Przykład (cd.):
  - wyznaczamy mantysę dopisując na początku 1 (1 - część całkowita) i stawiając przecinek

$$M = 1,10010000000000000000000 =$$

$$= 1 \cdot 2^{0} + 1 \cdot 2^{-1} + 1 \cdot 2^{-4} = 1 + 0,5 + 0,0625 = 1,5625_{(10)}$$

  - wartość dziesiętną liczby zmiennoprzecinkowej obliczamy według wzoru:

$$L = (-1)^{S} \cdot M \cdot 2^{E}$$

  - podstawiając otrzymujemy:

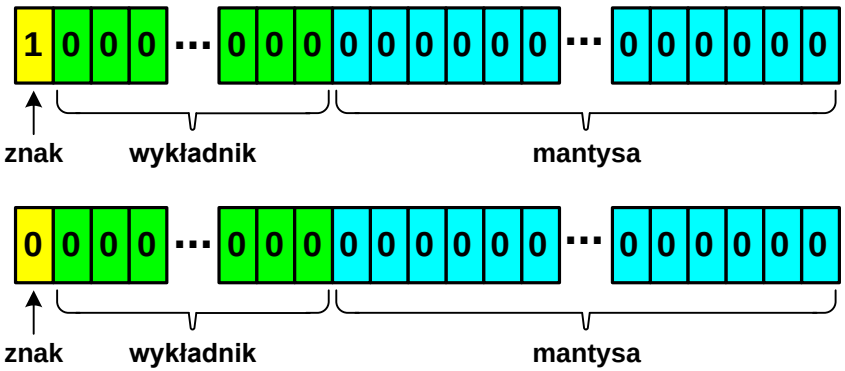
$$S = 0, \qquad E = 6_{(10)}, \qquad M = 1,5625_{(10)}$$

$$L = (-1)^{0} \cdot 1,5625 \cdot 2^{6} = 100_{(10)}$$

$$0100001011\ 0010000000\ 0000000000\ 00_{(\text{IEEE}\,754)} = 100_{(10)}$$

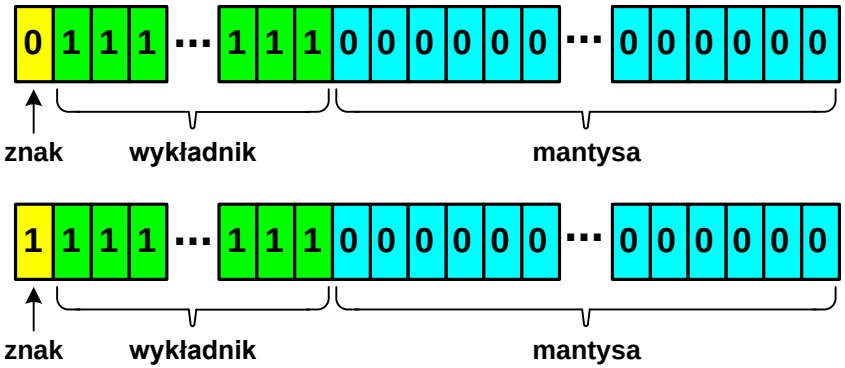
# Standard IEEE 754 - wartości specjalne

- Zero:




- zero ujemne
- zero dodatnie

- Nieskończoność:




- nieskończoność dodatnia
- nieskończoność ujemna

- Liczba zdenormalizowana, nieliczby (QNaN, SNaN)

# Standard IEEE 754 - wartości specjalne

- Standard IEEE 754 definiuje dokładnie wyniki operacji, w których występują specjalne argumenty

| Operacja | Wynik |
|---|---|
| x / ±∞ | 0 |
| ±∞ · ±∞ | ±∞ |
| ±wart_niezer / 0 | ±∞ |
| ∞ + ∞ | ∞ |
| ±0 / ±0 | NaN |
| ∞ - ∞ | NaN |
| ±∞ / ±∞ | NaN |
| ±∞ · 0 | NaN |

# Język C - operacje z wartościami specjalnymi

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

int main(void)
{
  float x = 0.0;
  printf("1.0/0.0     = %f\n",1.0/x);
  printf("-1.0/0.0    = %f\n",-1.0/x);
  printf("0.0/0.0     = %f\n",0.0/x);
  printf("sqrt(-1.0)  = %f\n",sqrt(-1.0));
  printf("1.0/INF     = %f\n",1.0/(1.0/x));
  printf("0*INF       = %f\n",0.0*(1.0/x));

  return 0;
}
```

```
1.0/0.0     =  1.#INF00
-1.0/0.0    = -1.#INF00
0.0/0.0     = -1.#IND00
sqrt(-1.0)  = -1.#IND00
1.0/INF     =  0.000000
0*INF       = -1.#IND00
```

- Środowisko: Microsoft Visual C++ 2008 Express Edition

| Operacja | Wynik |
|---|---|
| x / ±∞ | 0 |
| ±∞ · ±∞ | ±∞ |
| ±wart_niezer / 0 | ±∞ |
| ∞ + ∞ | ∞ |
| ±0 / ±0 | NaN |
| ∞ - ∞ | NaN |
| ±∞ / ±∞ | NaN |
| ±∞ · 0 | NaN |

# Język C - operacje z wartościami specjalnymi

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

int main(void)
{
  printf("1.0/0.0     = %f\n",1.0/0.0);
  printf("-1.0/0.0    = %f\n",-1.0/0.0);
  printf("0.0/0.0     = %f\n",0.0/0.0);
  printf("sqrt(-1.0)  = %f\n",sqrt(-1.0));
  printf("1.0/INF     = %f\n",1.0/(1.0/0.0));
  printf("0*INF       = %f\n",0.0*(1.0/0.0));

  return 0;
}
```

```
1.0/0.0     =  1.#INF00
-1.0/0.0    = -1.#INF00
0.0/0.0     = -1.#IND00
sqrt(-1.0)  = -1.#IND00
1.0/INF     =  0.000000
0*INF       = -1.#IND00
```

- Środowisko: Code::Blocks 20.03

| Operacja | Wynik |
|---|---|
| x / ±∞ | 0 |
| ±∞ · ±∞ | ±∞ |
| ±wart_niezer / 0 | ±∞ |
| ∞ + ∞ | ∞ |
| ±0 / ±0 | NaN |
| ∞ - ∞ | NaN |
| ±∞ / ±∞ | NaN |
| ±∞ · 0 | NaN |

# Reprezentacja liczb zmiennoprzecinkowych w C

- Typy zmiennoprzecinkowe w języku C:

| Nazwa typu | Rozmiar (bajty) | Zakres wartości | Cyfry znaczące |
|---|---|---|---|
| float | 4 bajty | $-3{,}4\cdot10^{38}$ … $3{,}4\cdot10^{38}$ | 7-8 |
| double | 8 bajtów | $-1{,}8\cdot10^{308}$ … $1{,}8\cdot10^{308}$ | 15-16 |
| long double | 10 bajtów | $-1{,}2\cdot10^{4932}$ … $1{,}2\cdot10^{4932}$ | 19-20 |

- Typ long double może mieć także inny rozmiar:

| Środowisko | Rozmiar (bajty) |
|---|---|
| MS Visual C++ 2008 EE | 8 bajtów |
| Borland Turbo C++ Explorer | 10 bajtów |
| Code:Blocks 20.03 | 16 bajtów (*) |
| Dev-C++ 5.11 | 16 bajtów (*) |

# Reprezentacja liczb zmiennoprzecinkowych w C

```
#include <stdio.h>

int main(void)
{
    float       sf  = 0.0f;
    double      sd  = 0.0;
    long double slg = 0.0L;

    for (int i=0; i<10000; i++)
    {
        sf  = sf  + 0.01f;
        sd  = sd  + 0.01;
        slg = slg + 0.01L;
    }

    printf("float:       %.20f\n",sf);
    printf("double:      %.20f\n",sd);
    printf("long double: %.20Lf\n",slg);

    return 0;
}
```

# Reprezentacja liczb zmiennoprzecinkowych w C

- Microsoft Visual C++ 2008 Express Edition (long double - 8 bajtów)

```
float:       100.00295257568359000000
double:      100.00000000001425000000
long double: 100.00000000001425000000
```

- Borland Turbo C++ Explorer (long double - 10 bajtów)

```
float:       100.00295257568359375000
double:      100.00000000001425349000
long double: 100.00000000000001388000
```

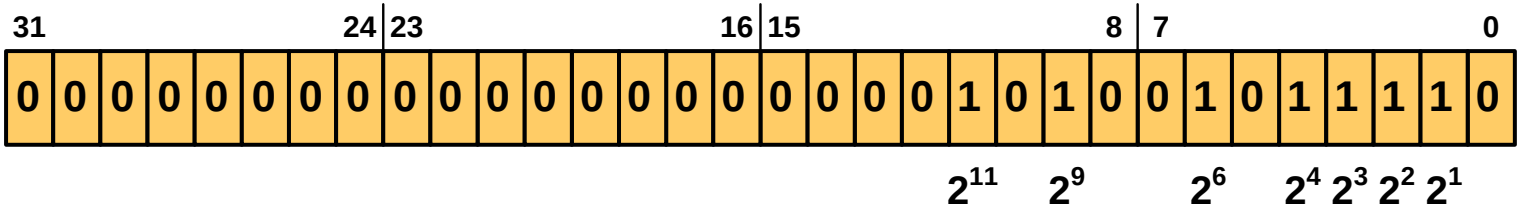
- Code::Blocks 20.03 (long double - 16 bajtów)

```
float:       100.00295257568359000000
double:      100.00000000001425000000
long double: 0.00000000000000000000
```

```
warning: unknown conversion
type character 'L' in format
[-Wformat=]|
```

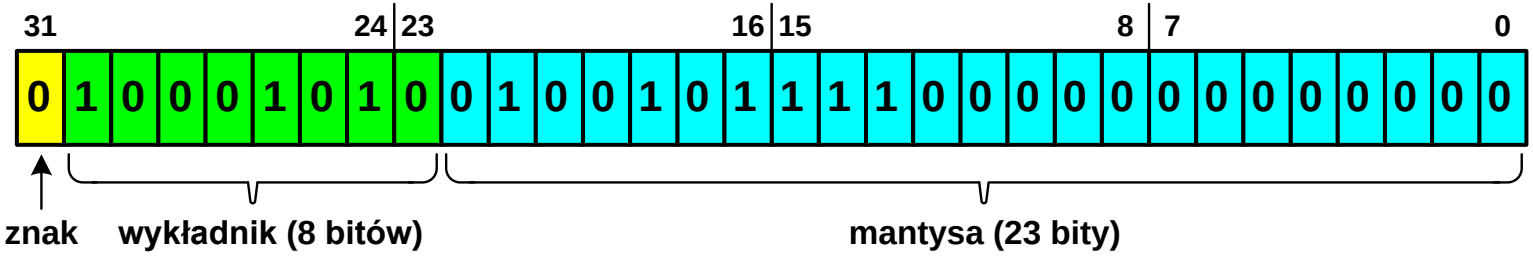
# Liczba $2654_{(10)}$ jako całkowita i rzeczywista w C

- int (4 bajty): $2654_{(10)} = 00\ 00\ 0A\ 5E_{(16)}$




$$2^{11} + 2^9 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 = 2048 + 512 + 64 + 16 + 8 + 4 + 2 = 2654_{(10)}$$

- float (4 bajty): $2654_{(10)} = 45\ 25\ E0\ 00_{(IEEE\ 754)}$




$+ \quad 138 - 127 = 11_{(10)}$ $\qquad 1\ .0100101111_{(2)} = 1.2958984_{(10)}$

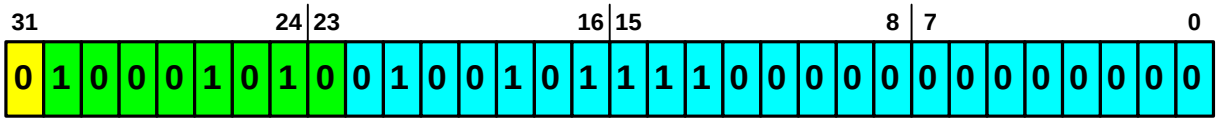
$$1.2958984 \cdot 2^{11} = 2654_{(10)}$$

# Język C - nieprawidłowy specyfikator formatu

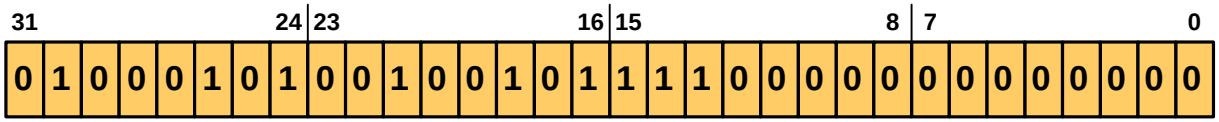
```
int x;
printf("x (%%f) = "); scanf("%f",&x);
printf("x (%%d) = %d\n",x);
printf("x (%%f) = %f\n",x);
printf("x (%%e) = %e\n",x);
```

```
x (%f) = 2654
x (%d) = 1160110080
x (%f) = 0.000000
x (%e) = 5.731705e-315
```

- Zgodnie ze standardem języka C wynik jest niezdefiniowany
- Zapamiętana wartość:




- Wyświetlona wartość przy wykorzystaniu %d:



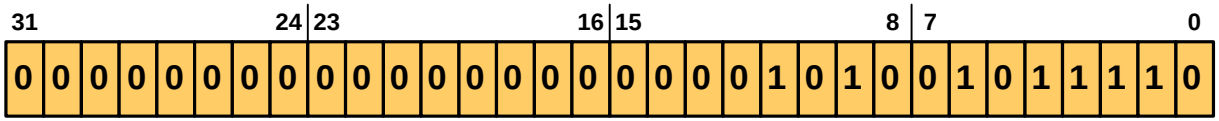

$$2^{30} + 2^{26} + 2^{24} + 2^{21} + 2^{18} + 2^{16} + 2^{15} + 2^{14} + 2^{13} = 1.160.110.080_{(10)}$$

# Język C - nieprawidłowy specyfikator formatu

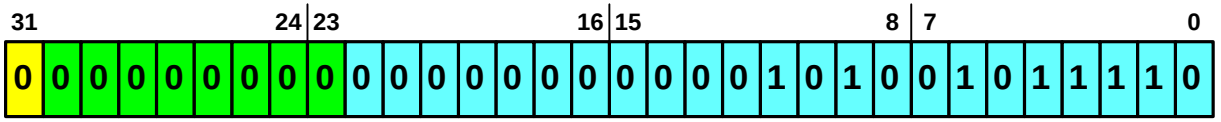
```
float x;
printf("x (%%d) = "); scanf("%d",&x);
printf("x (%%d) = %d\n",x);
printf("x (%%f) = %f\n",x);
printf("x (%%e) = %e\n",x);
```

```
x (%d) = 2654
x (%d) = 0
x (%f) = 0.000000
x (%e) = 3.719046e-042
```

- Zgodnie ze standardem języka C wynik jest niezdefiniowany
- Zapamiętana wartość:




- Wyświetlona wartość przy wykorzystaniu %e:




**Liczba zdenormalizowana: 3,719046E-42**

# Koniec wykładu nr 3

## Dziękuję za uwagę!

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Materiały przygotowane w ramach projektu „PB 5.0 - dostosowanie oferty dydaktycznej Politechniki Białostockiej do potrzeb nowoczesnej gospodarki oraz zielonej i cyfrowej transformacji” (nr umowy FERS.01.05-IP.08-0327/23-00) w ramach programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027 współfinansowanego ze środków Europejskiego Funduszu Społecznego Plus.