



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



Wydział Elektryczny  
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Materiały do wykładu z przedmiotu:

**Informatyka**

**Kod: EDS1B1007**

**WYKŁAD NR 2**

**Opracował: dr inż. Jarosław Forenc**

**Białystok 2024**

Materiały zostały opracowane w ramach projektu „PB2020 - Zintegrowany Program Rozwoju Politechniki Białostockiej” realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

## Plan wykładu nr 2

- Deklaracje zmiennych i stałych
- Operatory, priorytet operatorów
- Wyrażenia, instrukcje
- Wyrażenia arytmetyczne, funkcje matematyczne (math.h)
- Funkcje printf() i scanf()
- Instrukcja if, operatory relacyjne i logiczne, wyrażenia logiczne
- Operator warunkowy

## Przykład: zamiana wzrostu w cm na stopy i cale

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

```
int main(void)  
{
```

```
    float cm;        /* wzrost w cm */  
    float stopy;     /* wzrost w stopach */  
    float cale;      /* wzrost w calach */
```

```
    printf("Podaj wzrost w cm: ");  
    scanf("%f", &cm);
```

```
    stopy = cm / 30.48f;  
    cale = cm / 2.54f;
```

```
    printf("%f [cm] = %f [ft]\n", cm, stopy);  
    printf("%f [cm] = %f [in]\n", cm, cale);
```

```
    return 0;  
}
```

```
Podaj wzrost w cm: 175  
175.000000 [cm] = 5.741470 [ft]  
175.000000 [cm] = 68.897636 [in]
```

## Język C - deklaracje zmiennych i stałych

- **Zmienne** (ang. variables) - zmieniają swoje wartości podczas pracy programu
- **Stałe** (ang. constants) - mają wartości ustalone przed uruchomieniem programu i pozostają niezmiennione przez cały czas jego działania
- **Deklaracja** nadaje zmiennej / stałej nazwę, określa typ przechowywanej wartości i rezerwuje odpowiednio obszar pamięci

- Deklaracje zmiennych:

```
int x;  
float a, b;  
char zn1;
```

- Deklaracje stałych:

```
const int y = 5;  
const float c = 1.25f;  
const char zn2 = 'Q';
```

- **Inicjalizacja** zmiennej:

```
int x = -10;
```

## Język C - stałe symboliczne (#define)

- Dyrektywa preprocesora **#define** umożliwia definiowanie tzw. stałych symbolicznych

**#define nazwa\_stalej wartość\_stalej**

```
#define PI 3.14  
#define KOMUNIKAT "Zaczynamy!!!\n"
```

- Wyrażenia stałe zazwyczaj pisze się wielkimi literami
- W miejscu występowania stałej wstawiana jest jej wartość (przed właściwą kompilacją programu)

## Przykład: pole i obwód koła

```
#include <stdio.h>
#define PI 3.14
#define KOMUNIKAT "Zaczynamy!!!\n"

int main(void)
{
    double pole, obwod;
    double r = 1.5;

    printf(KOMUNIKAT);
    pole = PI * r * r;
    obwod = 2 * PI * r;

    printf("Pole = %g\n", pole);
    printf("Obwod = %g\n", obwod);

    return 0;
}
```

## Przykład: pole i obwód koła

```
/**  
...  
#endif /* _INC_STDIO */  
  
int main(void)  
{  
    double pole, obwod;  
    double r = 1.5;  
  
    printf("Zaczynamy!!!\n");  
    pole = 3.14 * r * r;  
    obwod = 2 * 3.14 * r;  
  
    printf("Pole = %g\n", pole);  
    printf("Obwod = %g\n", obwod);  
  
    return 0;  
}
```

Zaczynamy!!!  
Pole = 7.065  
Obwod = 9.42

zawartość pliku stdio.h

# Język C - Operatory

- **Operator** - symbol lub nazwa operacji
- Argumenty operatora nazywane są **operandami**
- Operator jednoargumentowy



- Operator dwuargumentowy



- Operator trójargumentowy



- Operator wieloargumentowy





# Język C - operatory

Typ	Symbol
Arytmetyczne	+ - * / %
Inkrementacji / dekrementacji	++ --
Porównania (relacyjne)	< > <= >= == !=
Logiczne	&&    !
Bitowe	&   ^ << >> ~
Przypisania	= += -= *= /= %= <<= >>= &=  = ^=
Inne	() [] & * -> . , ? : sizeof (typ)

## Język C - priorytet operatorów (1/2)

Priorytet	Operator / opis
1	++ -- (przyrostki) () [] . ->
2	++ -- (przedrostki) sizeof (typ) + - ! ~ * & (jednoargumentowe)
3	* / %
4	+ - (dwuargumentowe)
5	<< >>
6	< > <= >=
7	== !=
8	& (bitowy)
9	^

## Język C - priorytet operatorów (2/2)

Priorytet	Operator / opis
10	
11	&&
12	
13	? :
14	= += -= *= /= %= <<= >>= &=  = ^=
15	, (przecinek)

## Język C - wyrażenia

- **Wyrażenie** (ang. expression) - kombinacja operatorów i operandów

```
4      -6      4+2.1      x=5+2      a>3      x>5&& x<8
```

- Każde wyrażenie ma **typ** i **wartość**

Wyrażenie	Typ	Wartość
4	int	4
-6	int	-6
4 + 2.1	double	6.1
x = 5 + 2	<i>typ x</i>	7
a > 3	int	1 (prawda) / 0 (fałsz)
x > 5 && x < 8	int	1 (prawda) / 0 (fałsz)

## Język C - instrukcje

- **Instrukcja** (ang. statement) - główny element, z którego zbudowany jest program, kończy się średnikiem

Wyrażenie:

`x = 5`

Instrukcja:

`x = 5;`

- Język C za instrukcję uznaje każde wyrażenie, na którego końcu znajduje się średnik

```
8;  
x;  
3 + 4;  
a > 5;
```

- Powyższe instrukcje są poprawne, ale nie dają żadnego efektu

# Język C - instrukcje

## ■ Podział instrukcji:

- **proste** - kończą się średnikiem
- **złożone** - kilka instrukcji zawartych pomiędzy nawiasami klamrowymi

## ■ Typy instrukcji prostych:

- deklaracji:

```
int x;
```

- przypisania:

```
x = 5;
```

- wywołania funkcji:

```
printf("Witaj świecie\n");
```

- strukturalna:

```
while(x > 0) x--;
```

- pusta:

```
;
```

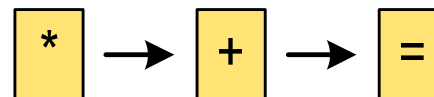
## Język C - wyrażenia arytmetyczne

- Wyrażenia arytmetyczne mogą zawierać:
  - stałe liczbowe, zmienne, stałe
  - operatory:  $+$   $-$   $*$   $/$   $\%$   $=$   $( )$  i inne
  - wywołania funkcji (plik nagłówkowy **math.h**)
- Kolejność wykonywania operacji wynika z priorytetu operatorów

```
w = a + b;
```



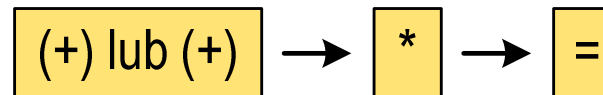
```
w = a + b * c;
```



```
w = (a + b) * c;
```



```
w = (a + b) * (c + d);
```



## Język C - wyrażenia arytmetyczne

### ■ Kolejność wykonywania operacji

$w = a + b + c;$	→	$w = ((a + b) + c);$
$w = x = y = a + b;$	→	$w = (x = (y = (a + b))) ;$

### ■ Zapis wyrażen arytmetycznych

$w = \frac{a+b}{c+d}$	$w = a + b / c + d;$	ŹLE
	$w = (a + b) / (c + d);$	DOBRZE
$w = \frac{a+b}{c \cdot d}$	$w = (a + b) / c * d;$	ŹLE
	$w = (a + b) / (c * d);$	DOBRZE



# Język C - wyrażenia arytmetyczne

- Podczas dzielenia liczb całkowitych odrzucana jest część ułamkowa

$$w = \frac{5}{4}$$

```
5 / 4 = 1
```

```
5.0 / 4 = 1.25
```

```
5 / 4.0 = 1.25
```

```
5.0 / 4.0 = 1.25
```

```
5.0f / 4 = 1.25
```

```
5. / 4 = 1.25
```

```
(float) 5 / 4 = 1.25
```

Rzutowanie: (typ)

## Język C - funkcje matematyczne (math.h)

- Plik nagłówkowy **math.h** zawiera definicje wybranych stałych

Nazwa	Wartość	Znaczenie
<b>M_PI</b>	3.14159265358979323846	liczba pi
<b>M_E</b>	2.71828182845904523536	e - liczba Eulera
<b>M_LN2</b>	0.693147180559945309417	ln 2
<b>M_SQRT2</b>	1.41421356237309504880	$\sqrt{2}$

- W środowisku Visual Studio 2008 użycie stałych wymaga definicji odpowiedniej stałej (przed **#include <math.h>**)

```
#define _USE_MATH_DEFINES  
#include <math.h>
```

# Język C - funkcje matematyczne (math.h)

- Wybrane funkcje matematyczne:

Nazwa	Nagłówek	Znaczenie
<b>abs</b>	<code>int abs(int x);</code>	moduł x (x - całkowite)
<b>fabs</b>	<code>double fabs(double x);</code>	moduł x (x - rzeczywiste)
<b>sqrt</b>	<code>double sqrt(double x);</code>	pierwiastek kwadratowy x
<b>pow</b>	<code>double pow(double x, double y);</code>	$x^y$ - x do potęgi y
<b>sin</b>	<code>double sin(double x);</code>	sinus argumentu x w radianach
<b>atan</b>	<code>double atan(double x);</code>	arcus tangens argumentu x
<b>atan2</b>	<code>double atan2(double y, double x);</code>	arcus tangens ilorazu y/x

- Wszystkie funkcje mają po trzy wersje - dla argumentów typu: **float**, **double** i **long double**

## Przykład: częstotliwość rezonansowa

```
#include <stdio.h>
#define _USE_MATH_DEFINES
#include <math.h>

int main(void)
{
    double L, C, fr;

    printf("Podaj L [H]: "); scanf("%lf", &L);
    printf("Podaj C [F]: "); scanf("%lf", &C);

    fr = 1 / (2 * M_PI * sqrt(L * C));

    printf("-----\n");
    printf("fr [Hz]: %.3f\n", fr);

    return 0;
}
```

```
Podaj L [H]: 0.01
Podaj C [F]: 1e-6
-----
fr [Hz]: 1591.549
```

$$f_r = \frac{1}{2\pi\sqrt{LC}}$$

## Język C - Funkcja printf()

- Ogólna składnia funkcji `printf()`

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

- W najprostszej postaci `printf()` wyświetla tylko tekst

```
printf("Witaj świecie");
```

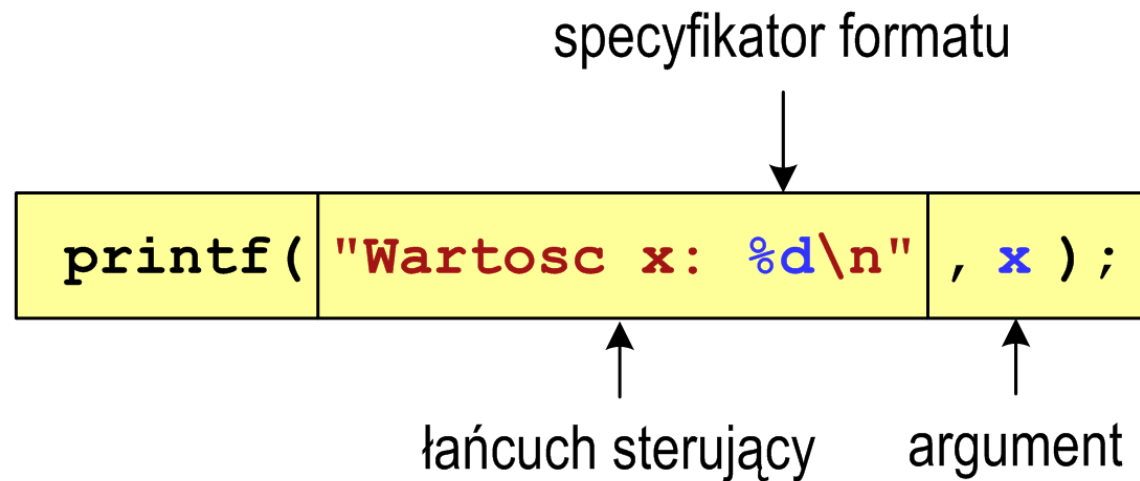
```
Witaj świecie
```

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

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

## Język C - Funkcja printf()

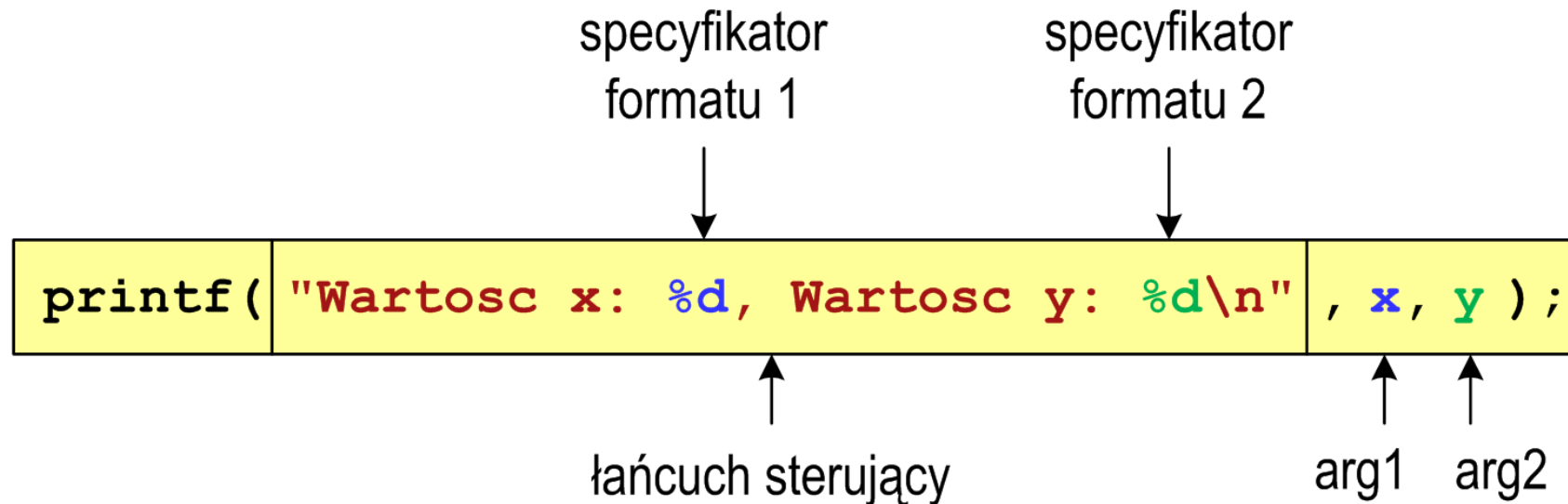
```
int x = 10;  
printf("Wartosc x: %d\n", x);
```



```
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
<b>char</b>	<b>%c</b>	pojedynczy znak
	<b>%d</b>	kod ASCII znaku, liczba całkowita
<b>char *</b>	<b>%s</b>	łańcuch znaków, napis
<b>int</b>	<b>%d %i</b>	liczba całkowita, dziesiętna
	<b>%o %O</b>	liczba całkowita, ósemkowa
	<b>%x %X</b>	liczba całkowita, szesnastkowa
<b>float</b> <b>double</b>	<b>%f</b>	liczba rzeczywista
	<b>%e %E</b>	liczba rzeczywista, format naukowy
	<b>%g %G</b>	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 ("specyfikator", 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	<b>%lf</b>	liczba rzeczywista
	<b>%le %LE</b>	liczba rzeczywista, format naukowy
	<b>%lg %LG</b>	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

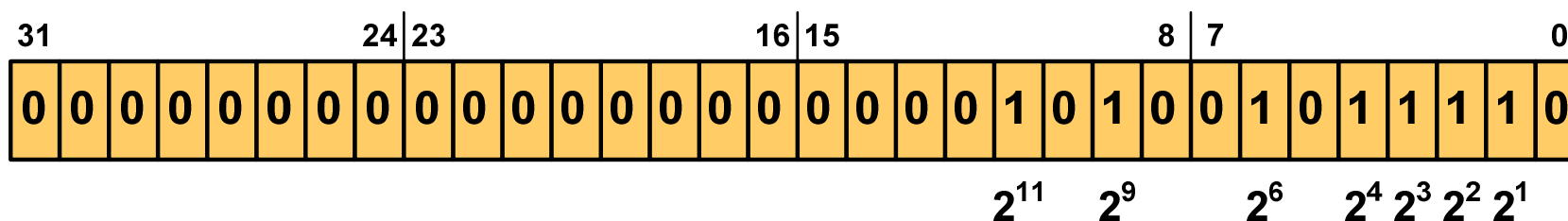
15 20 -30<enter>

15  
20  
-30

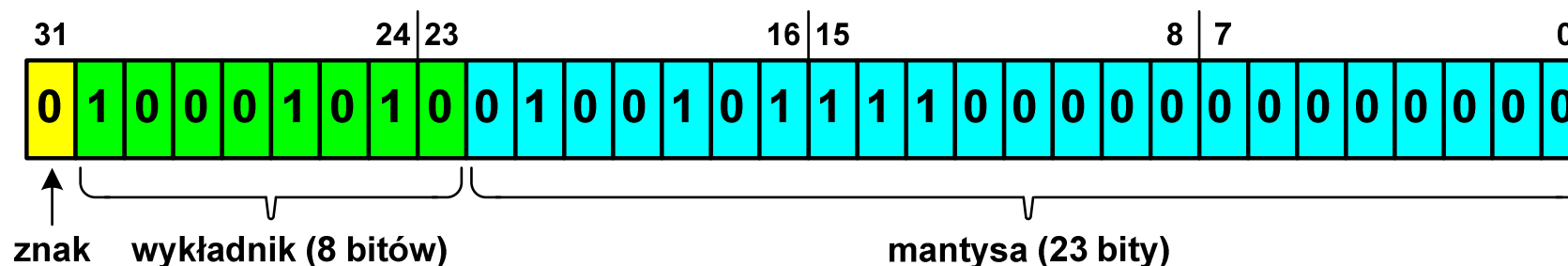
15<enter>  
20<enter>  
-30<enter>

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

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



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



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

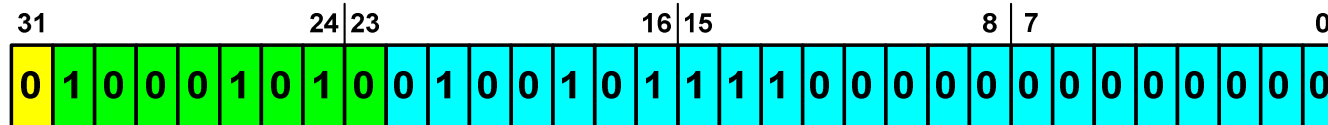


# Język C - nieprawidłowy specyfikator formatu

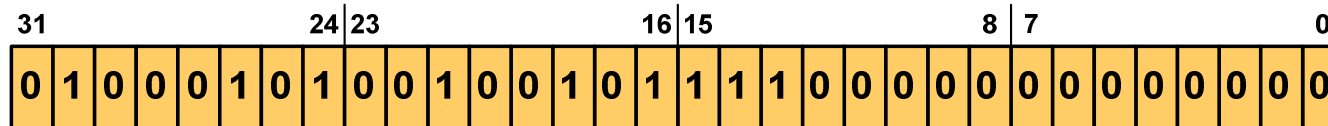
```
int x;  
  
printf("x (%f) = "); scanf("%f", &x);  
printf("x (%d) = %d\n", x);  
printf("x (%f) = %f\n", x);  
printf("x (%e) = %e\n", x);
```

```
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**:



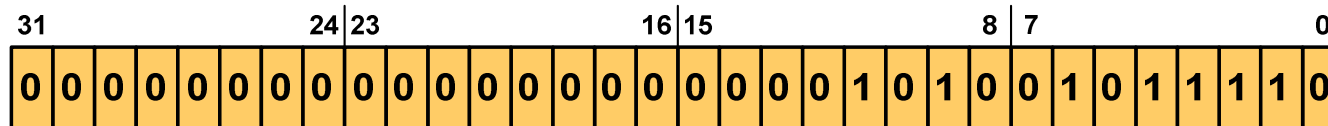
$$2^{30} + 2^{26} + 2^{24} + 2^{21} + 2^{18} + 2^{16} + 2^{15} + 2^{14} + 2^{13} = 1.160.110.080_{(10)}$$

# Język C - nieprawidłowy specyfikator formatu

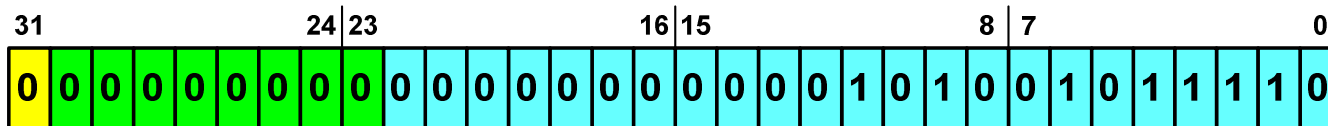
```
float x;  
  
printf("x (%d) = "); scanf("%d", &x);  
printf("x (%d) = %d\n", x);  
printf("x (%f) = %f\n", x);  
printf("x (%e) = %e\n", x);
```

```
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

## Przykład: pierwiastek kwadratowy

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

int main(void)
{
    float x, y;

    printf("Podaj liczbe: ");
    scanf("%f", &x);

    y = sqrt(x);

    printf("Pierwiastek liczby: %f\n", y);

    return 0;
}
```

```
Podaj liczbe: 15
Pierwiastek liczby: 3.872983
```

```
Podaj liczbe: -15
Pierwiastek liczby: -1.#IND00
```

## Przykład: pierwiastek kwadratowy

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

int main(void)
{
    float x, y;

    printf("Podaj liczbe: ");
    scanf("%f", &x);

    if (x >= 0)
    {
        y = sqrt(x);
        printf("Pierwiastek liczby: %f\n", y);
    }
    else
        printf("Blad! Liczba ujemna\n");

    return 0;
}
```

Podaj liczbe: 15  
Pierwiastek liczby: 3.872983

Podaj liczbe: -15  
Blad! Liczba ujemna

## Język C - instrukcja warunkowa if

```
if (wyrażenie)  
    instrukcja1;
```

- jeśli **wyrażenie** jest prawdziwe, to wykonywana jest **instrukcja1**
- gdy **wyrażenie** jest fałszywe, to **instrukcja1** nie jest wykonywana

```
if (wyrażenie)  
    instrukcja1;  
else  
    instrukcja2;
```

- jeśli **wyrażenie** jest prawdziwe, to wykonywana jest **instrukcja1**, zaś **instrukcja2** nie jest wykonywana
- gdy **wyrażenie** jest fałszywe, to wykonywana jest **instrukcja2**, zaś **instrukcja1** nie jest wykonywana

### ■ Wyrażenie w nawiasach:

- **prawdziwe** - gdy jego wartość jest różna od zera
- **fałszywe** - gdy jego wartość jest równa zero

## Język C - instrukcja warunkowa if

```
if (wyrażenie)  
    instrukcja;
```

### ■ Instrukcja:

- **prosta** - jedna instrukcja zakończona średnikiem
- **złożona** - jedna lub kilka instrukcji objętych nawiasami klamrowymi

```
if (x>0)  
    printf("inst1");
```

```
if (x>0)  
{  
    printf("inst1");  
    printf("inst2");  
    ...  
}
```

# Język C - instrukcja warunkowa if

```
if (wyr)
    instr;
```

```
if (wyr)
    instr;
else
    instr;
```

```
if (wyr)
{
    instr;
    instr;
}
else
    instr;
```

```
if (wyr)
{
    instr;
}
else
{
    instr;
}
```

```
if (wyr)
{
    instr;
    instr;
}
```

```
if (wyr)
{
    instr;
    instr;
}
else
{
    instr;
    instr;
}
```

```
if (wyr)
    instr;
else
{
    instr;
    instr;
}
```

## Język C - Operatory relacyjne (porównania)

Operator	Przykład	Znaczenie
>	<code>a &gt; b</code>	<code>a</code> większe od <code>b</code>
<	<code>a &lt; b</code>	<code>a</code> mniejsze od <code>b</code>
>=	<code>a &gt;= b</code>	<code>a</code> większe lub równe <code>b</code>
<=	<code>a &lt;= b</code>	<code>a</code> mniejsze lub równe <code>b</code>
==	<code>a == b</code>	<code>a</code> równe <code>b</code>
!=	<code>a != b</code>	<code>a</code> nierówne <code>b</code> ( <code>a</code> różne od <code>b</code> )

- Wynik porównania jest wartością typu `int` i jest równy:
  - `1` - gdy warunek jest prawdziwy
  - `0` - gdy warunek jest fałszywy (nie jest prawdziwy)



## Język C - Operatory logiczne

Operator	Znaczenie	Opis
!	NOT, nie	jednoargumentowy operator negacji logicznej - zmienia argument różny od zera na wartość 0, a argument równy zero na wartość 1
&&	AND, i	dwuargumentowy operator koniunkcji, iloczyn logiczny
	OR, lub	dwuargumentowy operator alternatywy, suma logiczna

- Wynikiem zastosowania operatorów logicznych `&&` i `||` jest wartość typu `int` równa 1 (prawda) lub 0 (fałsz)

```
if (x>5 && x<8)
```

```
if (x<=5 || x>8)
```

# Język C - Wyrażenia logiczne

- Wyrażenia logiczne mogą zawierać:

- operatory relacyjne
- operatory logiczne
- operatory arytmetyczne
- operatory przypisania
- zmienne
- stałe
- wywołania funkcji
- ...

- Kolejność operacji wynika z **priorytetu operatorów**

Operator	Typ operatora
!	logiczny
* / %	arytmetyczne
+ -	arytmetyczne
> < >= <=	relacyjne
== !=	relacyjne
&&	logiczny
	logiczny
=	przypisania

## Język C - Wyrażenia logiczne

```
int x = 0, y = 1, z = 2;
```

```
if ( x == 0 )
```

wynik: 1 (prawda)

```
if ( x = 0 )
```

wynik: 0 (fałsz) (!!!)

```
if ( x != 0 )
```

wynik: 0 (fałsz)

```
if ( x =! 0 )
```

wynik: 1 (prawda) (!!!)

```
if ( z > x + y )
```

wynik: 1 (prawda)

```
if ( z > (x + y) )
```

## Język C - Wyrażenia logiczne

```
int x = 0, y = 1, z = 2;
```

```
if ( x>2 && x<5 )
```

```
if ( (x>2) && (x<5) )
```

wynik: 0 (fałsz)

- Wyrażenia logiczne obliczane są od strony lewej do prawej
- Proces obliczeń kończy się, gdy wiadomo, jaki będzie wynik całego wyrażenia

```
if ( 2 < x < 5 )
```

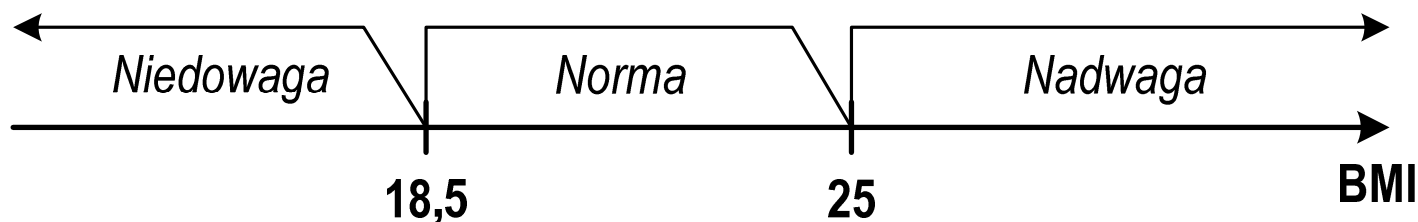
wynik: 1 (prawda) (!!!)

## Przykład: obliczanie BMI (Body Mass Index)

- **BMI** - współczynnik powstały przez podzielenie **masy** ciała podanej w kilogramach przez **kwadrat wzrostu** podanego w metrach

$$BMI = \frac{masa}{wzrost^2}$$

- Dla osób dorosłych:
  - BMI < 18,5 - wskazuje na niedowagę
  - BMI ≥ 18,5 i BMI < 25 - wskazuje na prawidłową masę ciała
  - BMI ≥ 25 - wskazuje na nadwagę



## Przykład: obliczanie BMI (Body Mass Index)

```
#include <stdio.h>

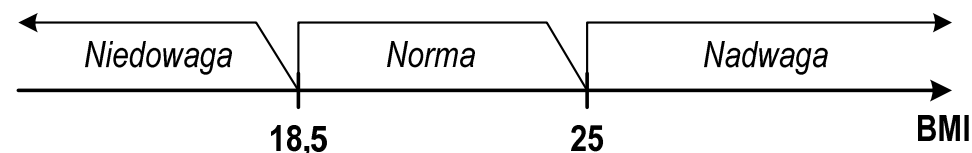
int main(void)
{
    double masa, wzrost, bmi;

    printf("Podaj mase [kg]: "); scanf("%lf", &masa);
    printf("Podaj wzrost [m]: "); scanf("%lf", &wzrost);
    bmi = masa / (wzrost*wzrost);
    printf("bmi: %.2f\n", bmi);

    if (bmi < 18.5)
        printf("Niedowaga\n");
    if (bmi >= 18.5 && bmi < 25)
        printf("Norma\n");
    if (bmi >= 25)
        printf("Nadwaga\n");

    return 0;
}
```

```
Podaj mase [kg]: 84
Podaj wzrost [m]: 1.85
bmi: 24.54
Norma
```



## Przykład: obliczanie BMI (Body Mass Index)

- Zamiast trzech instrukcji `if`:

```
if (bmi<18.5)
    printf("Niedowaga\n");
if (bmi>=18.5 && bmi<25)
    printf("Norma\n");
if (bmi>=25)
    printf("Nadwaga\n");
```

można zastosować tylko dwie:

```
if (bmi<18.5)
    printf("Niedowaga\n");
else
    if (bmi<25)
        printf("Norma\n");
    else
        printf("Nadwaga\n");
```

## Język C - Operator warunkowy

- Operator warunkowy składa się z dwóch symboli i trzech operandów

```
wyrażenie1 ? wyrażenie2 : wyrażenie3
```

- Najczęściej zastępuje proste instrukcje **if-else**

```
float akcyza, cena, pojemnosc;
```

```
if (pojemnosc <= 2000)
    akcyza = cena*0.031;    /* 3.1% */
else
    akcyza = cena*0.186;    /* 18.6% */
```

```
akcyza = pojemnosc <= 2000 ? cena*0.031 : cena*0.186 ;
```



## Język C - Operator warunkowy

```
if (x < 0)
    y = -x;
else
    y = x;
```

```
y = x < 0 ? -x : x;
```

- obliczenie modułu liczby x

```
if (a > b)
    max = a;
else
    max = b;
```

```
max = a > b ? a : b;
```

- wyznaczenie max z dwóch liczb

- Operator warunkowy ma bardzo niski priorytet
- Niższy priorytet mają tylko operatory przypisania (=, +=, -=, ...) i operator przecinkowy (,)

## Przykład: operator warunkowy

- Studenci chcą dojechać z akademika do sklepu - ile taksówek powinni zamówić? (Jedna taksówka może przewieźć 4 osoby.)

```
#include <stdio.h>

int main(void)
{
    int st, taxi;

    printf("Podaj liczbe studentow: ");
    scanf("%d", &st);

    taxi = st / 4 + (st % 4 != 0 ? 1 : 0);

    printf("Liczba taxi: %d\n", taxi);

    return 0;
}
```

```
Podaj liczbe studentow: 23
Liczba taxi: 6
```

## Przykład: sprawdzenie parzystości liczby

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

```
int main(void)
```

```
{
```

```
    int x;
```

```
    printf("Podaj x: ");
```

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

```
    if (x%2==0)
```

```
        printf("Liczba parzysta\n");
```

```
    else
```

```
        printf("Liczba nieparzysta\n");
```

```
    printf("Liczba %s\n", x%2==0 ? "parzysta": "nieparzysta");
```

```
    return 0;
```

```
}
```

Podaj x: -3

Liczba nieparzysta

Liczba nieparzysta

Koniec wykładu nr 2

Dziękuję za uwagę!