

# Informatyka 1 (ES1E2009)

---

Politechnika Białostocka - Wydział Elektryczny  
Elektrotechnika, semestr II, studia stacjonarne I stopnia  
Rok akademicki 2020/2021

**Wykład nr 2 (01.03.2021)**

dr inż. Jarosław Forenc

## Plan wykładu nr 2

- Język C
  - typy danych, stałe liczbowe, deklaracje zmiennych i stałych
  - operatory, priorytet operatorów
  - wyrażenia i instrukcje, wyrażenia arytmetyczne
  - funkcje matematyczne (`math.h`)
  - funkcje `printf()` i `scanf()`
  - instrukcja warunkowa `if`, operatory relacyjne (porównania) i logiczne
  - wyrażenia logiczne

## 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 - Typy danych

Nazwa	Rozmiar (bajty)	Zakres wartości
<code>char</code>	1	-128 ... 127
<code>int</code>	4	-2147483648 ... 2147483647
<code>float</code>	4	$-3,4 \cdot 10^{38} \dots 3,4 \cdot 10^{38}$
<code>double</code>	8	$-1,7 \cdot 10^{308} \dots 1,7 \cdot 10^{308}$
<code>void</code>	-	-

- Słowa kluczowe wpływające na typy:
  - `signed` - liczba ze znakiem (dla typów `char` i `int`), np. `signed char`
  - `unsigned` - liczba bez znaku (dla typów `char` i `int`), np. `unsigned int`
  - `short`, `long`, `long long` - liczba krótka/długa (dla typu `int`), np. `short int`
  - `long` - większa precyzja (dla typu `double`), `long double`

## Język C - Typy danych

- Zależnie od środowiska programistycznego (kompilatora) zmienne typów **int** i **long double** mogą zajmować różną liczbę bajtów

<b>Środowisko</b>	<b>int (bajty)</b>	<b>long double (bajty)</b>
Microsoft Visual Studio 2008	4	8
Microsoft Visual Studio 2015	4	8
Dev-C++ 5.11	4	16*
Code::Blocks 20.03	4	16*
Borland Turbo C++ 2006	4	10
Borland C++ 3.1	2	10

## Język C - Typy danych (sizeof)

- **sizeof** - operator zwracający liczbę bajtów zajmowanych przez obiekt lub zmienną podanego typu

```
sizeof(nazwa_typu)
sizeof(nazwa_zmiennej)
sizeof nazwa_zmiennej
```

- Operator **sizeof** zwraca wartość typu **size\_t**
- Zależnie od środowiska programistycznego typ **size\_t** może odpowiadać typowi **unsigned int** lub **unsigned long int**
- W standardach C99 i C11 wprowadzono specyfikator formatu **%z**, który określa, że występujący po nim specyfikator (**d, i, o, u, x, X**) dotyczy wyświetlania wartości typu **size\_t** (np. **%zd**)

## Język C - Typy danych (sizeof)

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

```
int main(void)
```

```
{
```

```
    int x;
```

```
    printf("int: %d\n", sizeof(int));
```

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

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

```
    printf("long double: %d\n", sizeof(long double));
```

```
    return 0;
```

```
}
```

```
int: 4  
int: 4  
int: 4  
long double: 16
```

## Język C - Stałe liczbowe (całkowite)

- **Liczby całkowite** (ang. integer) domyślnie zapisywane są w systemie dziesiętnym i mają typ **int**

1	100	-125	123456
---	-----	------	--------

- Zapis liczb w innych systemach liczbowych
  - **ósemkowy**: 0 na początku, np. **011**, **024**
  - **szesnastkowy**: **0x** na początku, np. **0x2F**, **0xab**
- Przyrostki na końcu liczby zmieniają typ
  - **l** lub **L** - typ **long int**, np. **10l**, **10L**, **011L**, **0x2FL**
  - **ll** lub **LL** - typ **long long int**, np. **10ll**, **10LL**, **011LL**, **0x2FLL**
  - **u** lub **U** - typ **unsigned**, np. **10u**, **10U**, **10U**, **10LLU**, **0x2FUll**



## Język C - Stałe liczbowe (rzeczywiste)

- Domyślny typ liczb rzeczywistych to **double**
- Format zapisu **stałych zmiennoprzecinkowych** (ang. floating-point)

-2.41e+15	-2.41e+15	+4.123E-3	+4.123E-3
-----------	-----------	-----------	-----------

znak plus/minus	mantysa (ciąg cyfr z kropką dziesiętną)	e lub E	wykładnik ze znakiem
-----------------	---	---------	----------------------

- W zapisie można pominąć:
  - znak plus, np. **-2.41e15**, **4.123E-3**
  - kropkę dziesiętną lub część wykładniczą, np. **2e-5**, **14.15**
  - część ułamkową lub część całkowitą, np. **2.e-5**, **.12e4**

## Język C - Stałe liczbowe (rzeczywiste)

- W środku stałej zmiennoprzecinkowej nie mogą występować spacje
- Błędnie zapisane stałe zmiennoprzecinkowe:

- 2.41e+15

-2.41 e+15

-2.41e +15

- Przyrostki na końcu liczby zmieniają typ:
  - l lub L - typ **long double**, np. 2.5L, 1.24e7l
  - f lub F - typ **float**, np. 3.14f, 1.24e7F

## 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;
}
```

Zaczynamy!!!  
Pole = 7.065  
Obwod = 9.42

# Język C - Operatory

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



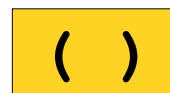
- Operator dwuargumentowy



- 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:

```
;
```

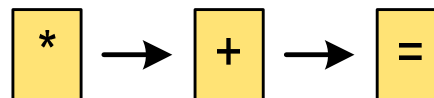
## Język C - Wyrażenia arytmetyczne

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

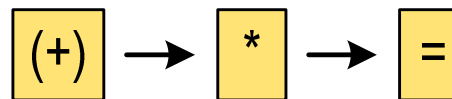
```
w = a + b;
```



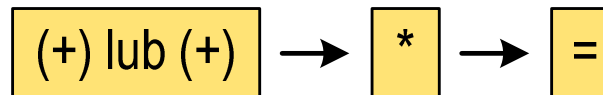
```
w = a + b * c;
```



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



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



## Język C - Wyrażenia arytmetyczne

- Kolejność wykonywania operacji

$$\boxed{w = a + b + c;} \rightarrow \boxed{w = ((a + b) + c);}$$

$$\boxed{w = x = y = a + b;} \rightarrow \boxed{w = (x = (y = (a + b)))};$$

- Zapis wyrażeń arytmetycznych

$$w = \frac{a + b}{c + d}$$

$$\boxed{w = a + b / c + d;}$$

ŹLE

$$\boxed{w = (a + b) / (c + d);}$$

DOBRCZE

$$w = \frac{a + b}{c \cdot d}$$

$$\boxed{w = (a + b) / c * d;}$$

ŹLE

$$\boxed{w = (a + b) / (c * d);}$$

DOBRCZE

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

- Większość funkcji ma 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 R, L, C, fr;

    printf("Podaj R [Om]: "); scanf("%lf", &R);
    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 R [Om]: 100
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");
```

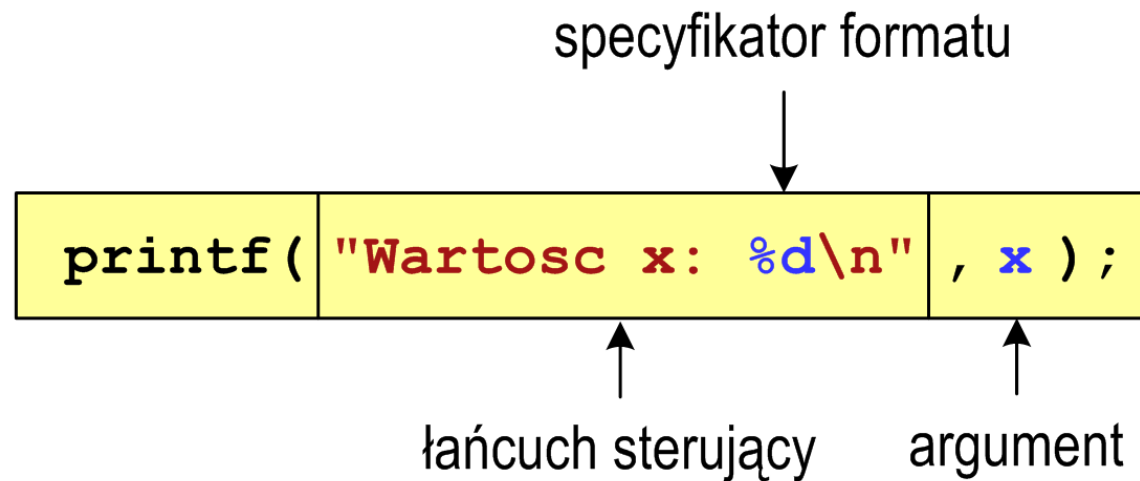
```
Witaj świecie
```

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

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

## Język C - Funkcja printf

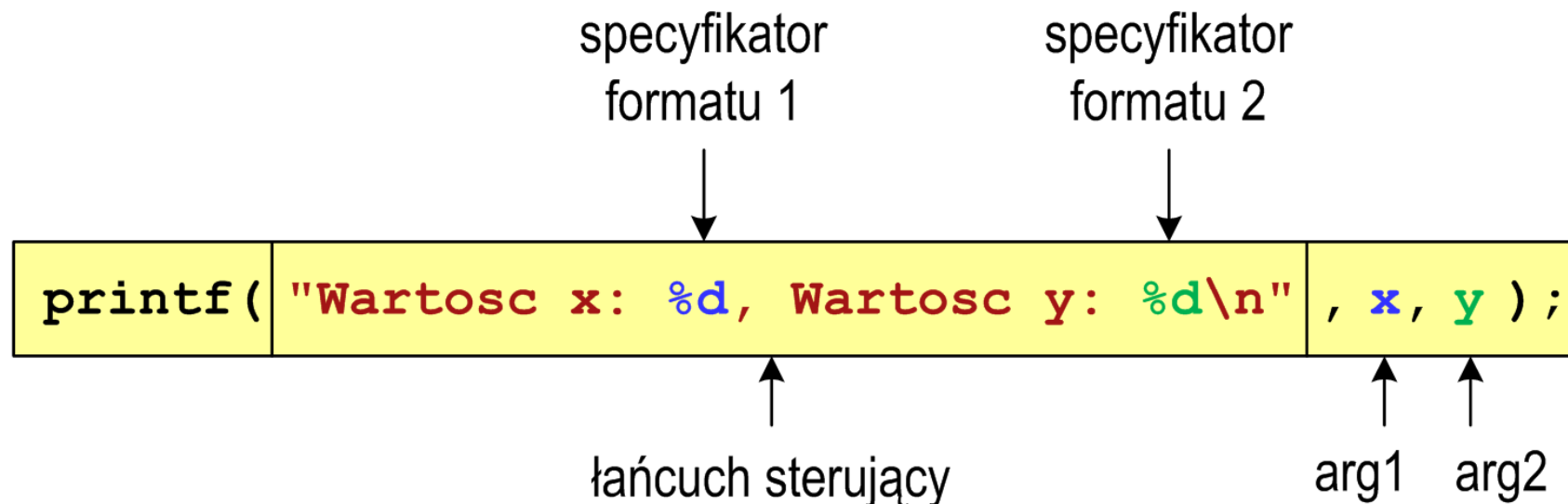
```
int x = 10;  
printf("Wartosc x: %d\n", x);
```



```
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 = [1.123457]
```

```
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 = [ 1.123457]
```

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

```
x = [ 123], y = [ 1.123]
```

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

```
x = [ 123], y = [1.123]
```



## Język C - Funkcja printf

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

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

```
x = [ +123], y = [ +1.123457]
```

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

```
x = [123   ], y = [1.123457   ]
```

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

```
x = [000123], y = [00001.123457]
```

## 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 = [28.648149]
```

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

```
x = [123], y = [2.119865]
```

## 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>

## 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 )
```

wynik: 0 (fałsz)

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

- 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) (!!!)

## Język C - Wyrażenia logiczne

- W przypadku sprawdzania czy wartość wyrażenia jest równa lub różna od zera można zastosować skrócony zapis
- Zamiast:

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

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

można napisać:

```
if ( !x )
```

```
if ( x )
```

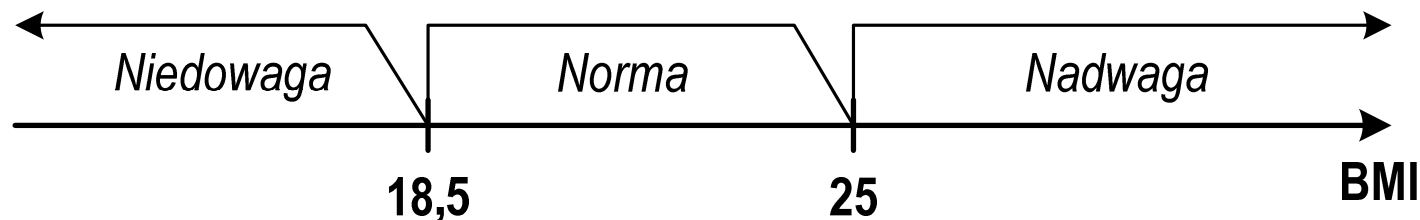


## Przykład: obliczanie BMI (Body Mass Index)

- **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");
```

Koniec wykładu nr 2

Dziękuję za uwagę!