



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 2022**

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

- Język C
  - funkcje matematyczne (`math.h`)
  - funkcje `printf` i `scanf`
  - instrukcja `if`, operatory relacyjne i logiczne, wyrażenia logiczne
  - operator warunkowy
  - instrukcja `switch`
  - pętla `for`
  - operatory `++` i `--`
  - pętle `while` i `do...while`

## 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>	<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

- 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");
```

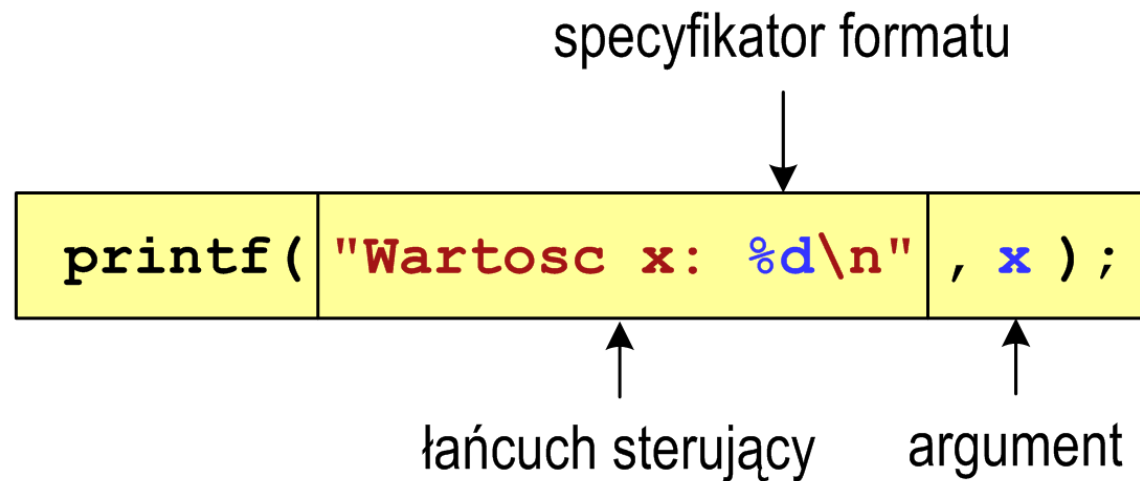
```
Witaj świecie
```

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

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

## Język C - Funkcja printf

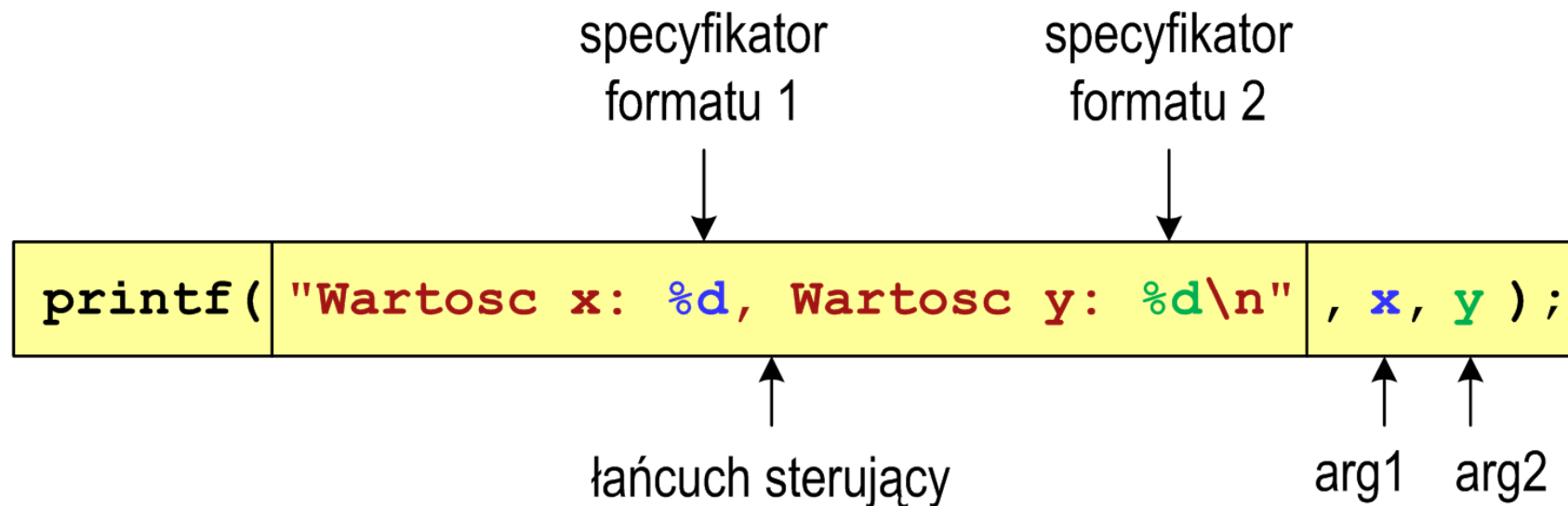
```
int x = 10;  
printf("Wartosc x: %d\n", x);
```



```
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>
```



## 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 <b>0</b> , a argument równy zero na wartość <b>1</b>
&&	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 )
```

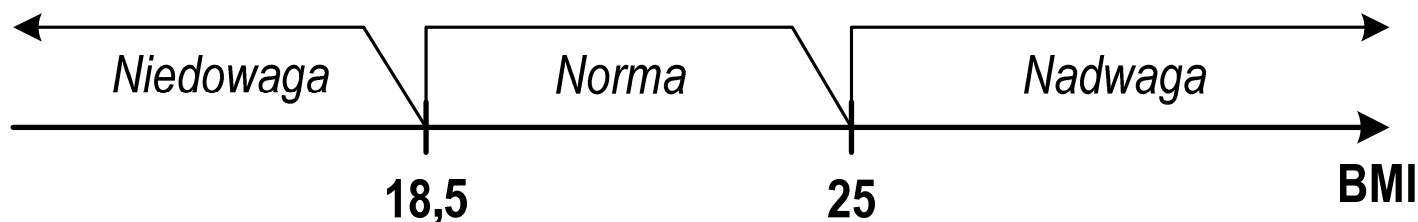
wynik: 1 (prawda) (!!!)

## 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>

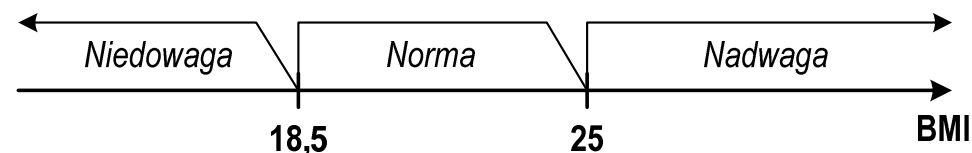
int main(void)
{
    double masa, wzrost, bmi;

    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

## Język C - Instrukcja switch

- Instrukcja wyboru wielowariantowego **switch**

```
switch (wyrażenie)
{
    case wyrażenie Stałe: instrukcje;
    case wyrażenie Stałe: instrukcje;
    case wyrażenie Stałe: instrukcje;
    ...
    default: instrukcje;
}
```

- **wyrażenie Stałe** - wartość typu całkowitego, znana podczas kompilacji
  - stała liczbowa, np. 3, 5, 9
  - znak w apostrofach, np. 'a', 'z', '+'
  - stała zdefiniowana przez **const** lub **#define**

## Język C - Instrukcja switch

- Program wyświetlający słownie liczbę z zakresu 1..5 wprowadzoną z klawiatury

```
#include <stdio.h>

int main(void)
{
    int liczba;

    printf("Podaj liczbę (1..5): ");
    scanf("%d", &liczba);
```

## Język C - Instrukcja switch

```
switch (liczba)
{
    case 1: printf("Liczba: jeden\n");
            break;
    case 2: printf("Liczba: dwa\n");
            break;
    case 3: printf("Liczba: trzy\n");
            break;
    case 4: printf("Liczba: cztery\n");
            break;
    case 5: printf("Liczba: piec\n");
            break;
    default: printf("Inna liczba\n");
}
}
```

Podaj liczbe: 2  
Liczba: dwa

Podaj liczbe: 0  
Inna liczba

## Język C - Instrukcja switch

```
switch (liczba)
{
    case 1:
    case 3:
    case 5: printf("Liczba nieparzysta\n");
           break;
    case 2:
    case 4: printf("Liczba parzysta\n");
           break;
    default: printf("Inna liczba\n");
}
```

Podaj liczbe: 2  
Liczba parzysta

- Te same instrukcje mogą być wykonane dla kilku etykiet **case**

## Język C - Instrukcja switch

```
switch (liczba)
{
    case 1: case 3: case 5:
        printf("Liczba nieparzysta\n");
        break;
    case 2: case 4:
        printf("Liczba parzysta\n");
        break;
    default: printf("Inna liczba\n");
}
```

Podaj liczbe: 2  
Liczba parzysta

- Etykiety **case** mogą być pisane w jednym wierszu

## Język C - Instrukcja switch

```
switch (liczba%2)
{
    case 1: case -1:
        printf("Liczba nieparzysta\n");
        break;
    case 0:
        printf("Liczba parzysta\n");
}
```

Podaj liczbe: 2  
Liczba parzysta

- Część domyślna (**default**) może być pominięta

## Język C - Instrukcja switch (bez break)

```
switch (liczba)
{
    case 1: printf("Liczba: jeden\n");
    case 2: printf("Liczba: dwa\n");
    case 3: printf("Liczba: trzy\n");
    case 4: printf("Liczba: cztery\n");
    case 5: printf("Liczba: piec\n");
    default: printf("Inna liczba\n");
}
```

```
Podaj liczbe: 2
Liczba: dwa
Liczba: trzy
Liczba: cztery
Liczba: piec
Inna liczba
```

- Pominięcie instrukcji **break** spowoduje wykonanie wszystkich instrukcji występujących po danym **case** (do końca **switch**)



## Przykład: suma kolejnych 10 liczb: $1+2+\dots+10$

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

```
int main(void)
```

```
{
```

```
    int suma;
```

```
    suma = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;
```

```
    printf("Suma wynosi: %d\n", suma);
```

```
    return 0;
```

```
}
```

Suma wynosi: 55

## Przykład: suma kolejnych 100 liczb: $1+2+\dots+100$

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

```
int main(void)
```

```
{
```

```
    int suma=0, i;
```

```
    for (i=1; i<=100; i=i+1)
```

```
        suma = suma + i;
```

```
    printf("Suma wynosi: %d\n", suma);
```

```
    return 0;
```

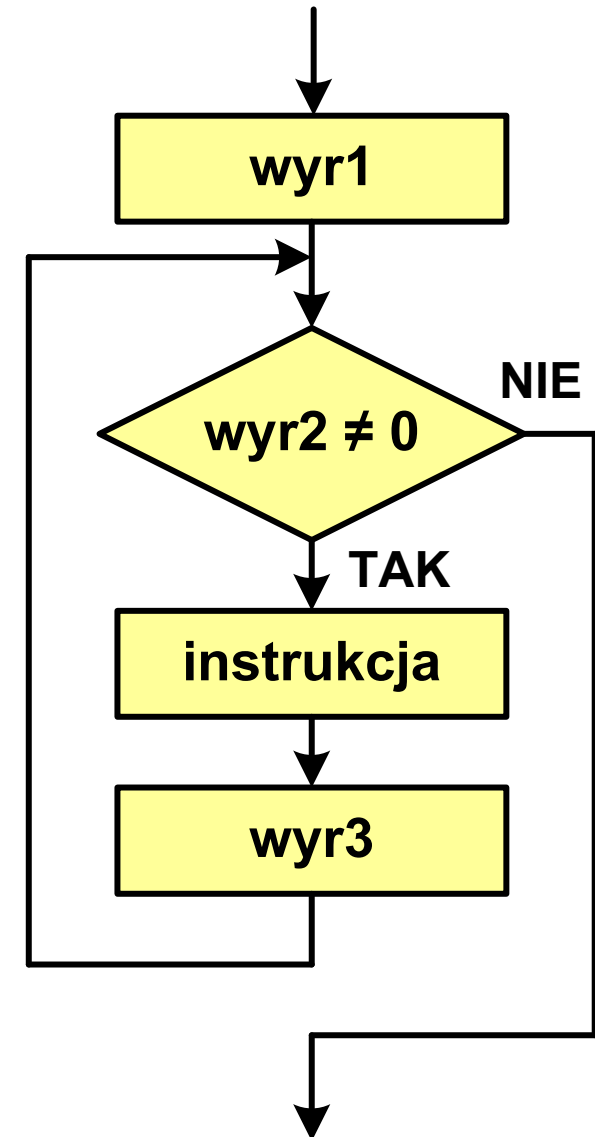
```
}
```

Suma wynosi: 5050

## Język C - pętla for

```
for (wyr1; wyr2; wyr3)  
instrukcja
```

- **wyr1, wyr2, wyr3** - dowolne wyrażenia w języku C
- Instrukcja:
  - **prosta** - jedna instrukcja zakończona średnikiem
  - **złożona** - jedna lub kilka instrukcji objętych nawiasami klamrowymi



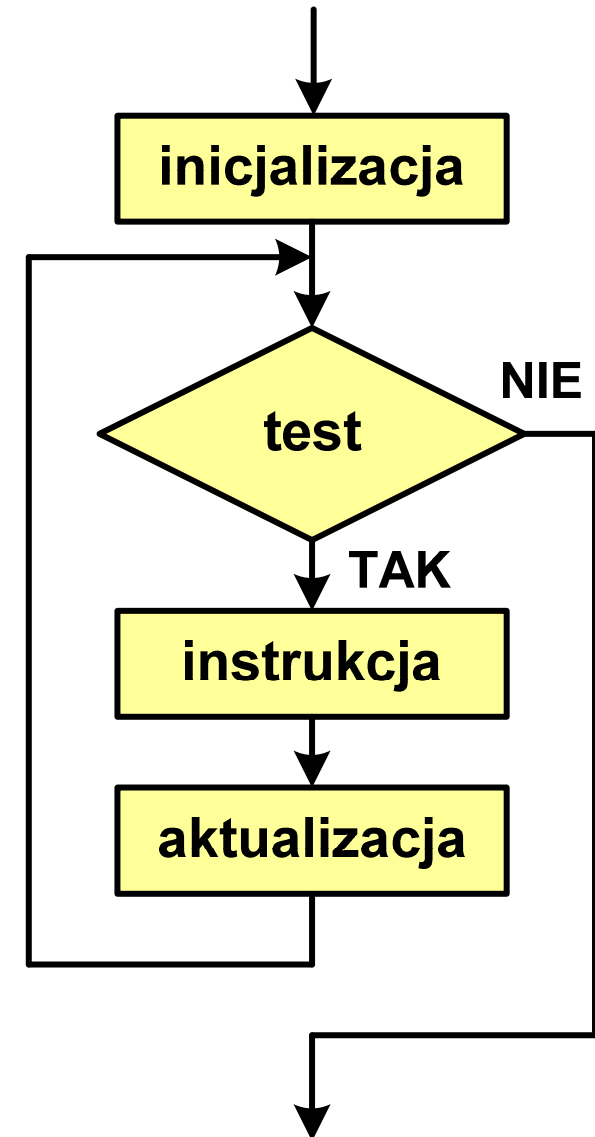
## Język C - pętla for

- Najczęściej stosowana postać pętli **for**

```
int i;  
for (i = 0; i < 10; i = i + 1)  
    instrukcja;
```

- Instrukcja zostanie wykonana 10 razy  
(dla  $i = 0, 1, 2, \dots, 9$ )
- Funkcje pełnione przez wyrażenia

```
for (inicjalizacja; test; aktualizacja)  
    instrukcja;
```



## Przykład: wyświetlenie tekstu 5 razy

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

```
int main(void)
```

```
{
```

```
    int i;
```

```
    for (i=0; i<5; i=i+1)
```

```
        printf("Programowanie nie jest trudne\n");
```

```
    return 0;
```

```
}
```

```
Programowanie nie jest trudne  
Programowanie nie jest trudne  
Programowanie nie jest trudne  
Programowanie nie jest trudne  
Programowanie nie jest trudne
```

## Przykład - suma liczb: $1 + 2 + \dots + N$

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

```
#define N 1234
```

```
int main(void)
```

```
{
```

```
    int i, suma=0;
```

```
    for (i=1; i<=N; i++)
```

```
        suma = suma + i;
```

```
    printf("Suma %d liczb to %d\n", N, suma);
```

```
    return 0;
```

```
}
```

Suma 1234 liczb to 761995

## Język C - pętla for (przykłady)

```
for (i=0; i<10; i++)  
    printf("%d ", i);
```

0 1 2 3 4 5 6 7 8 9

```
for (i=0; i<10; i++)  
    printf("%d ", i+1);
```

1 2 3 4 5 6 7 8 9 10

```
for (i=1; i<=10; i++)  
    printf("%d ", i);
```

1 2 3 4 5 6 7 8 9 10

## Język C - pętla for (przykłady)

```
for (i=1; i<10; i=i+2)  
    printf("%d ", i);
```

1 3 5 7 9

```
for (i=10; i>0; i--)  
    printf("%d ", i);
```

10 9 8 7 6 5 4 3 2 1

```
for (i=-9; i<=9; i=i+3)  
    printf("%d ", i);
```

-9 -6 -3 0 3 6 9



## Język C - pętla for (break, continue)

- W pętli **for** można stosować instrukcje skoku: **break** i **continue**

```
int i;
for (i=1; i<10; i++)
{
    if (i%2==0)
        continue;
    if (i%7==0)
        break;
    printf("%d\n", i);
}
```

1 3 5

- **continue** przerywa bieżącą iterację i przechodzi do obliczania **wyr3**
- **break** przerywa wykonywanie pętli

## Język C - pętla for (najczęstsze błędy)

- Postawienie średnika na końcu pętli **for**

```
int i;  
for (i=0; i<10; i++);  
printf("%d ", i);
```

10

- Przecinki zamiast średników pomiędzy wyrażeniami

```
int i;  
for (i=0, i<10, i++)  
    printf("%d ", i);
```

*Błąd kompilacji!*

## Język C - pętla for (najczęstsze błędy)

- Błędny warunek - brak wykonania instrukcji

```
int i;  
for (i=0; i>10; i++)  
    printf("%d ", i);
```



- Błędny warunek - pętla nieskończona

```
int i;  
for (i=1; i>0; i++)  
    printf("%d ", i);
```

1 2 3 4 5 6 7 8 9 ...

## Język C - pętla nieskończona

```
for (wyr1; wyr2; wyr3)  
    instrukcja;
```

- Wszystkie wyrażenia (**wyr1**, **wyr2**, **wyr3**) w pętli for są opcjonalne

```
for ( ; ; )  
    instrukcja;
```

- pętla nieskończona

- W przypadku braku **wyr2** przyjmuje się, że jest ono **prawdziwe**

## Język C - zagnieżdżanie pętli for

- Jako instrukcja w pętli **for** może występować kolejna pętla **for**

```
int i, j;
for (i=1; i<=3; i++)           // pętla zewnętrzna
    for (j=1; j<=2; j++)       // pętla wewnętrzna
        printf("i: %d    j: %d\n", i, j);
```

```
i: 1    j: 1
i: 1    j: 2
i: 2    j: 1
i: 2    j: 2
i: 3    j: 1
i: 3    j: 2
```

## Język C - operator inkrementacji (++)

- Jednoargumentowy operator **++** zwiększa wartość zmiennej o 1 (nie wolno stosować go do wyrażeń)
- Operator **++** może występować jako przedrostek lub przyrostek

Zapis	Nazwa	Znaczenie
<b>++x</b>	preinkrementacji	wartość zmiennej jest modyfikowana przed jej użyciem
<b>x++</b>	postinkrementacji	wartość zmiennej jest modyfikowana po użyciu jej poprzedniej wartości

# Język C - operator inkrementacji (++)

## ■ Przykład

```
int x = 1, y;  
y = 2 * ++x;
```

```
int x = 1, y;  
y = 2 * x++;
```

## ■ Kolejność operacji

```
++x           x = 2  
2 * ++x      2 * 2  
y = 2 * ++x  y = 4
```

```
2 * x         2 * 1  
y = 2 * x     y = 2  
x++           x = 2
```

## ■ Wartości zmiennych

```
x = 2    y = 4
```

```
x = 2    y = 2
```

## Język C - operator inkrementacji (++)

- Miejsce umieszczenia operatora ++ nie ma znaczenia w przypadku instrukcji typu:

```
x++;  
++x;
```

równoważne

```
x = x + 1;
```

- Nie należy stosować operatora ++ do zmiennych pojawiających się w wyrażeniu więcej niż jeden raz

```
x = x++;  
x = ++x;
```

- Zgodnie ze standardem języka C wynik powyższych instrukcji jest **niezdefiniowany**



## Język C - operator dekrementacji (--)

- Jednoargumentowy operator -- zmniejsza wartość zmiennej o 1 (nie wolno stosować go do wyrażeń)
- Operator -- może występować jako przedrostek lub przyrostek

Zapis	Nazwa	Znaczenie
-- <b>x</b>	predekrementacji	wartość zmiennej jest modyfikowana przed jej użyciem
<b>x</b> --	postdekrementacji	wartość zmiennej jest modyfikowana po użyciu jej poprzedniej wartości

## Język C - priorytet operatorów ++ i --

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

## Język C - miesięczny kalendarz

- Napisz program wyświetlający miesięczny kalendarz. Wczytaj liczbę dni w miesiącu i dzień tygodnia, od którego zaczyna się miesiąc.
- Przykład działania programu:

**Liczba dni w miesiącu: 31**

**Pierwszy dzień tygodnia (1-Pn, 2-Wt, ...): 4**

-----

Pn	Wt	Sr	Cz	Pt	So	N
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

# Język C - miesięczny kalendarz

```
#include <stdio.h>

int main()
{
    int ile_dni, dzien_tyg, i;

    printf("Liczba dni w miesiacu: "); scanf("%d",&ile_dni);
    printf("Pierwszy dzien tygodnia (1-Pn, 2-Wt, ...): ");
    scanf("%d",&dzien_tyg);

    printf("-----\n");
    printf(" Pn Wt Sr Cz Pt So  N\n");

    for (i=1; i<dzien_tyg; i++) printf("  ");
    for (i=0; i<ile_dni; i++)
    {
        printf("%3d",i+1);
        if ((i+dzien_tyg)%7==0) printf("\n");
    }
    printf("\n"); return 0;
}
```

## Język C - miesięczny kalendarz

```
#include <stdio.h>
int main()
{
    int ile_dni;
    int dzien_tyg;

    printf("Liczba dni w miesiącu: ");
    printf("Pierwszy dzień tygodnia (1-Pn, 2-Wt, ...): ");
    scanf("%d %d", &ile_dni, &dzien_tyg);

    printf("-----\n");
    printf("Pn Wt Sr Cz Pt So N\n");

    for (i=1; i<dzien_tyg; i++) printf(" ");
    for (i=0; i<ile_dni; i++)
    {
        printf("%3d", i+1);
        if ((i+dzien_tyg)%7==0) printf("\n");
    }
    printf("\n"); return 0;
}
```

```
Liczba dni w miesiacu: 30
Pierwszy dzien tygodnia (1-Pn, 2-Wt, ...): 5
-----
Pn Wt Sr Cz Pt So N
    1  2  3
  4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

## 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: -3  
Blad! Liczba ujemna

Podaj liczbe: 3  
Pierwiastek liczby: 1.732051

## Przykład: pierwiastek kwadratowy (pętla while)

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

int main(void)
{
    float x, y;

    printf("Podaj liczbe: ");
    scanf("%f", &x);
    while (x<0)
    {
        printf("Blad! Liczba ujemna\n\n");
        printf("Podaj liczbe: ");
        scanf("%f", &x);
    }
    y = sqrt(x);
    printf("Pierwiastek liczby: %f\n", y);

    return 0;
}
```

```
Podaj liczbe: -3
Blad! Liczba ujemna
```

```
Podaj liczbe: -5
Blad! Liczba ujemna
```

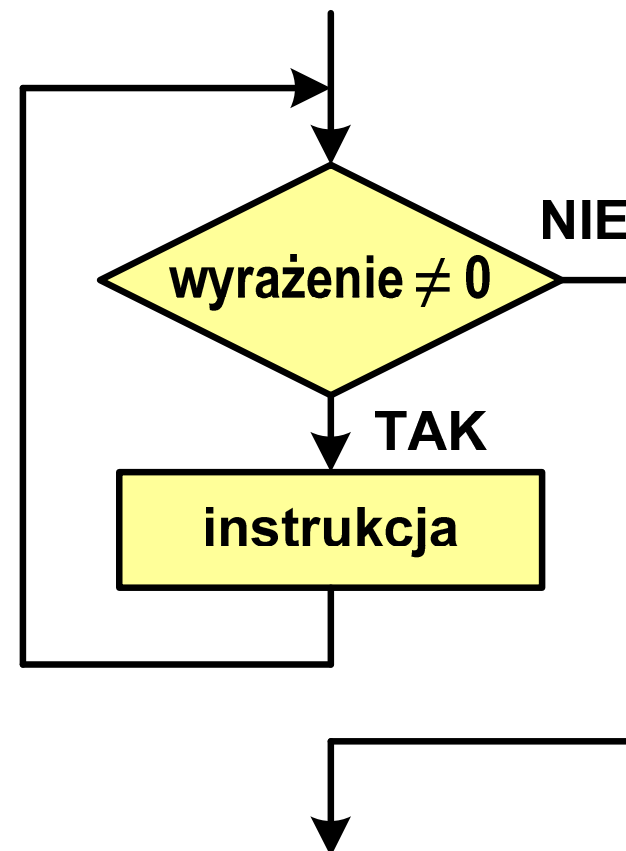
```
Podaj liczbe: 3
Pierwiastek liczby: 1.732051
```

## Język C - pętla while

```
while (wyrażenie)  
    instrukcja;
```

- „dopóki wyrażenie w nawiasach jest prawdziwe wykonuj instrukcję”

- Wyrażenie w nawiasach:
  - **prawdziwe** - gdy jego wartość jest różna od zera
  - **fałszywe** - gdy jego wartość jest równa zero
- Jako wyrażenie najczęściej stosowane jest **wyrażenie logiczne**





## Język C - pętla while

```
while (wyrażenie)
    instrukcja;
```

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

```
int x = 10;
while (x>0)
    x = x - 1;
```

```
int x = 10;
while (x>0)
{
    printf("%d\n", x);
    x = x - 1;
}
```

## Przykład: suma liczb dodatnich

```
#include <stdio.h>

int main(void)
{
    int x, suma = 0;

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

    while (x>0)
    {
        suma = suma + x;
        printf("Podaj liczbe: ");
        scanf("%d", &x);
    }
    printf("Suma liczb: %d\n", suma);

    return 0;
}
```

```
Podaj liczbe: 4
Podaj liczbe: 8
Podaj liczbe: 2
Podaj liczbe: 3
Podaj liczbe: 5
Podaj liczbe: -2
Suma liczb: 22
```

## Język C - pętla while

- Program pokazany na poprzednim slajdzie zawiera typowy schemat przetwarzania danych z wykorzystaniem pętli **while**

```
printf("Podaj liczbę: ");  
scanf("%d", &x);
```

wczytanie danych

```
while (x>0)
```

```
{
```

```
    suma = suma + x;
```

operacje na danych

```
    printf("Podaj liczbę: ");  
    scanf("%d", &x);
```

wczytanie danych

```
}
```

- Dane mogą być wczytywane z klawiatury, pliku, itp.

## Język C - pętla while (break, continue)

- **break** i **continue** są to instrukcje skoku

```
int x=0;
while (x<10)
{
    x++;
    if (x%2==0)
        continue;
    if (x%5==0)
        break;
    printf ("%d\n", x);
}
```

- **continue** przerywa bieżącą iterację
- **break** przerywa wykonywanie pętli

## Język C - pętla while (najczęstsze błędy)

- Postawienie średnika po wyrażeniu w nawiasach powoduje powstanie pętli nieskończonej - program zatrzymuje się na pętli

```
int x = 10;  
while (x>0);  
    printf("%d ", x--);
```



- Brak aktualizacji zmiennej powoduje także powstanie pętli nieskończonej - program wyświetla wielokrotnie tę samą wartość

```
int x = 10;  
while (x>0)  
    printf("%d ", x);
```

10 10 10 10 10 ...

## Język C - pętla while (pętla nieskończona)

- W pewnych sytuacjach celowo stosuje się pętlę nieskończoną (np. w mikrokontrolerach)

```
while (1)
{
    instrukcja;
    instrukcja;
    ...
}
```

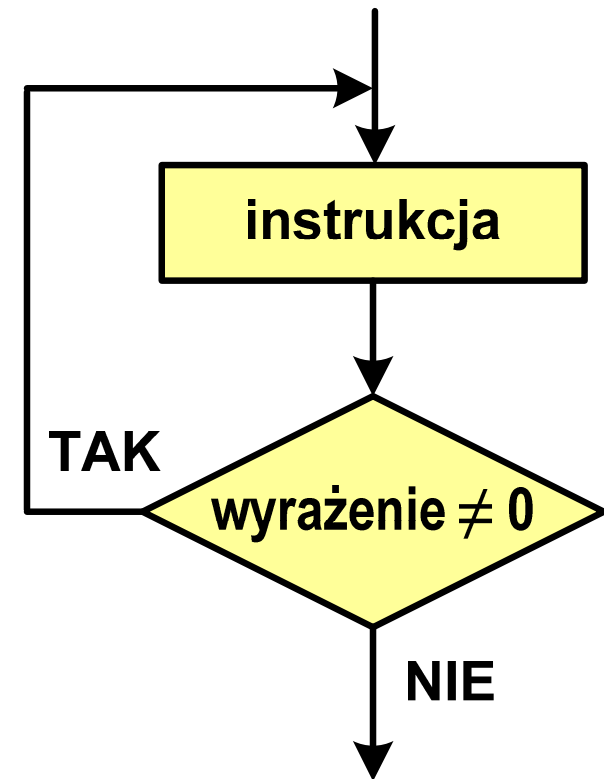
- W układach mikroprocesorowych program działa aż do wyłączenia zasilania

## Język C - pętla do ... while

```
do  
    instrukcja;  
while (wyrażenie);
```

- „wykonuj instrukcję dopóki wyrażenie w nawiasach jest prawdziwe”

- Wyrażenie w nawiasach:
  - **prawdziwe** - gdy jego wartość jest różna od zera
  - **fałszywe** - gdy jego wartość jest równa zero



## Język C - pętla do ... while

```
do
    instrukcja;
while (wyrażenie);
```

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

```
int x = 10;
do
    x = x - 1;
while (x>0);
```

```
int x = 10;
do
{
    printf("%d\n", x);
    x = x - 1;
}
while (x>0);
```



## Język C - pętla do ... while (break, continue)

- **break** i **continue** są to instrukcje skoku

```
int x=0;

do
{
    x++;
    if (x%5==0)
        break;
    if (x%2==0)
        continue;
    printf ("%d\n", x);
}
while (i<10);
```

- **break** przerywa wykonywanie pętli
- **continue** przerywa bieżącą iterację

## Przykład: suma liczb < 100

```
#include <stdio.h>

int main(void)
{
    int x, suma = 0;

    do
    {
        printf("Podaj liczbe: ");
        scanf("%d", &x);
        suma = suma + x;
    }
    while (suma < 100);

    printf("Suma liczb: %d\n", suma);

    return 0;
}
```

```
Podaj liczbe: 34
Podaj liczbe: 9
Podaj liczbe: 26
Podaj liczbe: -8
Podaj liczbe: 67
Suma liczb: 128
```

Koniec wykładu nr 2

Dziękuję za uwagę!