Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Materiały do wykładu z przedmiotu:
**Informatyka**
**Kod: EDS1B1007**

# WYKŁAD NR 2

**Opracował: dr inż. Jarosław Forenc**

**Białystok 2020**

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

- Stałe liczbowe, deklaracje zmiennych i stałych
- Operatory, priorytet operatorów
- Wyrażenia, instrukcje
- Wyrażenia arytmetyczne
- Funkcje printf i scanf
- Instrukcja warunkowa if
- Operator warunkowy
- Instrukcja switch

# 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, 10lU, 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 dzisiętną) | 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 ustalane przed uruchomieniem programu i pozostają niezmienione 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_stałej wartość_stałej

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

- Wyrażenia stałe zazwyczaj pisze się wielkimi literami
- Wyrażenia stałe są obliczane przed właściwą kompilacją programu
- W kodzie programu w miejscu występowania stałej wstawiana jest jej wartość

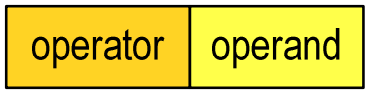
# Język C - stałe symboliczne (#define)

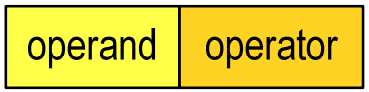
```c
#include <stdio.h>
#define PI 3.14
#define KOMUNIKAT "Zaczynamy!!!\n"

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



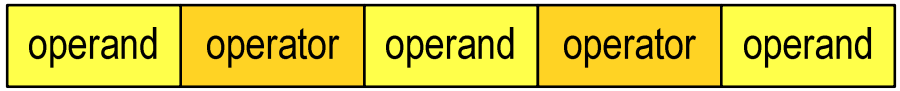

```
-x
```

```
x++
```

- Operator dwuargumentowy


```
x * y
```

- Operator trójargumentowy




```
x > y ? x : y
```

- Operator wieloargumentowy

```
( )
```

# Język C - operatory

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

# Język C - priorytet operatorów (1/2)

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

# Język C - priorytet operatorów (2/2)

| Priorytet | Operator / opis |
|---|---|
| 10 | \| |
| 11 | && |
| 12 | \|\| |
| 13 | ? : |
| 14 | = <br> += -= *= /= %= <<= >>= &= \|= ^= |
| 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 | *typ x* | 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 swiecie\n");
```

  - strukturalna:

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

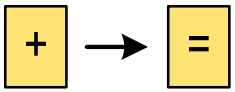
  - pusta:

```
;
```

# Język C - wyrażenia arytmetyczne

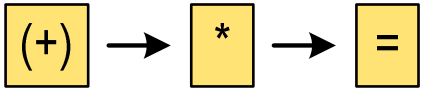
- Wyrażenia arytmetyczne mogą zawierać:
  - stałe liczbowe, zmienne, stałe
  - operatory: `+ - * / % = ( )` i inne
  - wywołania funkcji (plik nagłówkowy math.h)
- Kolejność wykonywania operacji wynika z priorytetu operatorów

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

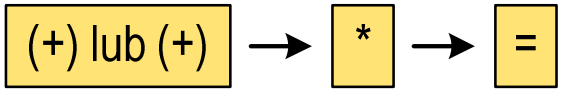



```
w = a + b * c;
```


```
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 wyrażeń 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 |
|---|---|---|
| **M_PI** | 3.14159265358979323846 | liczba pi |
| **M_E** | 2.71828182845904523536 | *e* - liczba Eulera |
| **M_LN2** | 0.69314718055994530941 7 | ln 2 |
| **M_SQRT2** | 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 |
|---|---|---|
| `abs` | **int abs(int x);** | moduł x (x - całkowite) |
| `fabs` | **double fabs(double x);** | moduł x (x - rzeczywiste) |
| `sqrt` | **double sqrt(double x);** | pierwiastek kwadratowy x |
| `pow` | **double pow(double x, double y);** | $x^y$ - x do potęgi y |
| `sin` | **double sin(double x);** | sinus argumentu x w radianach |
| `atan` | **double atan(double x);** | arcus tangens argumentu x |
| `atan2` | **double atan2(double y, double x);** | 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 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 swiecie");
```

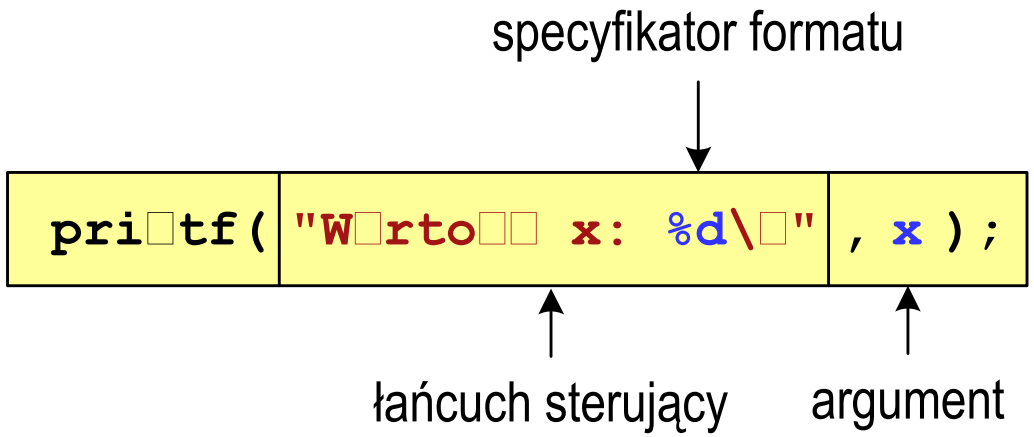
```
Witaj swiecie
```

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

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

# Język C - funkcja printf

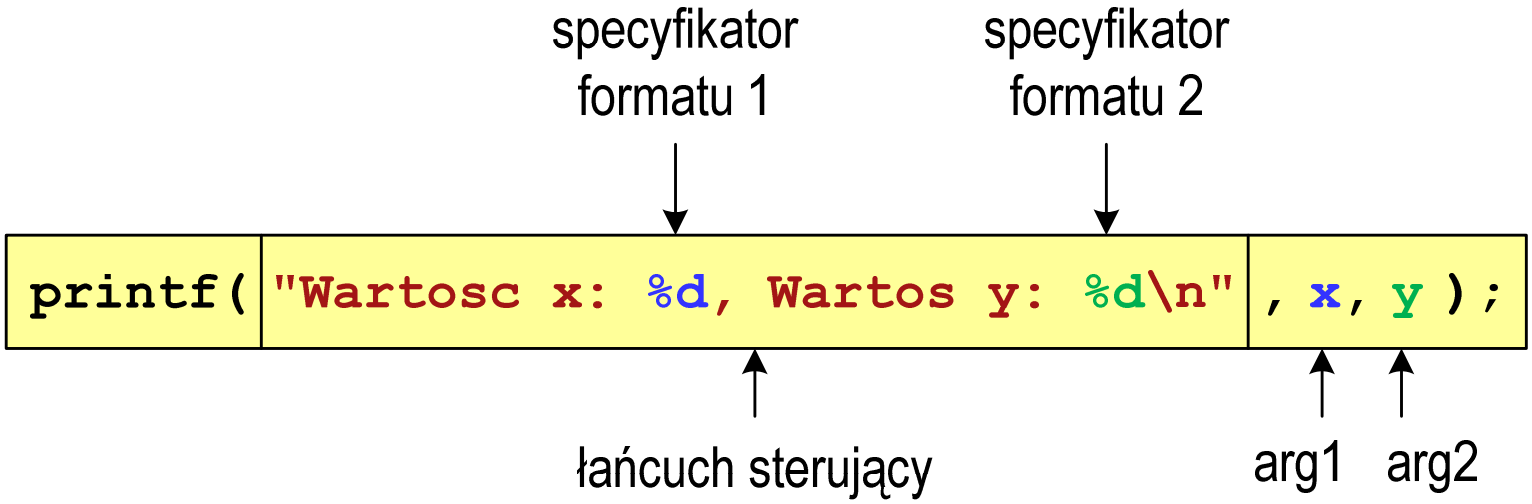
```
int x = 10;
printf("Wartosc x: %d\n", x);
```




```
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 = [1.123457]
```

```
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("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>
```

# Język C - pierwiastek kwadratowy

```c
#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
```

# Język C - pierwiastek kwadratowy

```c
#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;
    instr;
}
```

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

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

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

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

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

# Język C - operatory relacyjne (porównania)

| Operator | Przykład | Znaczenie |
|---|---|---|
| `>` | `a > b` | a większe od b |
| `<` | `a < b` | a mniejsze od b |
| `>=` | `a >= b` | a większe lub równe b |
| `<=` | `a <= b` | a mniejsze lub równe b |
| `==` | `a == b` | a równe b |
| `!=` | `a != b` | a nierówne b (a różne od b) |

- 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 )
```

```
if ( z > (x + y) )
```
wynik: 1 (prawda)

# 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) (!!!)

# 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)
```

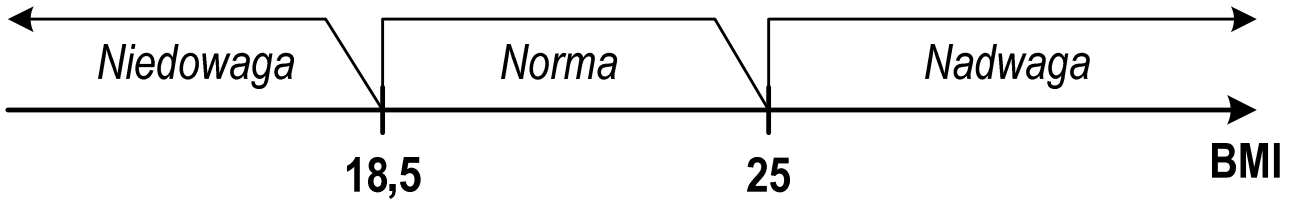
```
if (x)
```

# Język C - BMI

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

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

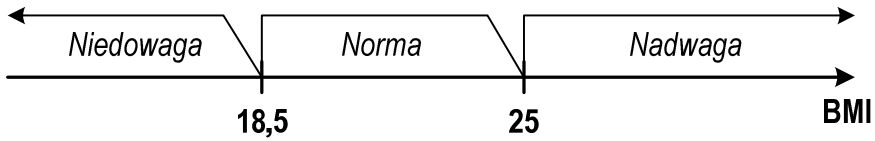
- Dla osób dorosłych:
  - BMI < 18,5 - wskazuje na niedowagę
  - BMI ≥ 18,5 i BMI < 25 - wskazuje na prawidłową masę ciała
  - BMI ≥ 25 - wskazuje na nadwagę



# Język C - BMI

```c
#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
```



# Język C - BMI

- 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 skład 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 (,)

# Język C - operator warunkowy

- Studenci chcą dojechać z akademika do biblioteki - 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 ? 1 : 0);

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

    return 0;
}
```

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

# 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 liczbe (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)

# Koniec wykładu nr 2

Dziękuję za uwagę!