

Informatyka 1

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr II, studia stacjonarne I stopnia
Rok akademicki 2017/2018

Wykład nr 3 (23.04.2018)

dr inż. Jarosław Forenc

Plan wykładu nr 3

- Język C
 - instrukcje warunkowa if, operator warunkowy
 - instrukcja switch
 - pętle: for, while, do...while
 - operatory ++ i --
- Kodowanie liczb
 - NKB, BCD, 2 z 5, kod Graya

Język C - 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
```

Język C - 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
>	$a > b$	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 )
```

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) (!!!)

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 - 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 (`,`)

Język C - Operator warunkowy

- x studentów chce dojechać z akademika do biblioteki - ile taksówek powinni zamówić? (jedna taksówka może przewieźć 4 osoby)

```
#include <stdio.h>

int main(void)
{
    int x, taxi;

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

    taxi = x / 4 + (x % 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 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**)

Język C - 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

Język C - 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);

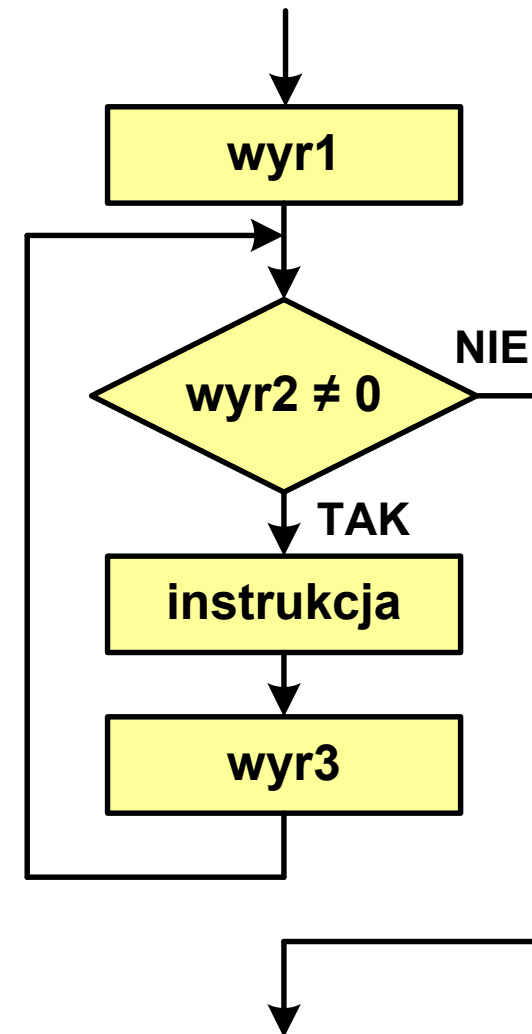
    return 0;
}
```

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



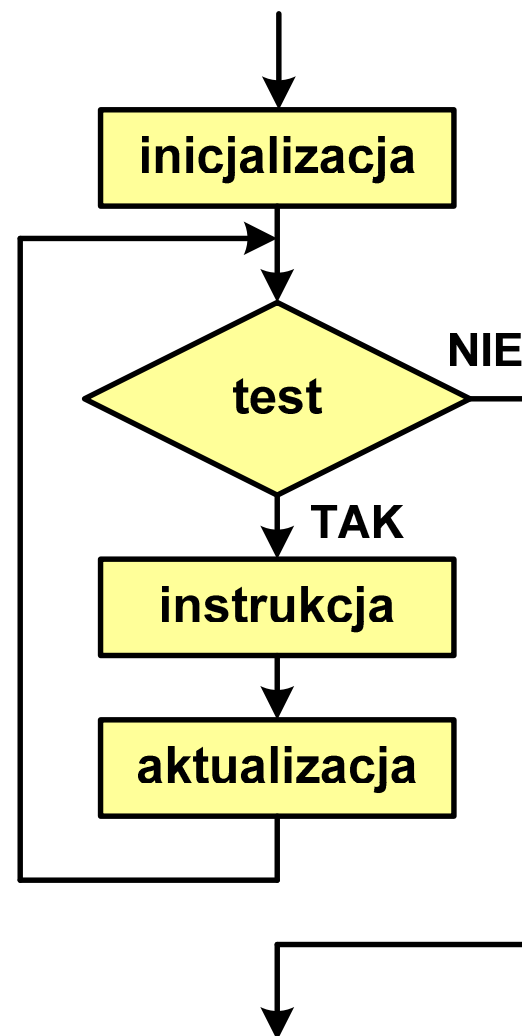
Język C - pętla for

- 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
```



Język C - pętla for (wyświetlenie tekstu)

```
#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
```

Język C - pętla for (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);
```

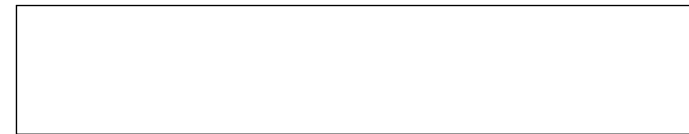
Błąd kompilacji!

error C2143: syntax error : missing ';' before ')'

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
++x	preinkrementacji	wartość zmiennej jest modyfikowana przed jej użyciem
x++	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
<code>--x</code>	predekrementacji	wartość zmiennej jest modyfikowana przed jej użyciem
<code>x--</code>	postdekrementacji	wartość zmiennej jest modyfikowana po użyciu jej poprzedniej wartości

Język C - priorytet operatorów ++ i --

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

Język C - 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

Język C - 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
```

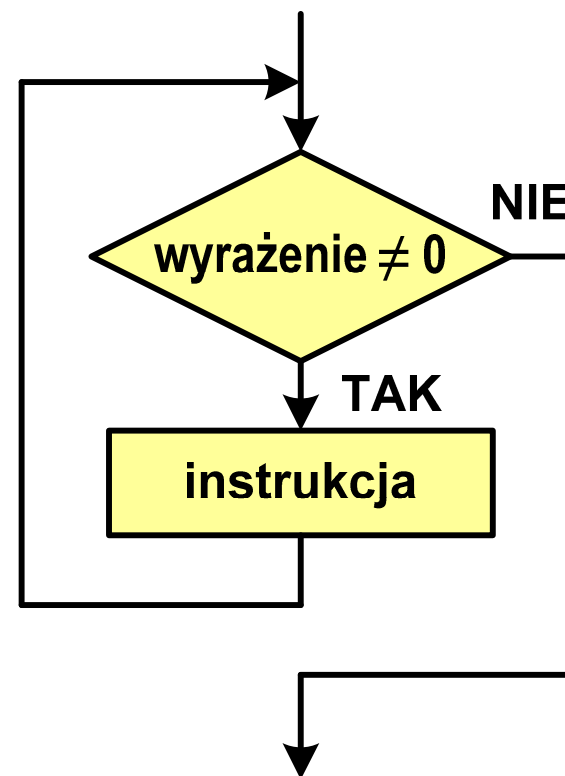
```
Podaj liczbe: 3
Pierwiastek liczby: 1.732051
```

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;  
}
```

Język C - suma liczb dodatnich

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

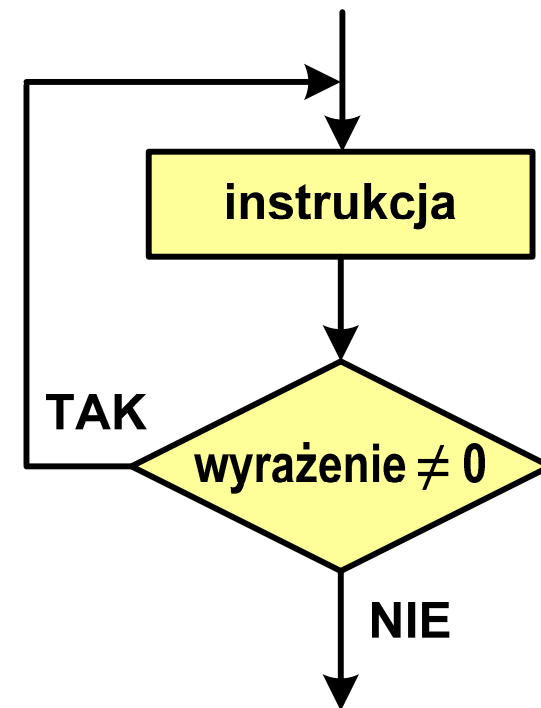
- W układach mikroprocesorowych program działa aż do wyłączenia zasilania

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

Język C - wartość liczby π

- Suma szeregu liczbowego

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots + (-1)^{n-1} \frac{1}{2n-1} = \frac{\pi}{4}$$

$$\pi = 4 \cdot \sum_{n=1}^{\infty} (-1)^{n-1} \frac{1}{2n-1}$$

- Sumy częściowe

$$s_1 = 1,$$

$$s_2 = 1 - \frac{1}{3},$$

$$s_3 = 1 - \frac{1}{3} + \frac{1}{5},$$

$$s_4 = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7},$$

...

$$s_n$$

- Warunek stopu

$$|\pi_n - \pi_{n-1}| < \varepsilon$$

ε - założona dokładność

Język C - wartość liczby π

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

int main(void)
{
    float s2 = 1, s1, eps = 0.01f;
    int n = 1, zn = 1;

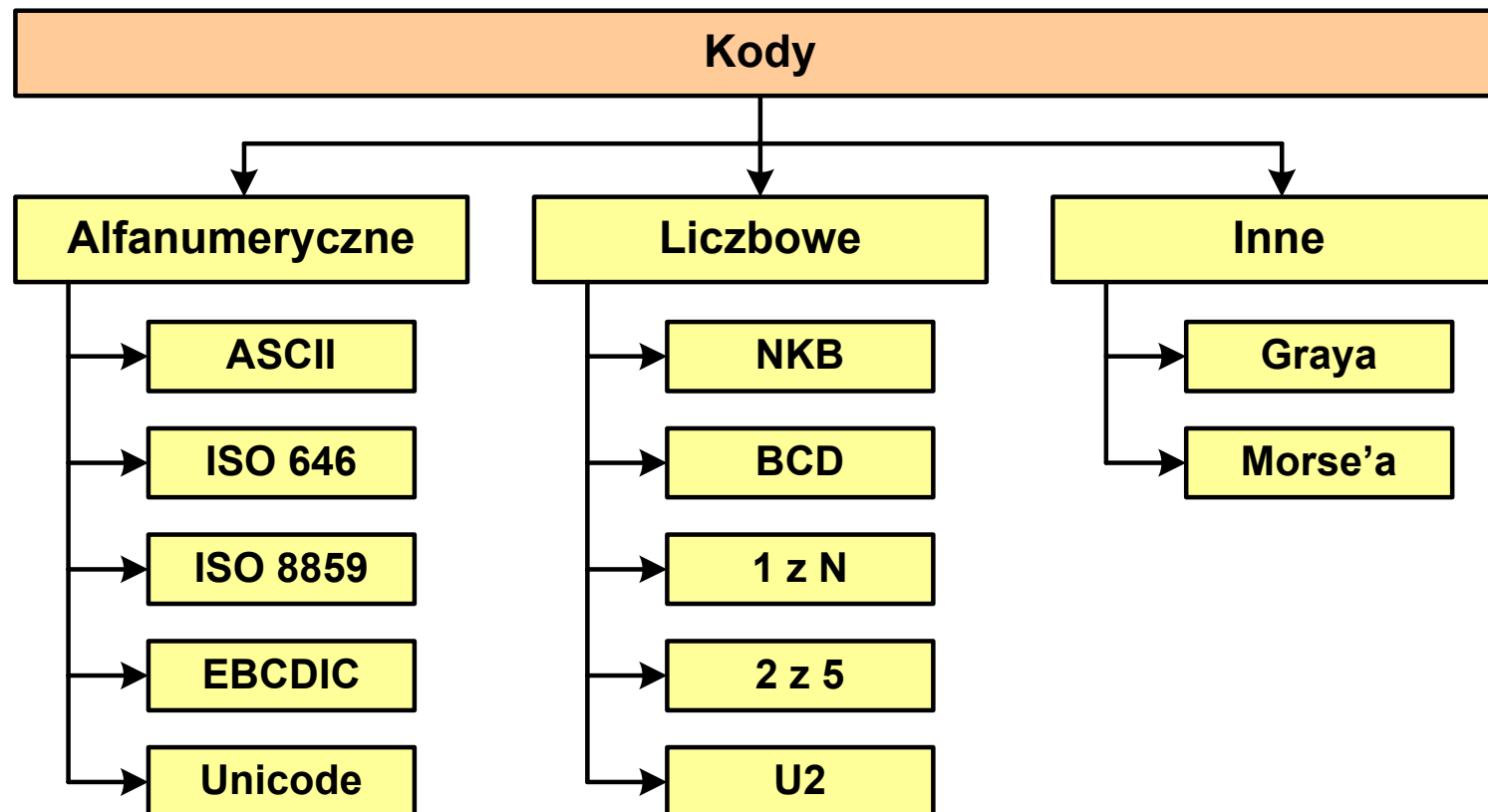
    do
    {
        s1 = s2; zn = -zn; n = n + 1;
        s2 = s2 + zn*1.0f/(2*n-1);
    }
    while (fabs(4*s2-4*s1)>=eps);

    printf("Pi[%d] = %f\n",n,4*s2);
    printf("Roznica = %f\n",M_PI-4*s2);
    return 0;
}
```

```
Pi[201] = 3.146568
Roznica = -0.004975
```

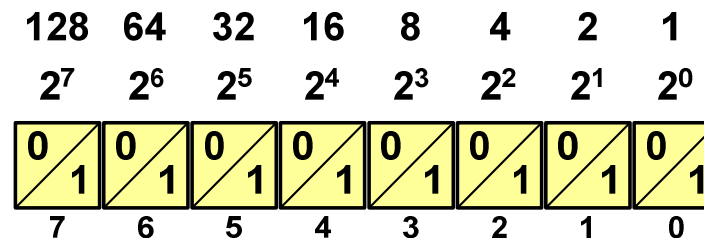
Kodowanie

- **Kodowanie** - proces przekształcania jednego rodzaju postaci informacji na inną postać



Kody liczbowe - Naturalny Kod Binarny (NKB)

- Jeżeli dowolnej liczbie dziesiętnej przypiszemy odpowiadającą jej liczbę binarną, to otrzymamy **naturalny kod binarny** (NKB)



Liczba dziesiętna	Kod NKB
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Liczba dziesiętna	Kod NKB
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Kody liczbowe - Kod BCD

- **B**inary-**C**oded **D**ecimal - dziesiętny zakodowany dwójkowo
- **BCD** - sposób zapisu liczb polegający na zakodowaniu kolejnych cyfr liczby dziesiętnej w 4-bitowym systemie dwójkowym (NKB)

Cyfra dziesiętna	BCD	Cyfra dziesiętna	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

- W ogólnym przypadku kodowane są tylko znaki **0 ÷ 9**
- Pozostałe kombinacje bitowe mogą być stosowane do kodowania znaku liczby lub innych znaczników.

Kody liczbowe - Kod BCD

■ Przykład:

$$168_{(10)} = ?_{(BCD)}$$

$$\begin{array}{ccc} \overbrace{0001}^1 & \overbrace{0110}^6 & \overbrace{1000}^8 \end{array}$$

$$168_{(10)} = 000101101000_{(BCD)}$$

$$1001 | 0101 | 0011_{(BCD)} = ?_{(10)}$$

$$\begin{array}{ccc} \underbrace{1001}_9 & \underbrace{0101}_5 & \underbrace{0011}_3 \end{array}$$

$$100101010011_{(BCD)} = 953_{(10)}$$

■ Zastosowania:

- urządzenia elektroniczne z wyświetlaczem cyfrowym (np. kalkulatory, mierniki cyfrowe, kasy sklepowe, wagi)
- przechowywania daty i czasu w BIOSie komputerów (także wczesne modele PlayStation 3)
- zapis części ułamkowych kwot (systemy bankowe).

Kody liczbowe - Kod BCD: przechowywanie liczb

- Użycie 4 najmłodszych bitów jednego bajta, 4 starsze bity są ustawiane na jakąś konkretną wartość:
 - 0000
 - 1111 (np. kod EBCDIC, liczby $F0_{(16)} \div F9_{(16)}$)
 - 0011 (tak jak w ASCII, liczby $30_{(16)} \div 39_{(16)}$)
- Zapis dwóch cyfr w każdym bajcie (starsza na starszej połówce, młodsza na młodszej połówce) - jest to tzw. **spakowane BCD**
 - w przypadku liczby zapisanej na kilku bajtach, najmniej znacząca tetrada (4 bity) używane są jako flaga znaku
 - standardowo przyjmuje się 1100 ($C_{(16)}$) dla znaku plus (+) i 1101 ($D_{(16)}$) dla znaku minus (-), np.

$$127_{(10)} = 0001\ 0010\ 0111\ \mathbf{1100} \quad (127C_{(16)})$$

$$-127_{(10)} = 0001\ 0010\ 0111\ \mathbf{1101} \quad (127D_{(16)})$$

Kody liczbowe - Kod BCD

- Warianty kodu BCD:

Cyfra dziesiętna	BCD 8421	Excess-3	BCD 2421	BCD 84-2-1	IBM 1401 BCD 8421
0	0000	0011	0000	0000	1010
1	0001	0100	0001	0111	0001
2	0010	0101	0010	0110	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0100	0100
5	0101	1000	1011	1011	0101
6	0110	1001	1100	1010	0110
7	0111	1010	1101	1001	0111
8	1000	1011	1110	1000	1000
9	1001	1100	1111	1111	1001

- Podstawowy wariant: **BCD 8421** (SBCD - Simple Binary Coded Decimal)

Kody liczbowe - Kod 2 z 5

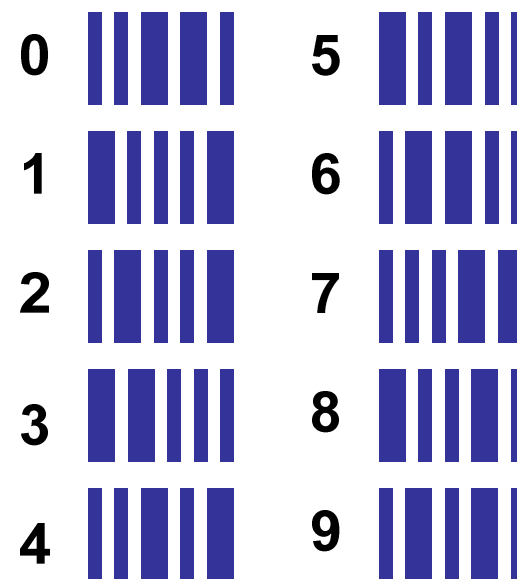
- Kod 5-bitowy: 2 bity zawsze równe 1, a 3 bity zawsze równe 0
- Koduje 10 znaków (cyfry dziesiętne), kody nie są wzajemnie jednoznaczne (ta sama wartość może być zakodowana w różny sposób)

- Kod stałowagowy
- Kod detekcyjny
- Stosowany głównie w **kodach kreskowych**

Liczba dziesiętna	2 z 5 (01236)	2 z 5 (01234)	2 z 5 (74210)
0	01100	01100	11000
1	11000	11000	00011
2	10100	10100	00101
3	10010	10010	00110
4	01010	01010	01001
5	00110	00110	01010
6	10001	10001	01100
7	01001	01001	10001
8	00101	00101	10010
9	00011	00011	10100

Kody liczbowe - Kod 2 z 5 Industrial (1960 r.)

- Jednowymiarowy kod kreskowy kodujący cyfry: 0 ÷ 9
- Znak to 5 pasków: 2 szerokie i 3 wąskie
- Szeroki pasek jest wielokrotnością wąskiego, szerokości muszą być takie same dla całego kodu
- Struktura kodu:
 - start: 11011010
 - numer
 - stop: 11010110



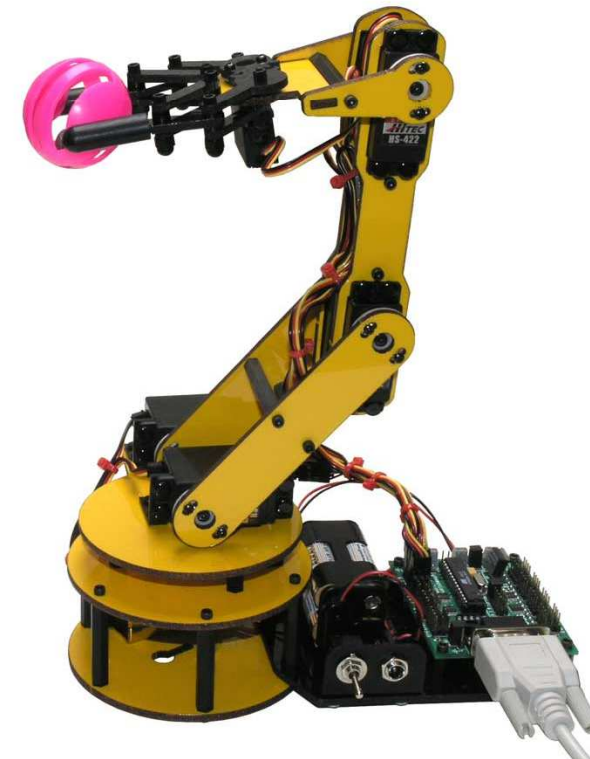
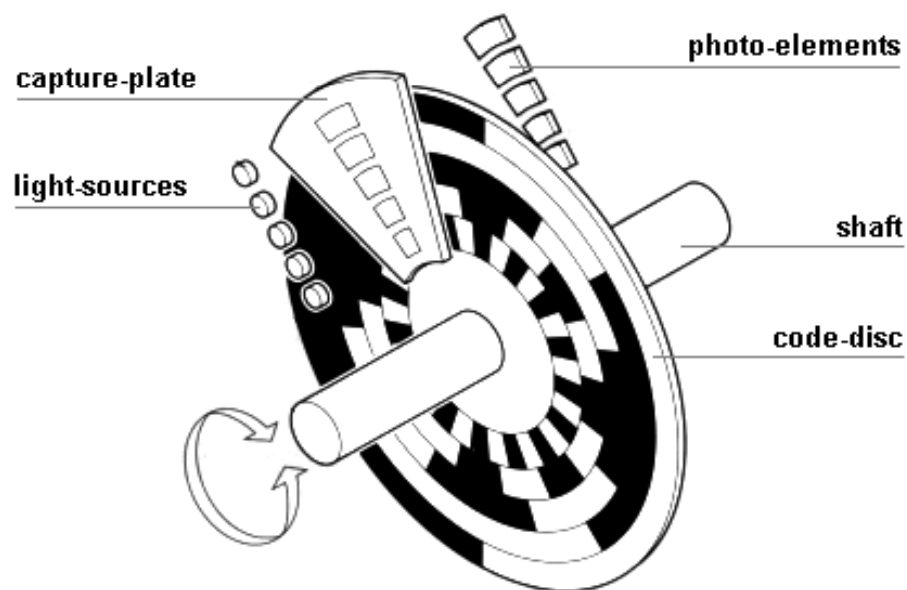
Kod Graya (refleksyjny)

- Kod dwójkowy, bezwagowy, niepozycyjny
- Dwa kolejne słowa kodowe różnią się stanem jednego bitu
- Kod cykliczny - ostatni i pierwszy wyraz również różnią się stanem jednego bitu

<u>kod 1-bitowy</u>	<u>kod 2-bitowy</u>	<u>kod 3-bitowy</u>
0	00	000
1	01	001
	<u>11</u>	011
	10	<u>010</u>
		110
		111
		101
		100

Kod Graya

- Stosowany w przetwornikach analogowo-cyfrowych, do cyfrowego pomiaru analogowych wielkości mechanicznych (np. kąt obrotu)



<http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/10-gates/15-graycode/dual2gray.html>

Koniec wykładu nr 3

Dziękuję za uwagę!
(następny wykład: 14.05.2018)