



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 3

Opracował: dr inż. Jarosław Forenc

Białystok 2024

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

Plan wykładu nr 3

- Instrukcja switch
- Pętla for
- Operatory ++ i –
- Pętle while i do...while
- Tablice jednowymiarowe (wektory)
- Tablice dwuwymiarowe (macierze)

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;
```

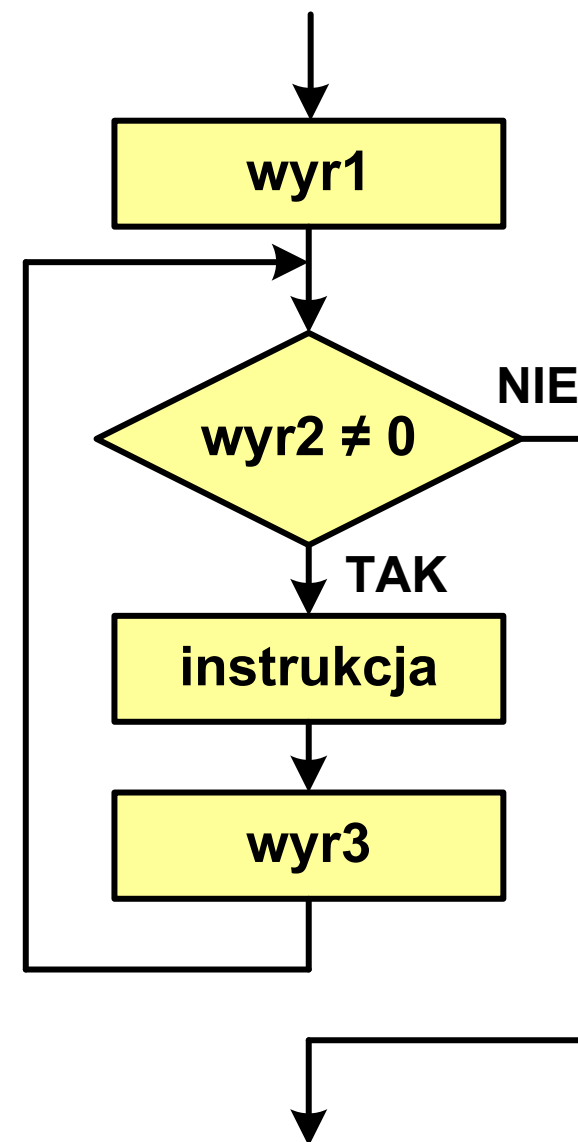
```
}
```

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



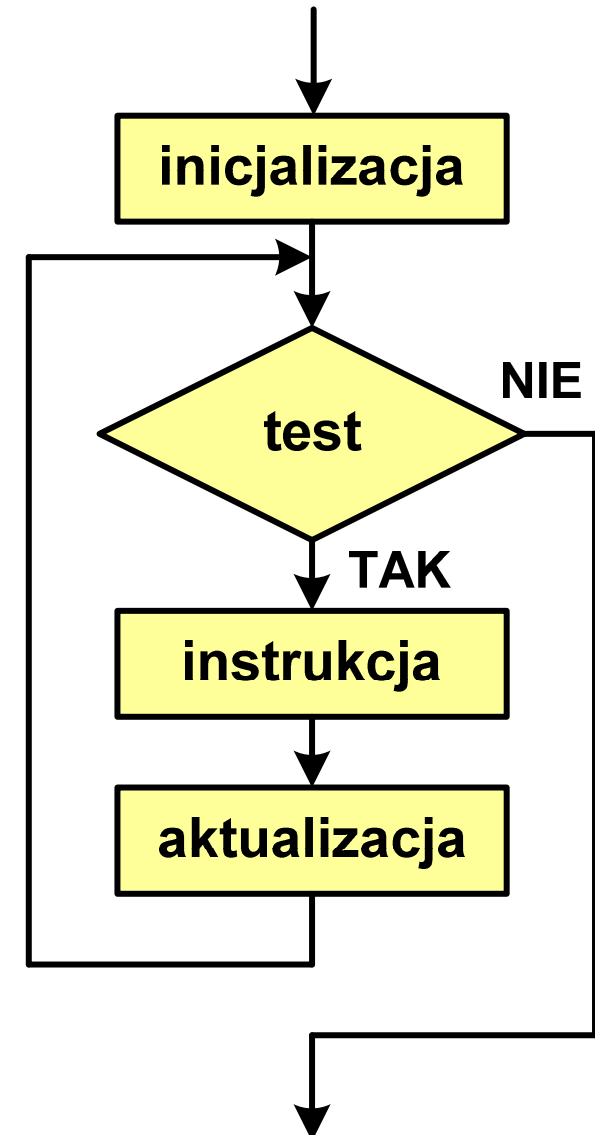
Język C - pętla for

- 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
```

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)

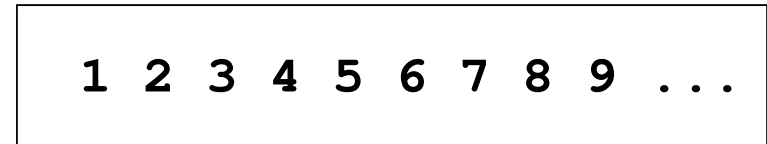
- Błędny warunek - brak wykonania instrukcji

```
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);
```



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

- Operatory `++` i `--` mają jeden z najwyższych priorytetów

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
```

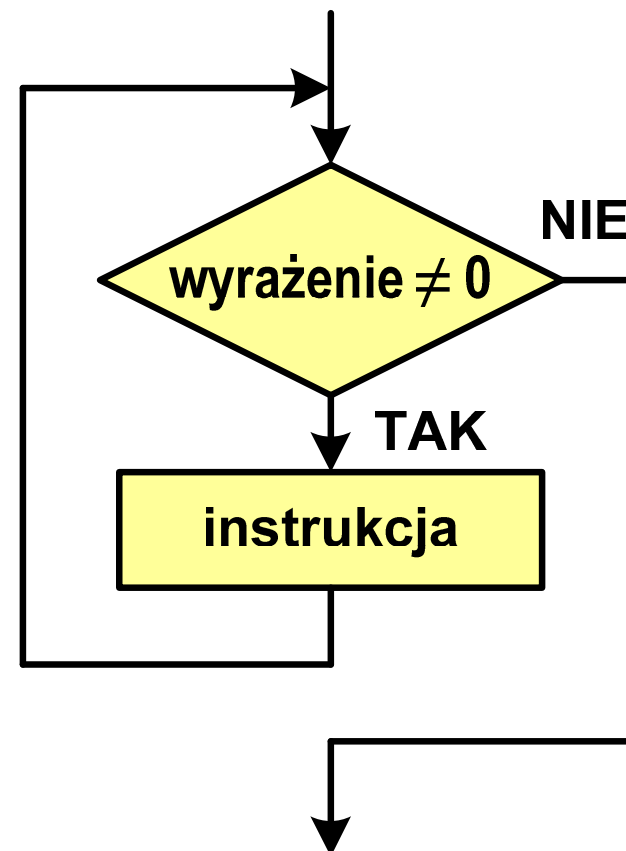
```
Podaj liczbe: 3
Pierwiastek liczby: 1.732051
```

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

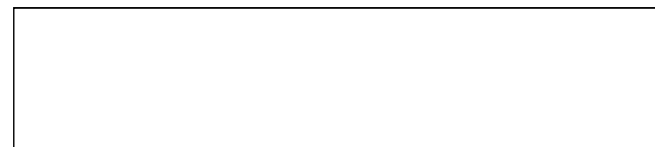
```
int x=0;
while (x<10)
{
    x++;
    if (x%2==0)
        continue;
    if (x%5==0)
        break;
    printf ("%d\n", x);
}
```

- **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)

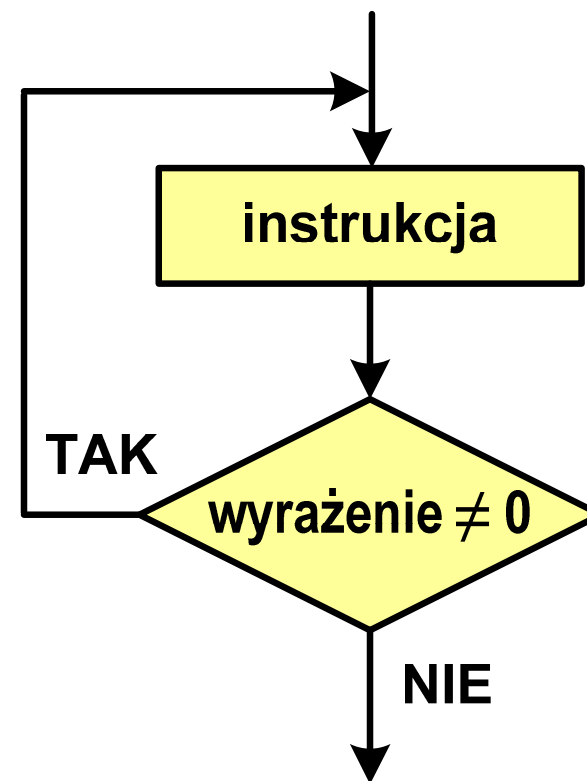
```
while (1)
{
    instrukcja;
    instrukcja;
    ...
}
```

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: pierwiastek kwadratowy (pętla do...while)

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

int main(void)
{
    float x, y;

    do
    {
        printf("Podaj liczbe: ");
        scanf("%f", &x);
    }
    while (x<0);

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

    return 0;
}
```

Podaj liczbe: -3

Podaj liczbe: -5

Podaj liczbe: 3

Pierwiastek liczby: 1.732051

Język C - tablica jednowymiarowa

- **Tablica** - ciągły obszar pamięci, w którym umieszczone są elementy tego samego typu
- **Wektor** - tablica jednowymiarowa

5	3	-2	0	-4
---	---	----	---	----

- liczby całkowite

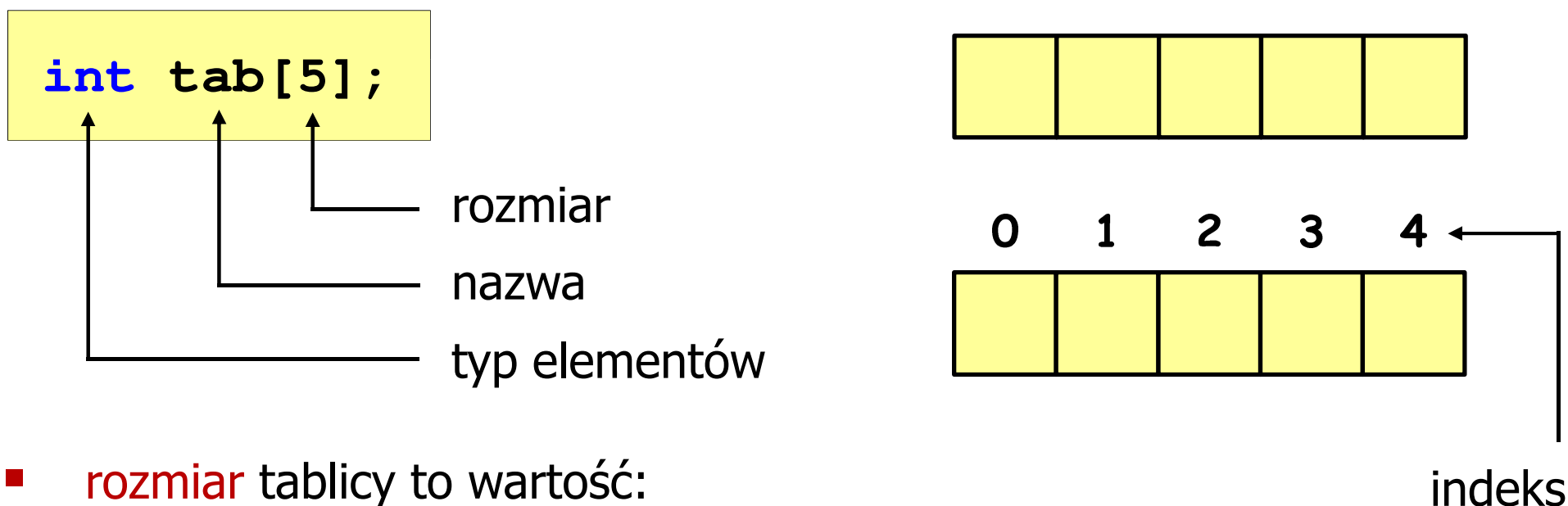
3.1	0.2	2.3	-1.3	1.5	1.1	-4.0
-----	-----	-----	------	-----	-----	------

- liczby rzeczywiste

a	Z	x	&	M	+
---	---	---	---	---	---

- znaki

Język C - deklaracja tablicy jednowymiarowej



- **rozmiar** tablicy to wartość:
 - całkowita, dodatnia
 - znana na etapie kompilacji programu
(stała liczbowa: **5**, `#define N 5`, `const int n = 5;`)

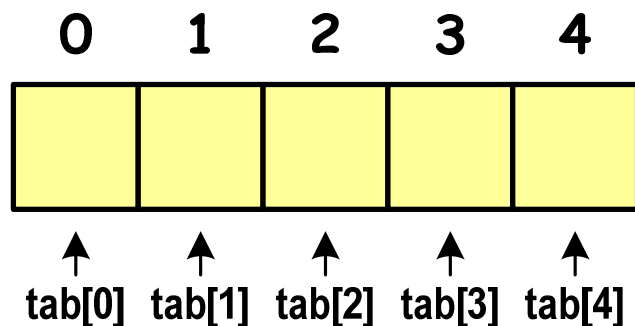
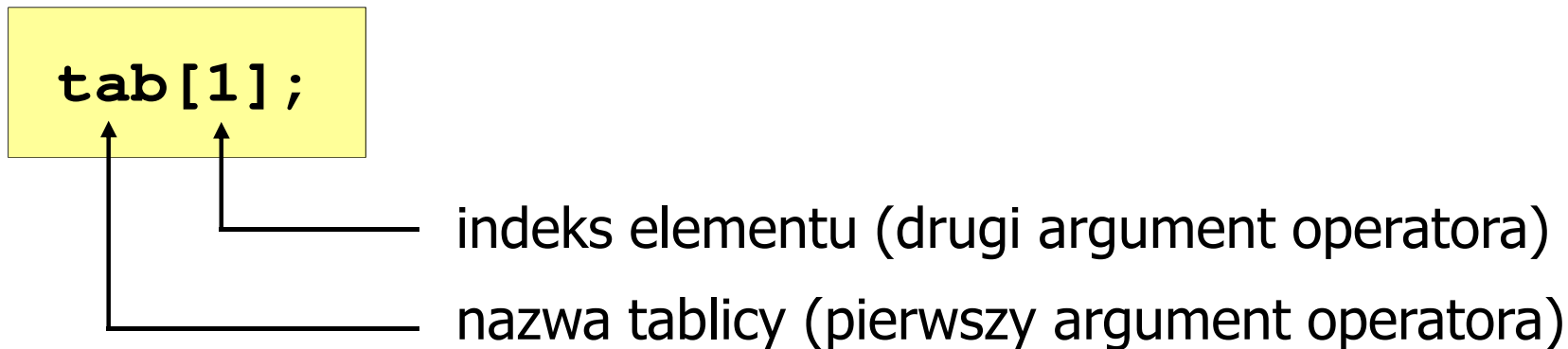
```
int tab[5];
```

```
int tab[N];
```

```
int tab[n];
```

Język C - odwołania do elementów tablicy

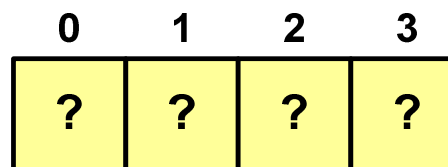
[] - dwuargumentowy operator indeksowania



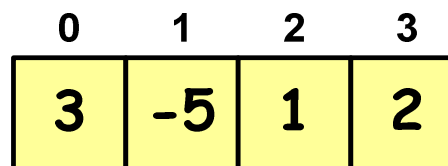
- indeks:
 - stała liczbowa, np. **0**, **1**, **10**
 - nazwa zmiennej, np. **i**, **idx**
 - wyrażenie, np. **i*j+5**

Język C - odwołania do elementów tablicy

```
int tab[4];
```



```
tab[0] = 3;  
tab[1] = -5;  
tab[2] = 1;  
tab[3] = 2;
```



- Każdy element tablicy traktowany jest tak samo jak zmienna typu `int`

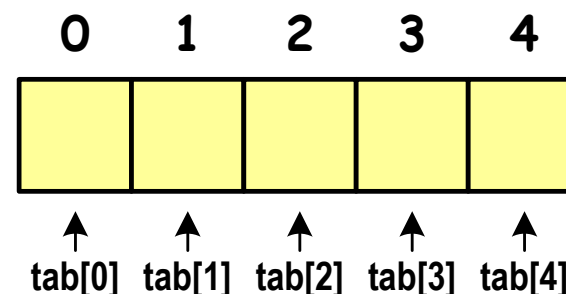
```
printf("%d", tab[0]);
```

```
scanf("%d", &tab[1]);
```

Język C - odwołania do elementów tablicy

- Przy odwołaniach do elementów tablicy kompilator nie sprawdza poprawności indeksów

```
int tab[5];  
tab[5] = 10;
```



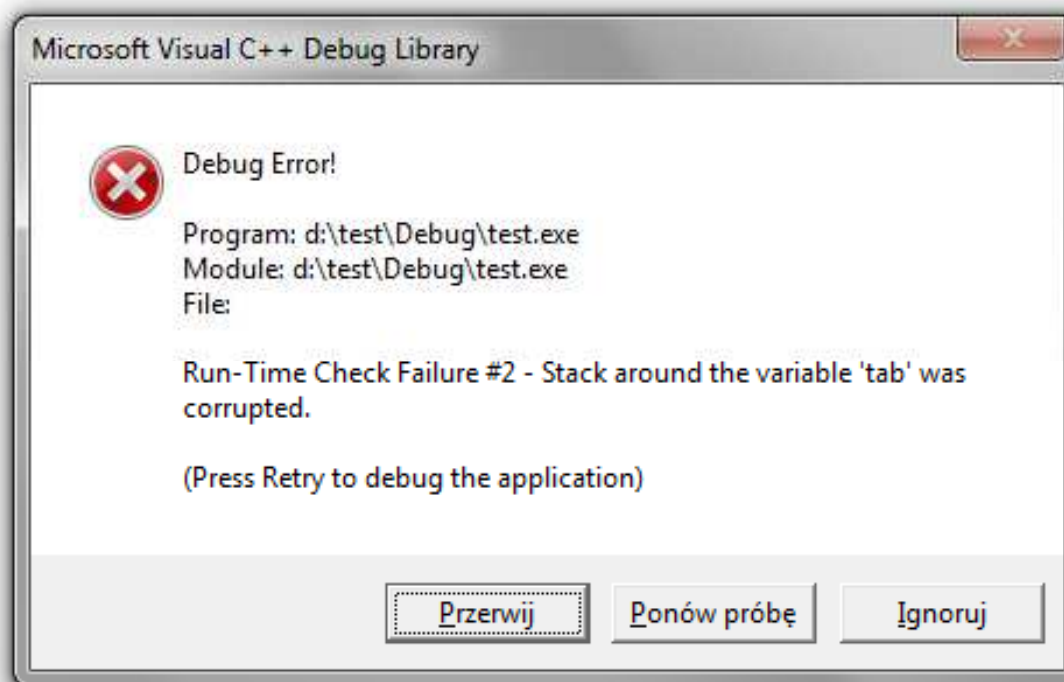
- **błąd!!!** - nie istnieje element **tab[5]**

- Kompilator nie zasygnalizuje błędu
- Program wykona operację
- Środowisko programistyczne może zasygnalizować problem

Język C - odwołania do elementów tablicy

- Przy odwołaniach do elementów tablicy kompilator nie sprawdza poprawności indeksów

```
int tab[5];  
tab[5] = 10;
```



Język C - inicjalizacja tablicy jednowymiarowej

```
int tab[5] = {1, 2, 3, 4, 5};
```

0	1	2	3	4
1	2	3	4	5

```
int tab[5] = {1, 2, 3};
```

0	1	2	3	4
1	2	3	0	0

```
int tab[5] = {1, 2, 3, 4, 5, 6};
```

- błąd kompilacji

```
int tab[] = {1, 2, 3, 4, 5};
```

0	1	2	3	4
1	2	3	4	5

Język C - odwołania do elementów tablicy

- Zapisanie wartości **1** do wszystkich elementów tablicy

```
int tab[5];  
  
tab[0] = 1;  
tab[1] = 1;  
tab[2] = 1;  
tab[3] = 1;  
tab[4] = 1;
```

0	1	2	3	4
1	1	1	1	1

```
int tab[5], i;  
  
for (i=0; i<5; i++)  
    tab[i] = 1;
```

Język C - operacje na dużej ilości danych (tablica)

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

```
int main(void)
```

```
{
```

```
    double U[5] = { 5.0, 10.0, 15.0, 20.0, 25.0 };
```

```
    double I[5] = { 0.16, 0.21, 0.27, 0.33, 0.36 };
```

```
    double R[5];
```

```
    int i;
```

```
    for (i=0; i<5; i++)  
        R[i] = U[i]/I[i];
```

```
    for (i=0; i<5; i++)  
        printf("R%d = %f\n", i+1, R[i]);
```

```
    return 0;
```

```
}
```

R1 = 31.250000

R2 = 47.619048

R3 = 55.555556

R4 = 60.606061

R5 = 69.444444

	0	1	2	3	4
U	5.0	10.0	15.0	20.0	25.0
I	0.16	0.21	0.27	0.33	0.36
R	31.25	47.62	55.56	60.61	69.44

Język C - generator liczb pseudolosowych

- `rand()` - zwraca liczbę pseudolosową - zakres: `0 ... RAND_MAX`
(`0 ... 32767`)
- `srand()` - inicjalizuje generator liczb pseudolosowych
- Plik nagłówkowy: `stdlib.h` (`time.h`)

```
int x, y, z;
srand((unsigned int) time(NULL));
x = rand();           // zakres <0, 32767>
y = rand() % 100;    // zakres <0, 99>
z = rand() % (b-a+1) - a; // zakres <a, b>
```

Język C - operacje na wektorze

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

```
#define N 10
```

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

```
    int tab[N], i;
```

```
    /* generowanie elementów tablicy */
```

```
    srand((unsigned int) time(NULL));
```

```
    for (i=0; i<N; i++)
        tab[i] = rand() % 20;
```

0	1	2	3	4	5	6	7	8	9
11	12	14	9	6	11	6	18	9	10

Język C - operacje na wektorze

```
/* wyświetlenie elementów tablicy */  
  
printf("Elementy tablicy:\n");  
for (i=0; i<N; i++)  
    printf("%d  ", tab[i]);  
printf("\n");
```

Elementy tablicy:

11 12 14 9 6 11 6 18 9 10

0	1	2	3	4	5	6	7	8	9
11	12	14	9	6	11	6	18	9	10

N = 10

Język C - operacje na wektorze

```
/* wyświetlenie elementów w odwrotnej kolejności */  
  
printf("Elementy w odwrotnej kolejności:\n");  
for (i=N-1; i>=0; i--)  
    printf("%d  ", tab[i]);  
printf("\n");
```

```
Elementy w odwrotnej kolejności:  
10  9  18  6  11  6  9  14  12  11
```

0	1	2	3	4	5	6	7	8	9
11	12	14	9	6	11	6	18	9	10

N = 10

Język C - operacje na wektorze

```
/* wyszukanie elementu o najmniejszej wartości */  
  
int min;  
  
min = tab[0];  
for (i=1; i<N; i++)  
    if (tab[i]<min)  
        min = tab[i];  
printf("Wartosc elementu najmniejszego: %d\n",min);
```

Wartosc elementu najmniejszego: 6

0	1	2	3	4	5	6	7	8	9
11	12	14	9	6	11	6	18	9	10

N = 10

Język C - operacje na wektorze

```
/* indeksy elementów o najmniejszej wartości */  
  
printf("Indeksy elementu najmniejszego: ");  
for (i=0; i<N; i++)  
    if (tab[i]==min)  
        printf("%d ", i);  
printf("\n");
```

Indeksy elementu najmniejszego: 4 6

0	1	2	3	4	5	6	7	8	9
11	12	14	9	6	11	6	18	9	10

N = 10

Język C - operacje na wektorze

```
/* suma i średnia arytmetyczna elementów tablicy */  
  
int suma = 0;  
float srednia;  
  
for (i=0; i<N; i++)  
    suma = suma + tab[i];  
srednia = (float) suma/N;  
printf("Suma: %d, srednia: %g\n", suma, srednia);
```

Suma: 106, srednia: 10.6

0	1	2	3	4	5	6	7	8	9
11	12	14	9	6	11	6	18	9	10

N = 10

Język C - operacje na wektorze

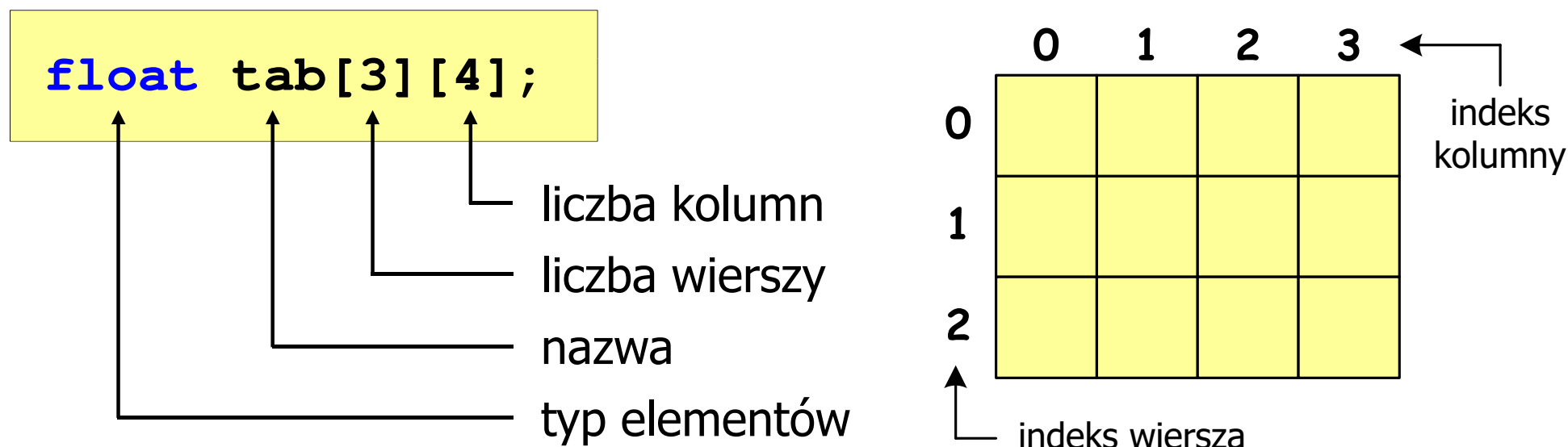
```
/* liczba parzystych elementów tablicy */  
  
int ile = 0;  
  
for (i=0; i<N; i++)  
    if (tab[i]%2==0)  
        ile++;  
printf("Liczba parzystych elementów: %d\n",ile);
```

Liczba parzystych elementów: 6

0	1	2	3	4	5	6	7	8	9
11	12	14	9	6	11	6	18	9	10

N = 10

Język C - deklaracja tablica dwuwymiarowej

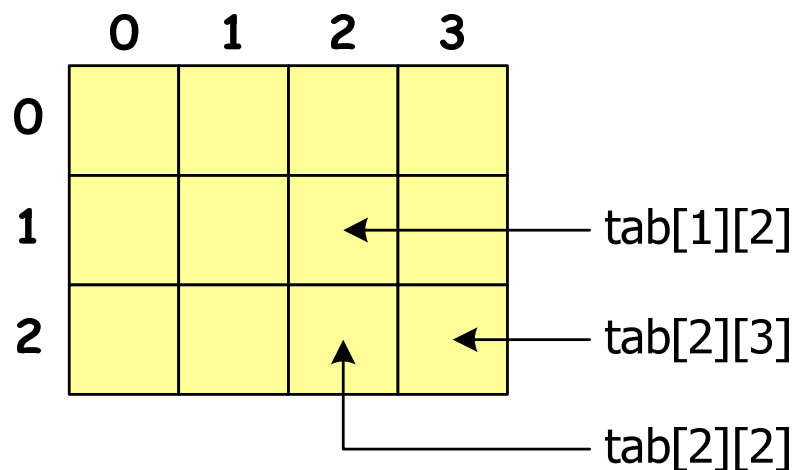
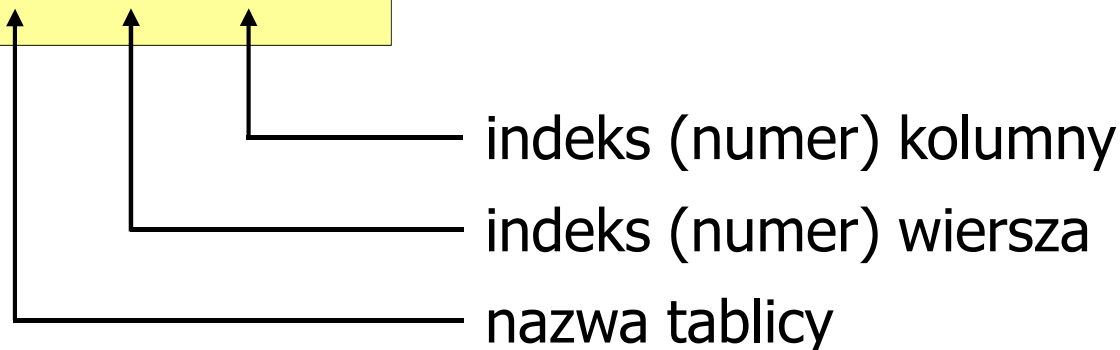


- **Rozmiar** tablicy (liczb wierszy i kolumn) to wartość:
 - całkowita, dodatnia
 - znana na etapie kompilacji programu
(stała liczbowa: **5**, `#define N 5`, `const int n = 5;`)

Język C - odwołania do elementów macierzy

```
tab[1][2];
```

[] - dwuargumentowy operator indeksowania



- Indeks:
 - stała liczbowa, np. 0, 1, 10
 - nazwa zmiennej, np. i, idx
 - wyrażenie, np. $i*j+5$
- Brak sprawdzania poprawności indeksów!

Język C - inicjalizacja elementów macierzy

```
int T[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

	0	1	2
0	1	2	3
1	4	5	6

```
int T[2][3] = {1, 2, 3, 4, 5, 6};
```

	0	1	2
0	1	2	3
1	4	0	0

```
int T[2][3] = {1, 2, 3, 4};
```

	0	1	2
0	1	0	0
1	4	5	0

```
int T[2][3] = {{1}, {4, 5}};
```

Język C - inicjalizacja elementów macierzy

```
int T[2][3] = {0};
```

```
int T[2][3] = {};
```

wyzerowanie elementów macierzy

	0	1	2
0	0	0	0
1	0	0	0

```
int T[][3] = {{1, 2, 3}, {4, 5, 6}};
```

pominięcie liczby wierszy

	0	1	2
0	1	2	3
1	4	5	6

Koniec wykładu nr 3

Dziękuję za uwagę!