

Informatyka 1 (ES1F1002)

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr II, studia stacjonarne I stopnia
Rok akademicki 2022/2023

Wykład nr 8 (28.11.2022)

dr inż. Jarosław Forenc

Plan wykładu nr 8

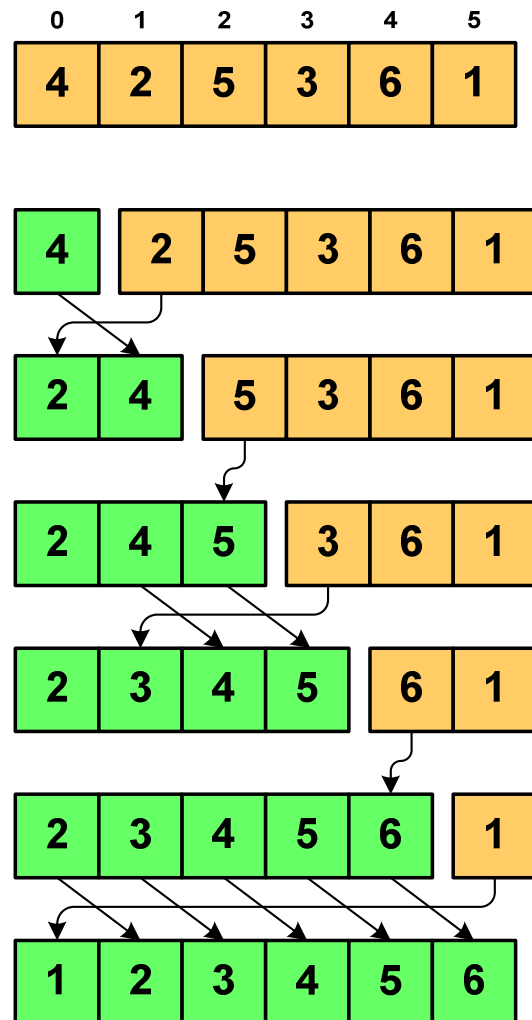
- Algorytmy komputerowe
 - algorytmy sortowania (bąbelkowe, szybkie - quick-sort)
- Język C
 - tablice jednowymiarowe (wektory)
 - tablice dwuwymiarowe (macierze)

Sortowanie

- **Sortowanie** polega na **uporządkowaniu** zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru (wartości każdego elementu)
- W przypadku liczb, sortowanie polega na znalezieniu kolejności liczb zgodnej z relacją \leq lub \geq
- Przykładowe algorytmy sortowania
 - proste wstawianie (insertion sort)
 - proste wybieranie (selection sort)
 - bąbelkowe (bubble sort)
 - szybkie (quick sort)
 - przez scalanie (merge sort)
 - kubełkowe / przez zliczanie (bucket sort)

Proste wstawianie (insertion sort)

Przykład:

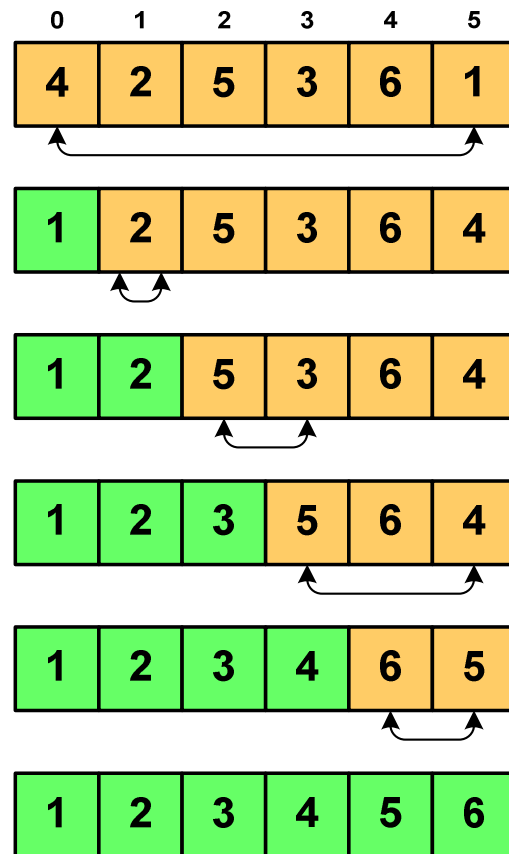


Program w języku C:

```
int main(void)
{
    int tab[N], i, j, tmp;
    // ...
    for (i=1; i<N; i++)
    {
        j=i;
        tmp=tab[i];
        while (tab[j-1]>tmp && j>0)
        {
            tab[j]=tab[j-1];
            j--;
        }
        tab[j]=tmp;
    }
}
```

Proste wybieranie (selection sort)

Przykład:



Program w języku C:

```
int main(void)
{
    int tab[N], i, j, k, tmp;
    // ...
    for (i=0; i<N-1; i++)
    {
        k=i;
        for (j=i+1; j<N; j++)
            if (tab[k]>=tab[j])
                k = j;
        tmp = tab[i];
        tab[i] = tab[k];
        tab[k] = tmp;
    }
}
```


Sortowanie szybkie (Quick-Sort) - faza dzielenia

- Tablica jest dzielona na dwie części wokół pewnego elementu x (nazywanego elementem centralnym)
- Jako element centralny x najczęściej wybierany jest element środkowy (choć może to być także element losowy)
- Przeglądamy tablicę od lewej strony, aż znajdziemy element $a_i \geq x$, a następnie przeglądamy tablicę od prawej strony, aż znajdziemy element $a_j \leq x$
- Zamieniamy elementy a_i i a_j miejscami i kontynuujemy proces przeglądania i zamiany, aż nastąpi spotkanie w środku tablicy
- W ten sposób otrzymujemy tablicę podzieloną na lewą część z wartościami mniejszymi lub równymi x i na prawą część z wartościami większymi lub równymi x

Sortowanie szybkie (Quick-Sort) - faza sortowania

- Zawiera dwa rekurencyjne wywołania tej samej funkcji sortowania: dla lewej i dla prawej części posortowanej tablicy
- Rekurencja zatrzymuje się, gdy wielkość tablicy wynosi 1

Przykład:

- Sortujemy 6-elementową tablicę **tab**:

	0	1	2	3	4	5
tab	4	2	5	3	6	1

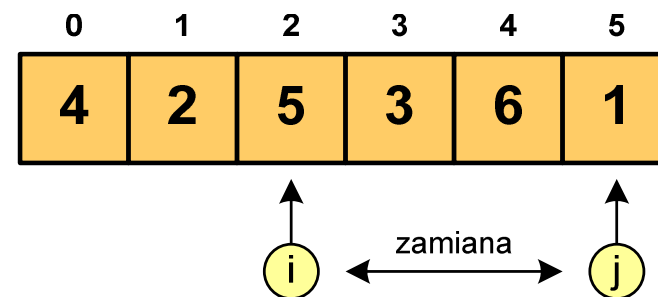
- Wywołanie funkcji **QS()** ma postać:

QS (tab, 0, 5) ;

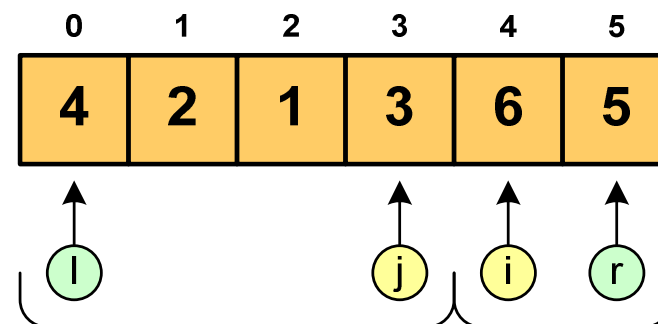
Sortowanie szybkie (Quick-Sort) - $QS(tab, 0, 5)$

- Element środkowy: $(0+5)/2 = 2$, $x = tab[2] = 5$

- Od lewej szukamy $tab[i] \geq x$,
a od prawej szukamy $tab[j] \leq x$,
zamieniamy elementy miejscami



- Poszukiwania kończymy,
gdy indeksy i, j mijają się



- Wywołujemy rekurencyjnie funkcję $QS()$ dla elementów
z zakresów $[l, j]$ i $[i, r]$:

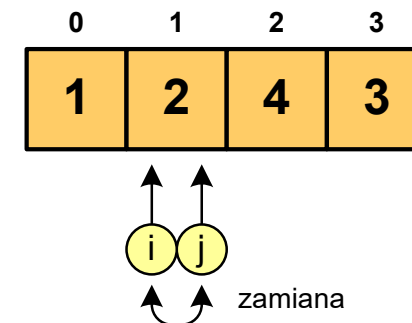
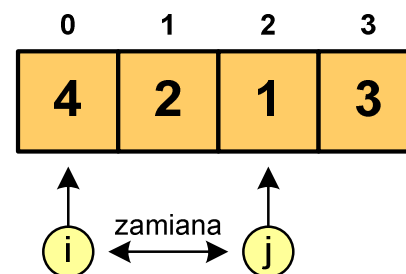
$QS(tab, 0, 3)$;

$QS(tab, 4, 5)$;

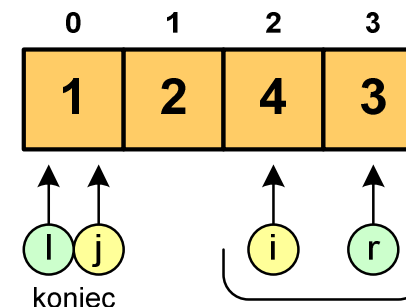
Sortowanie szybkie (Quick-Sort) - $QS(tab, 0, 3)$

- Element środkowy: $(0+3)/2 = 1$, $x = tab[1] = 2$

- Od lewej szukamy $tab[i] \geq x$,
a od prawej szukamy $tab[j] \leq x$,
zamieniamy elementy miejscami



- Poszukiwania kończymy,
gdy indeksy i, j mijają się



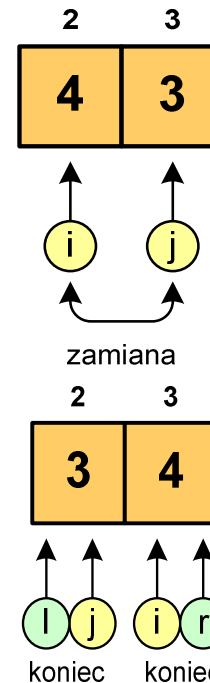
- Wywołanie $QS()$ tylko dla elementów z zakresu $[2,3]$, gdyż
po lewej stronie rozmiar tablicy do posortowania wynosi 1:

$QS(tab, 2, 3);$

Sortowanie szybkie (Quick-Sort) - QS(tab,2,3)

- Element środkowy: $(2+3)/2 = 2$, $x = \text{tab}[2] = 4$

- Od lewej szukamy $\text{tab}[i] \geq x$,
a od prawej szukamy $\text{tab}[j] \leq x$,
zamieniamy elementy miejscami

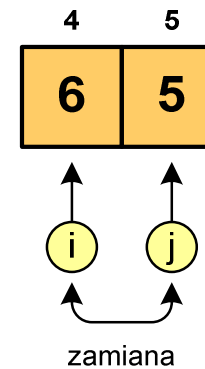


- Poszukiwania kończymy,
gdy indeksy i, j mijają się
- Rozmiar obu tablic do posortowania wynosi 1 więc nie ma nowych wywołań funkcji QS()

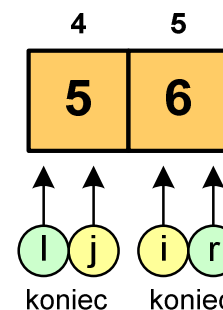
Sortowanie szybkie (Quick-Sort) - $QS(tab, 4, 5)$

- Element środkowy: $(4+5)/2 = 4$, $x = tab[4] = 6$

- Od lewej szukamy $tab[i] \geq x$,
a od prawej szukamy $tab[j] \leq x$,
zamieniamy elementy miejscami



- Poszukiwania kończymy,
gdy indeksy i, j mijają się



- Rozmiar obu tablic do posortowania wynosi 1 więc nie ma nowych wywołań funkcji $QS()$

Sortowanie szybkie (Quick-Sort) - funkcja w C

```
void QuickSort(int tab[], int l, int r)
{
    int i, j, x, y;

    i=l;
    j=r;
    x=tab[(l+r)/2];
    do
    {
        while (tab[i]<x) i++;
        while (x<tab[j]) j--;
        if (i<=j)
        {
            y=tab[i];
            tab[i]=tab[j];
            tab[j]=y;
            i++; j--;
        }
    } while (i<=j);
    if (l<j) QuickSort(tab, l, j);
    if (i<r) QuickSort(tab, i, r);
}
```

Funkcja `qsort()` w języku C

- Quick-Sort został zaimplementowany w języku C w funkcji:

QSORT

stdlib.h

```
void qsort(void *baza, size_t n, size_t size,  
          (*funkcja)(const void *element1, const void *element2));
```

- funkcja `qsort()` sortuje metodą Quick-Sort tablicę wskazywaną przez argument `baza` i zawierającą `n` elementów o rozmiarze `size`
- funkcja `qsort()` posługuje się funkcją porównującą `funkcja()`, której argumentami są wskazania do elementów tablicy `baza`
- `funkcja()` powinna zwracać wartości:
 - `< 0`, gdy `*element1 < *element2`
 - `== 0`, gdy `*element1 == *element2`
 - `> 0`, gdy `*element1 > *element2`

Funkcja qsort() w języku C - przykład (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 10

void generuj(int tab[])
{
    int i;
    srand(time(NULL));
    for (i=0;i<N;i++)
        tab[i]=rand()%100;
}

void drukuj(int tab[])
{
    int i;
    for (i=0;i<N;i++)
        printf("%2d  ",tab[i]);
    printf("\n");
}
```


Funkcja qsort() w języku C - przykład (2/2)

```
int funkcja(const void *element1, const void *element2)
{
    if (*(int*)element1 < *(int*)element2) return -1;
    if (*(int*)element1 == *(int*)element2) return 0;
    if (*(int*)element1 > *(int*)element2) return 1;
}
```

```
int main()
{
    int tab[N];

    generuj(tab);
    drukuj(tab);

    printf("\nqsort:\n");
    qsort((void*)tab, (size_t)N, sizeof(int), funkcja);
    drukuj(tab);

    return 0;
}
```

Funkcja qsort() w języku C - przykład (2/2)

```
int funkcja(const  
{  
    if (*(int*)ele  
    if (*(int*)ele  
    if (*(int*)ele  
}
```

```
65  22  15  26  87  43   3  21  11  73  
  
qsort:  
 3  11  15  21  22  26  43  65  73  87
```

```
int main()  
{  
    int tab[N];  
  
    generuj(tab);  
    drukuj(tab);  
  
    printf("\nqsort:\n");  
    qsort((void*)tab, (size_t)N, sizeof(int), funkcja);  
    drukuj(tab);  
  
    return 0;  
}
```

Język C - tablica elementów

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

wektor

5	3	-2	1	-4
---	---	----	---	----

macierz

a	c	d	m
p	d	q	l
a	t	x	v

1.2	2.5	2.0	10.0
-0.1	4.3	6.2	-5.1
0.0	12.2	4.1	-2.2

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

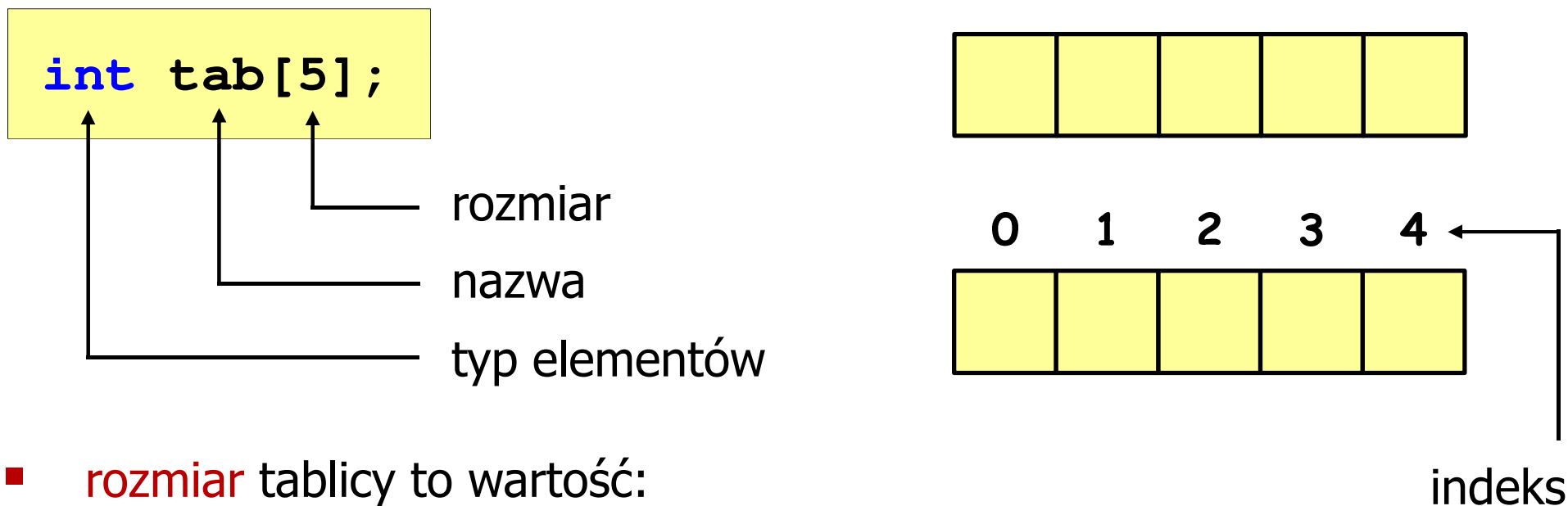
3.1	0.2	2.3	-1.3	1.5	1.1	-4.0
-----	-----	-----	------	-----	-----	------

- 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;`)

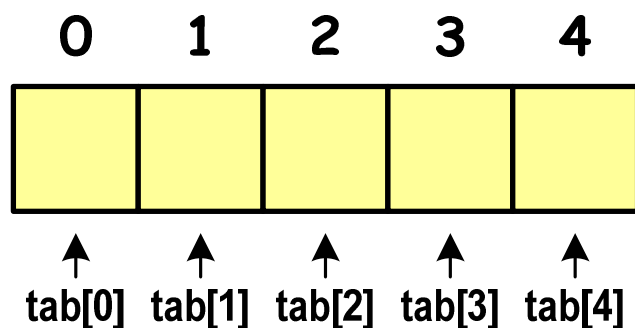
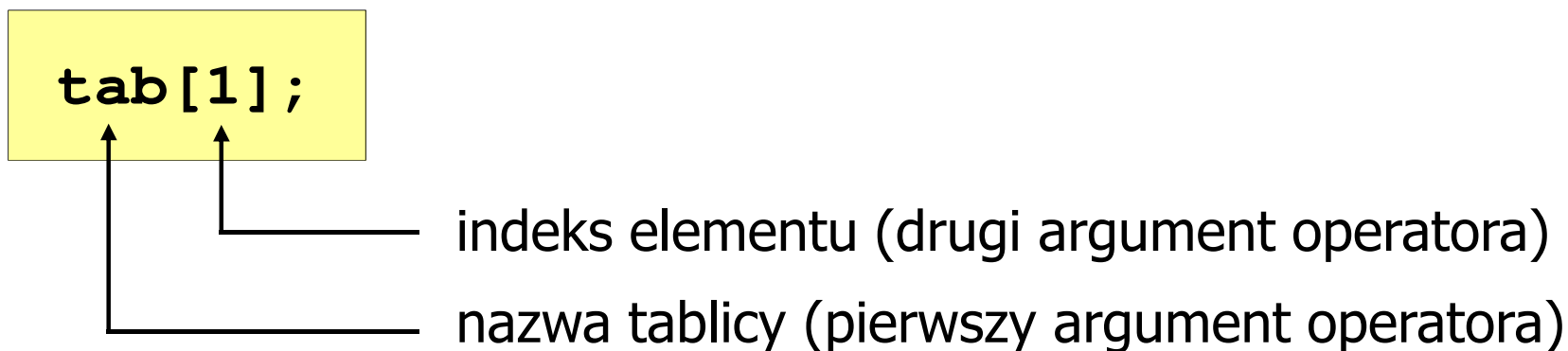
```
int tab[5];
```

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

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

Język C - odwołania do elementów tablicy

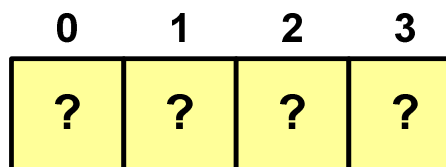
[] - dwuargumentowy operator indeksowania



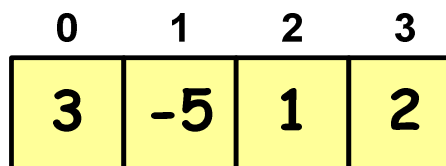
- indeks:
 - stała liczbowa, np. **0**, **1**, **10**
 - nazwa zmiennej, np. **i**, **idx**
 - wyrażenie, np. **i*j+5**

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`

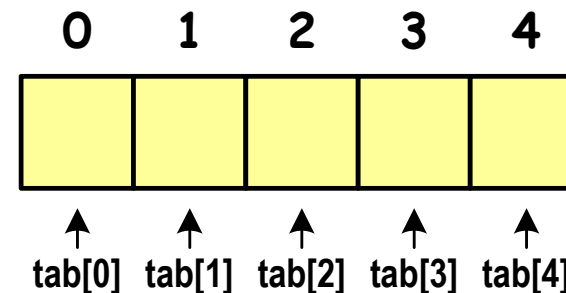
```
printf("%d", tab[0]);
```

```
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;
```



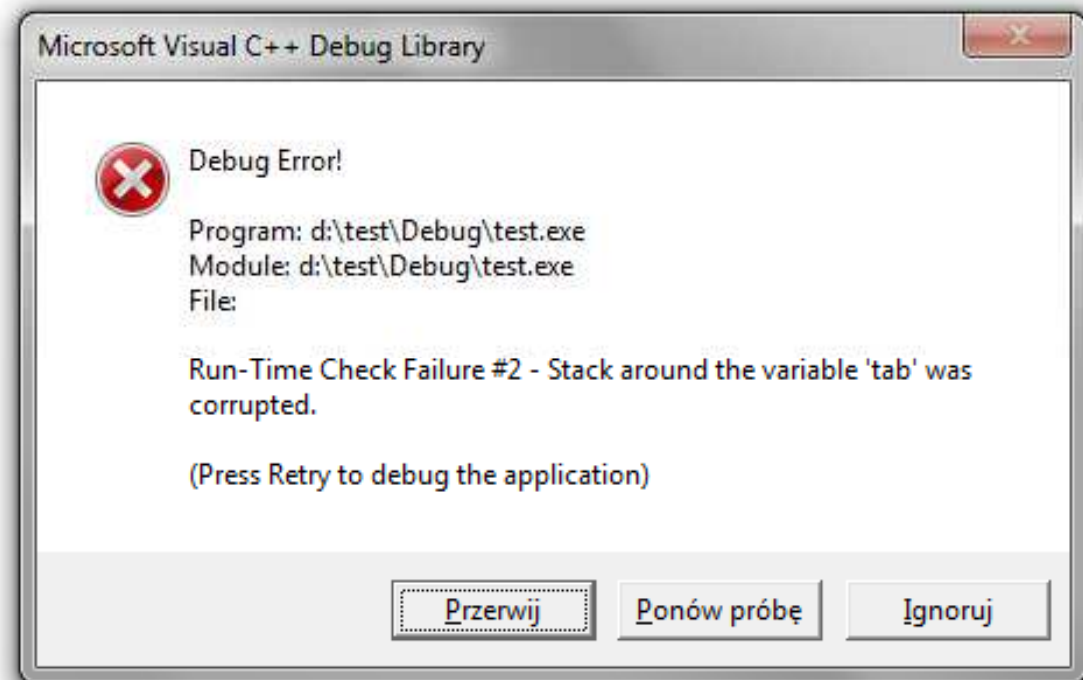
- **błąd!!!** - nie istnieje element **tab[5]**

- 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

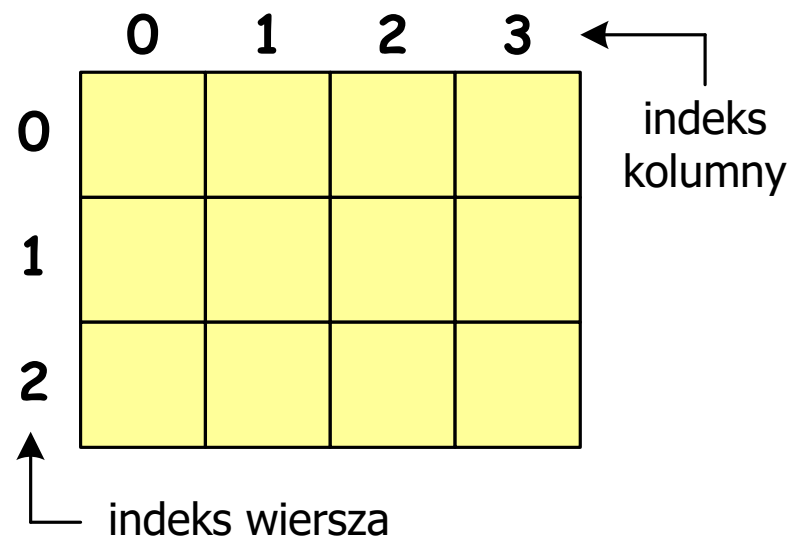
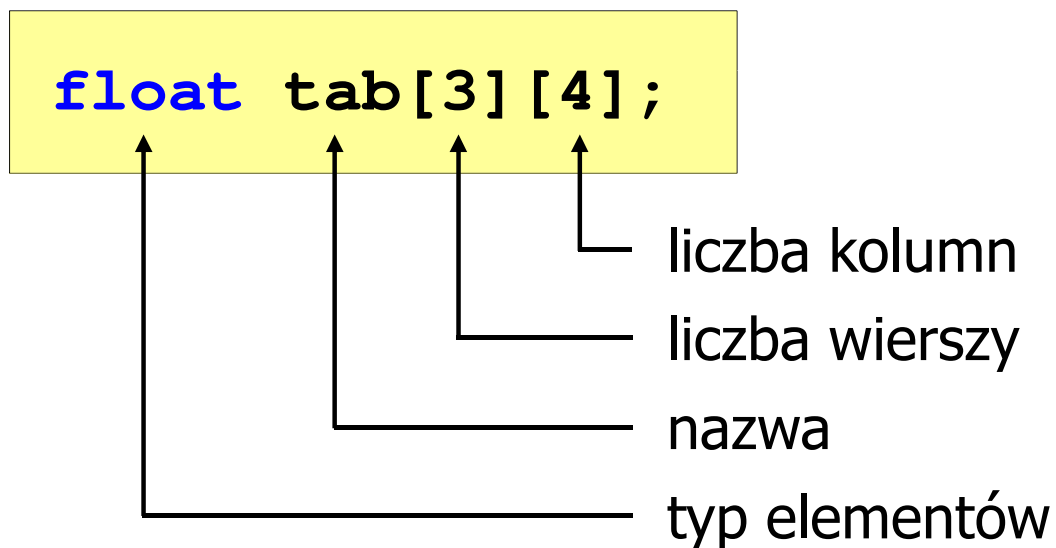
```
/* liczba parzystych elementów tablicy */  
  
int ile = 0;  
  
for (i=0; i<N; i++)  
    if (tab[i]%2==0)  
        ile++;  
printf("Liczba parzystych elementów: %d\n",ile);
```

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

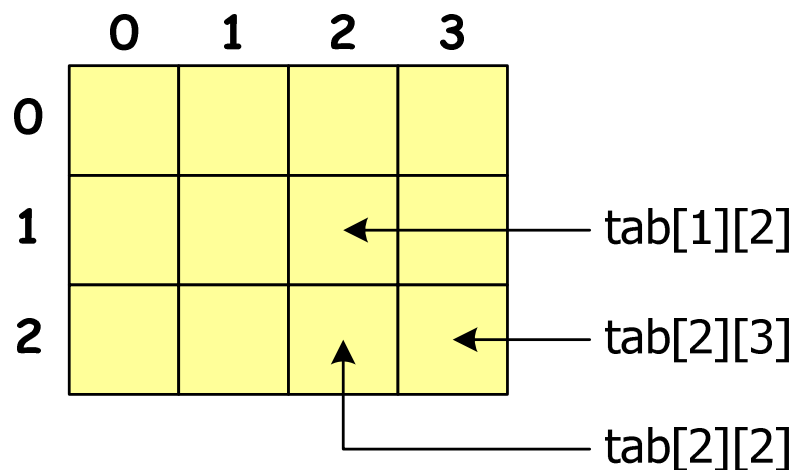
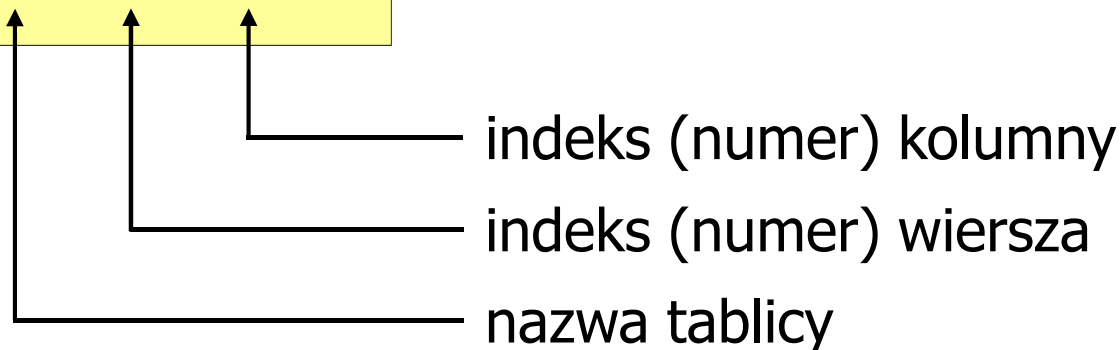


- **Rozmiar** tablicy (liczb wierszy i kolumn) to wartość:
 - całkowita, dodatnia
 - znana na etapie kompilacji programu
(stała liczbowa: **5**, `#define N 5`, `const int n = 5;`)

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

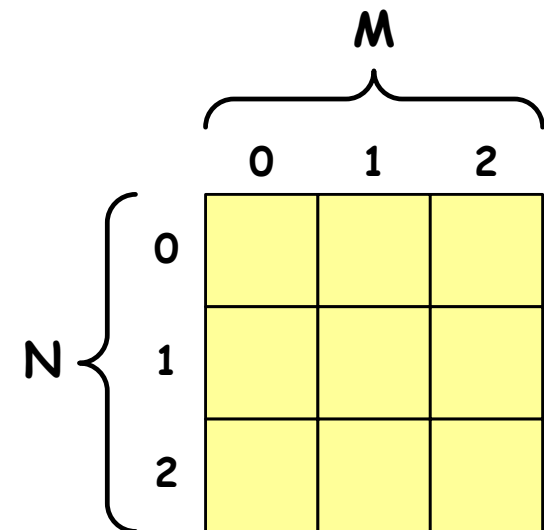
	0	1	2
0	1	2	3
1	4	5	6

Język C - operacje na macierzy

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

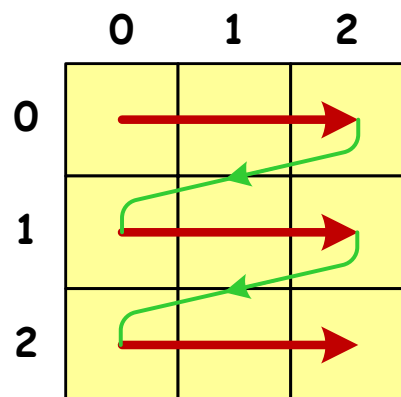
#define N 3      /* liczba wierszy */
#define M 3      /* liczba kolumn */

int main(void)
{
    int tab[N][M];
    int i, j;
```



Język C - operacje na macierzy

```
/* generowanie pseudolosowe elementów macierzy */  
  
srand((unsigned int) time(NULL));  
  
for (i=0; i<N; i++)  
    for (j=0; j<M; j++)  
        tab[i][j] = rand() % 10;
```



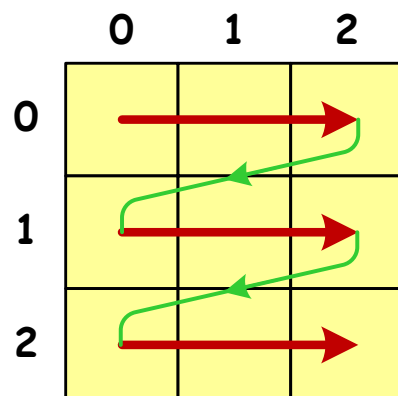
kolejność zapisywania
wartości elementów
macierzy

		M		
		0	1	2
N	0	9	3	1
	1	6	4	8
	2	9	4	6

Język C - operacje na macierzy

```
/* wyświetlenie elementów macierzy */  
  
for (i=0; i<N; i++)  
{  
    for (j=0; j<M; j++)  
        printf("%3d", tab[i][j]);  
    printf("\n");  
}
```

9	3	1
6	4	8
9	4	6



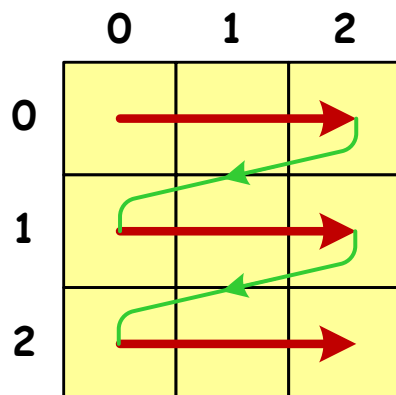
	0	1	2
0	9	3	1
1	6	4	8
2	9	4	6

Język C - operacje na macierzy

```
/* poszukiwanie elementu o wartości minimalnej */
```

```
int min = tab[0][0];  
for (i=0; i<N; i++)  
    for (j=0; j<M; j++)  
        if (tab[i][j] < min)  
            min = tab[i][j];  
printf("Wartosc min: %d\n",min);
```

Wartosc min: 1



	0	1	2
0	9	3	1
1	6	4	8
2	9	4	6

Język C - operacje na macierzy

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

	0	1	2
0			
1			
2			

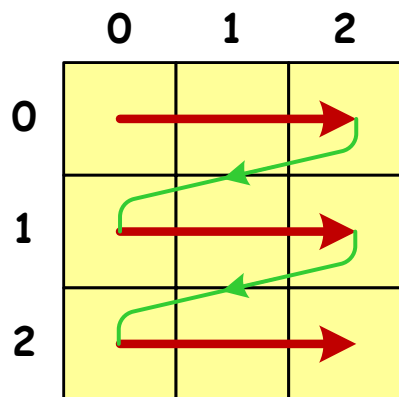
	0	1	2
0	9	3	1
1	6	4	8
2	9	4	6

Suma: 50
Srednia: 5.555555

Język C - operacje na macierzy

```
/* sumy elementów w poszczególnych wierszach */  
  
for (i=0; i<N; i++)  
{  
    suma = 0;  
    for (j=0; j<M; j++)  
        suma = suma + tab[i][j];  
    printf("Suma wiersza %d = %d\n", i, suma);  
}
```

	0	1	2
0			
1			
2			



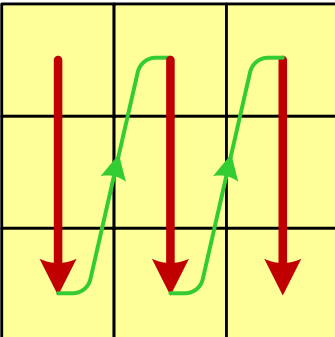
	0	1	2
0	9	3	1
1	6	4	8
2	9	4	6

Suma wiersza 0 = 13
Suma wiersza 1 = 18
Suma wiersza 2 = 19

Język C - operacje na macierzy

```
/* sumy elementów w poszczególnych kolumnach */  
for (j=0; j<M; j++)  
{  
    suma = 0;  
    for (i=0; i<N; i++)  
        suma = suma + tab[i][j];  
    printf("Suma kolumny %d = %d\n", j, suma);  
}
```

	0	1	2
0			
1			
2			



	0	1	2
0	9	3	1
1	6	4	8
2	9	4	6

Suma kolumny 0 = 24
Suma kolumny 1 = 11
Suma kolumny 2 = 15

Język C - operacje na macierzy

```
/* sumy elementów nad, na i poniżej przekątnej */  
  
suma = suma1 = suma2 = 0;  
for (i=0; i<N; i++)  
    for (j=0; j<M; j++)  
    {  
        if (i < j) suma1+=tab[i][j]; /* nad */  
        if (i > j) suma2+=tab[i][j]; /* pod */  
        if (i == j) suma+=tab[i][j]; /* na */  
    }  
  
printf("Suma nad: %d\n", suma1);  
printf("Suma na: %d\n", suma);  
printf("Suma pod: %d\n", suma2);
```

```
Suma nad: 12  
Suma na: 19  
Suma pod: 19
```


Język C - operacje na macierzy

		j		
		0	1	2
i	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

$i < j$

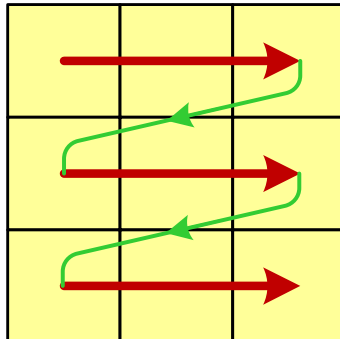
		j		
		0	1	2
i	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

$i = j$

		j		
		0	1	2
i	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

$i > j$

	0	1	2
0			
1			
2			



	0	1	2
0	9	3	1
1	6	4	8
2	9	4	6

Suma nad: 12
 Suma na: 19
 Suma pod: 19

Koniec wykładu nr 8

Dziękuję za uwagę!