Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Materiały do wykładu z przedmiotu:
Informatyka
Kod: EDS1A1 007

WYKŁAD NR 3

Opracował: dr inż. Jarosław Forenc

Białystok 2018

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

- Pętle while i do…while
- Tablice jednowymiarowe (wektory)
- Tablice dwuwymiarowe (macierze)
- Łańcuchy znaków w języku
- Struktury

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: -3
Blad! Liczba ujemna
```

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

Język C - pierwiastek kwadratowy (pętla while)

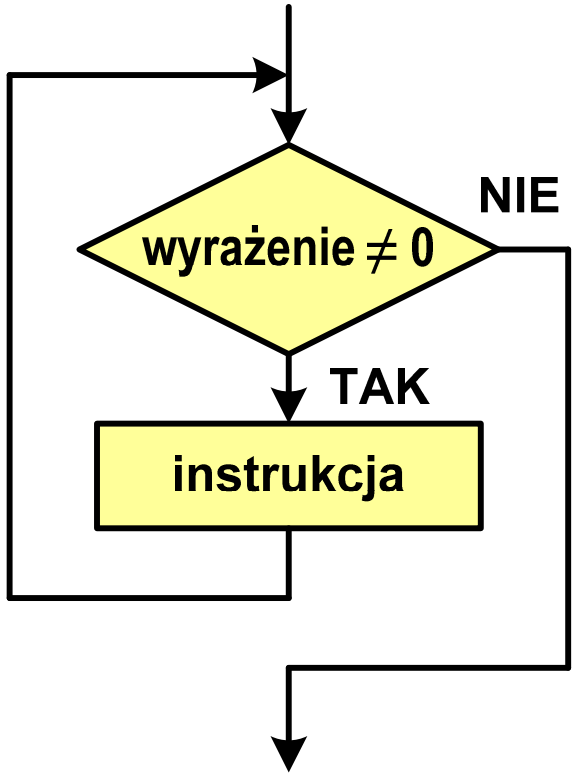
```c
#include <stdio.h>
#include <math.h>

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

```c
#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 liczbe: ");
scanf("%d",&x);

while(x>0)
{
    suma = suma + x;

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

wczytanie danych

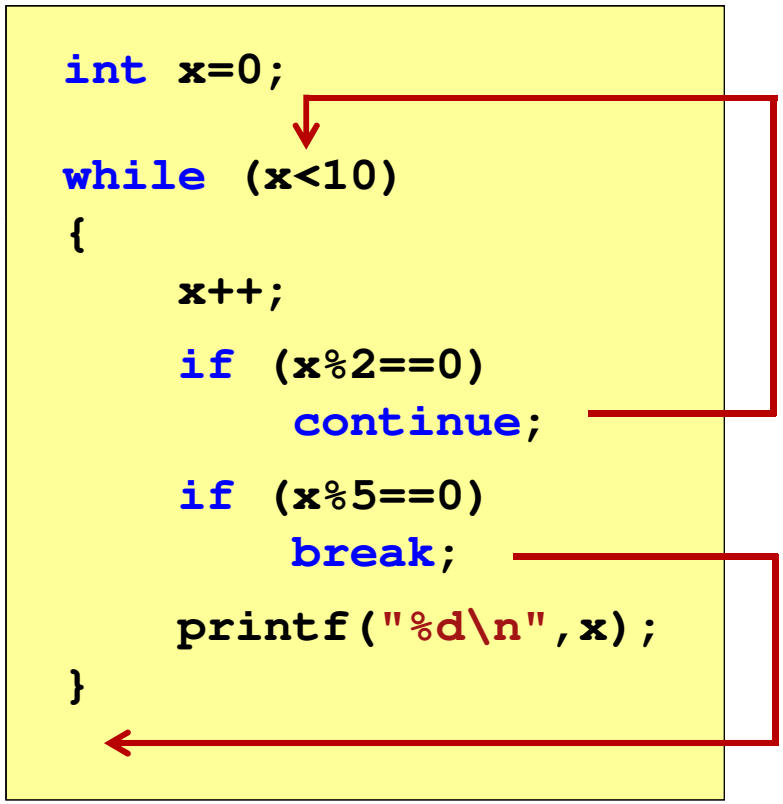
operacje na danych

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)

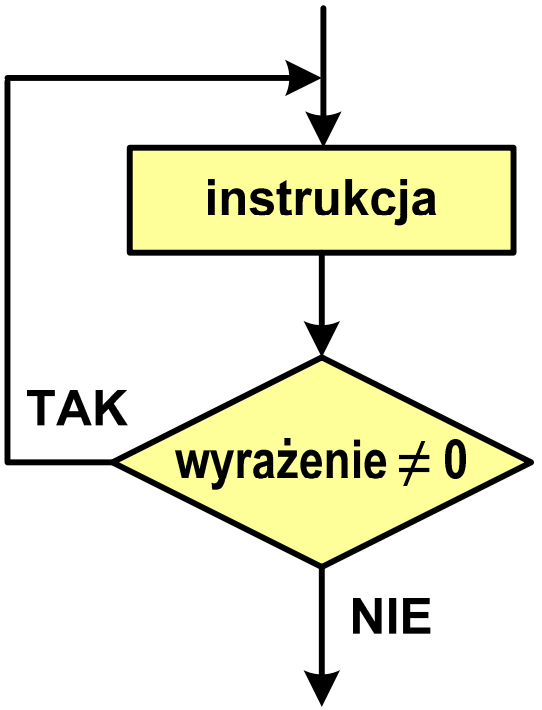
```
while (1)
{
     instrukcja
     instrukcja
     ...
}
```

- 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

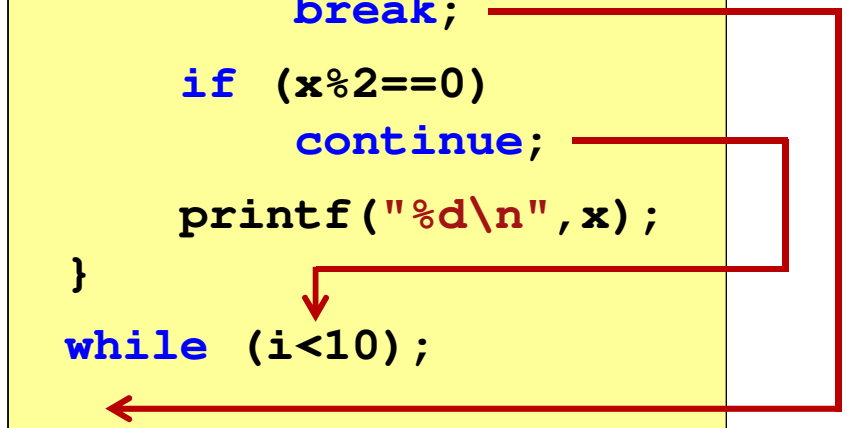
```
int x = 10;
do
    x = x - 1;
while (x>0);
```

```
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 - tablica elementów

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

wektor

5	3	-2	1	-4

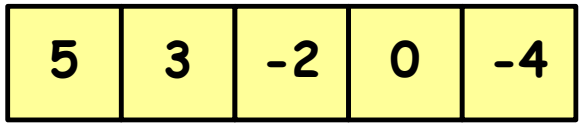
macierz

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

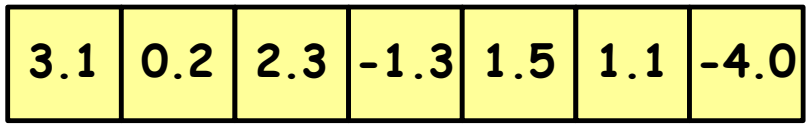
1.2	2.5	2.0	10.0
-0.1	4.3	6.2	-5.1
0.0	12.2	4.1	-2.2

Język C - tablica jednowymiarowa

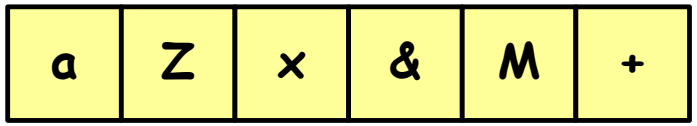
- Tablica - ciągły obszar pamięci, w którym umieszczone są elementy tego samego typu
- Wektor - tablica jednowymiarowa



- liczby całkowite

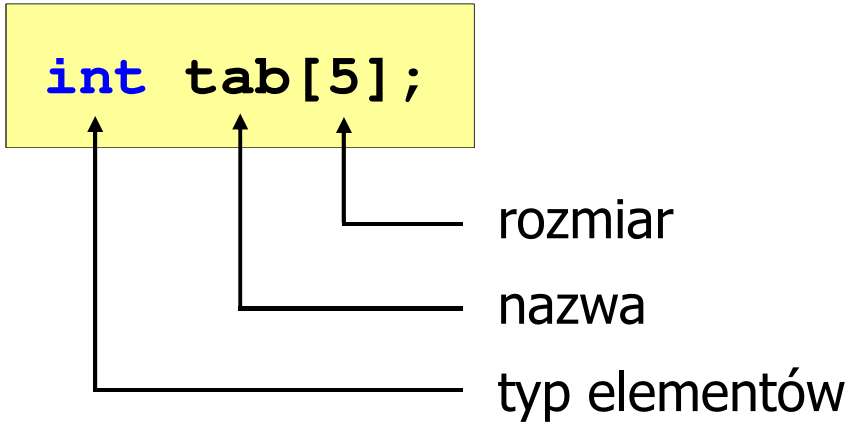


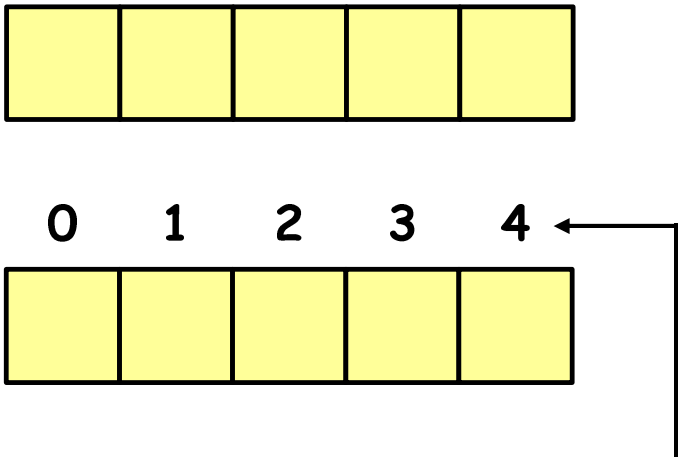
- liczby rzeczywiste



- znaki

Język C - deklaracja tablica jedniowymiarowej





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

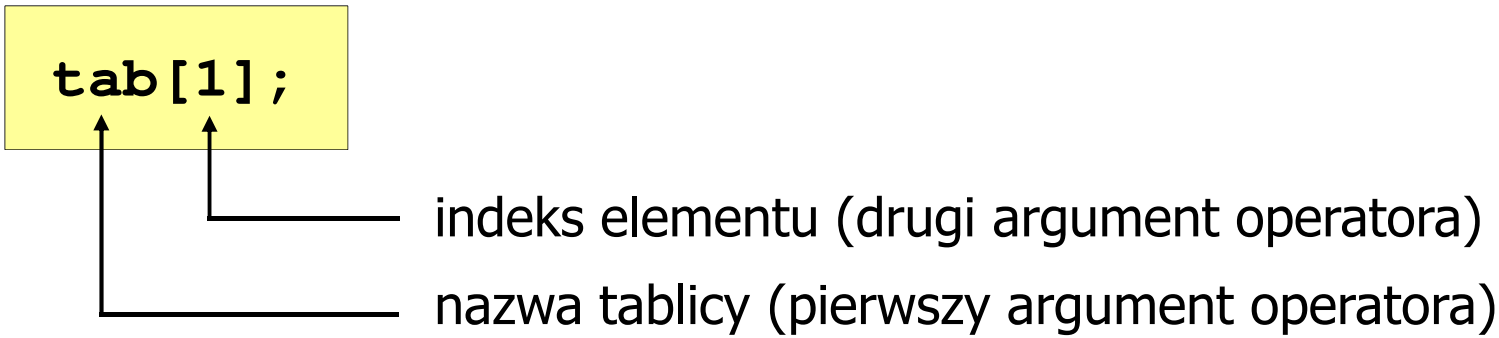
```
int tab[5];
```

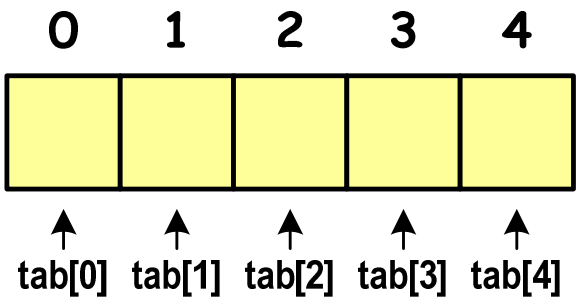
```
int tab[N];
```

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

Język C - odwołania do elementów tablicy

[ ] - dwuargumentowy operator indeksowania

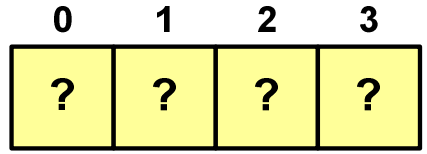




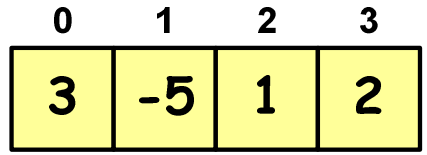
- indeks:
  - stała liczbowa, np. 0, 1, 10
  - nazwa zmiennej, np. i, idx
  - wyrażenie, np. i*j+5

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

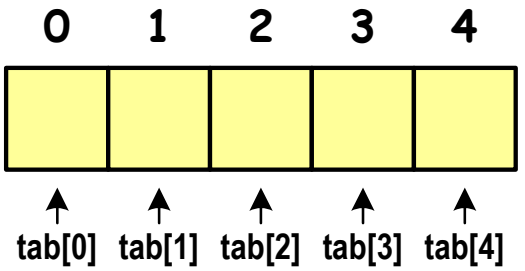
```
printf("%d",tab[0]);
```

```
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;
```



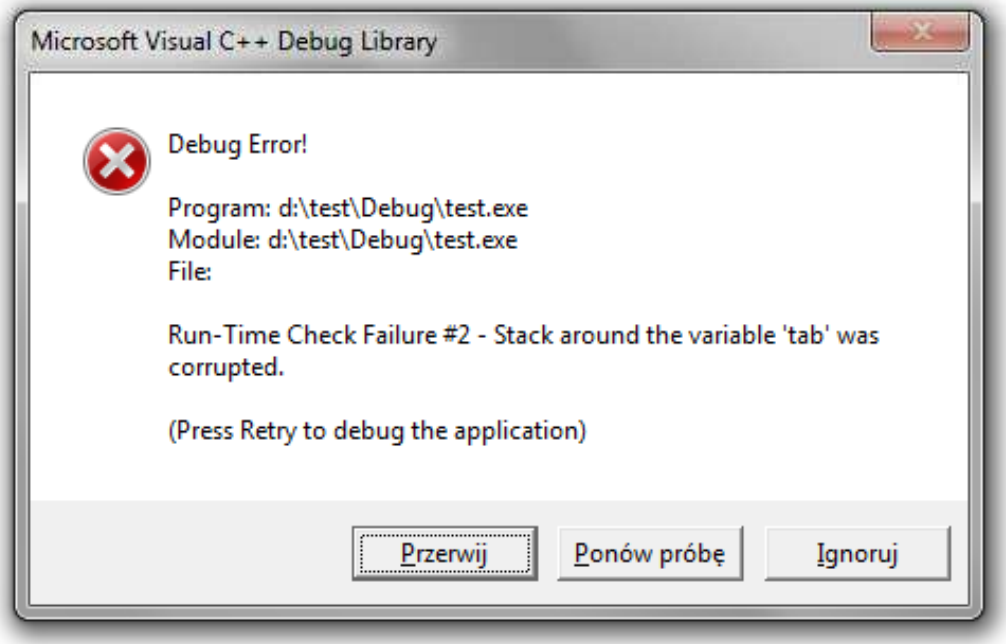
- błąd!!! - nie istnieje element tab[5]

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

Język C - odwołania do elementów tablicy

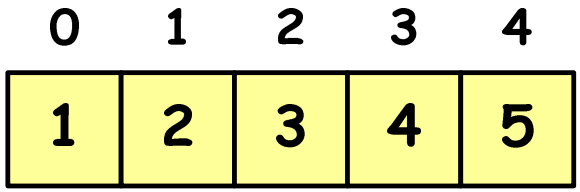
- Przy odwołaniach do elementów tablicy kompilator nie sprawdza poprawności indeksów

```
int tab[5];

tab[5] = 10;
```

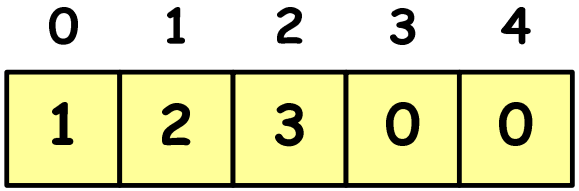


Język C - inicjalizacja tablicy jednowymiarowej

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



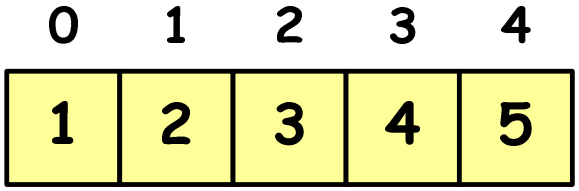

```
int tab[5] = {1,2,3};
```




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

- błąd kompilacji

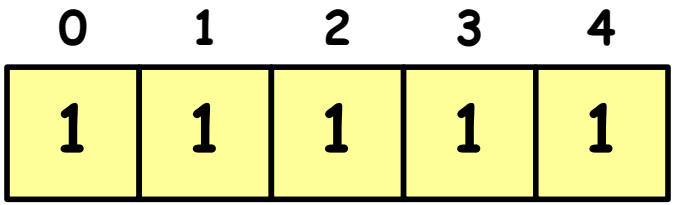
```
int tab[] = {1,2,3,4,5};
```



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;
```



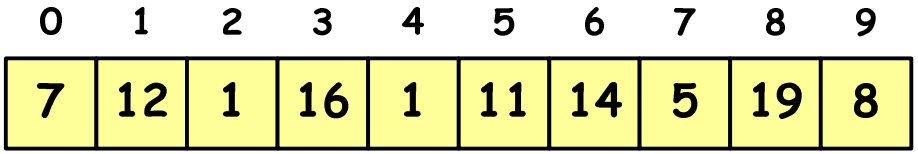

```
int tab[5], i;

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

Język C - generator liczb pseudolosowych

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

```
int x, y;
srand((unsigned int) time(NULL));
x = rand();                 // zakres <0,32767>
y = rand() % 100;           // zakres <0,99>
```

Język C - operacje na wektorze

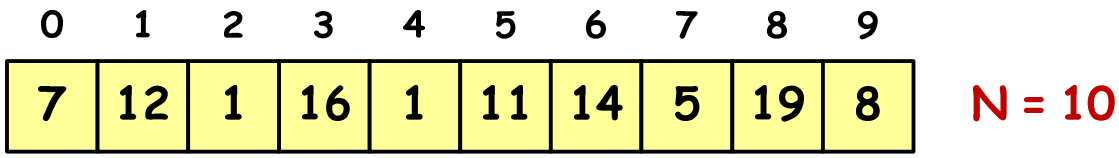



```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#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;
```

Język C - operacje na wektorze

```
/* wyświetlenie elementów tablicy */

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

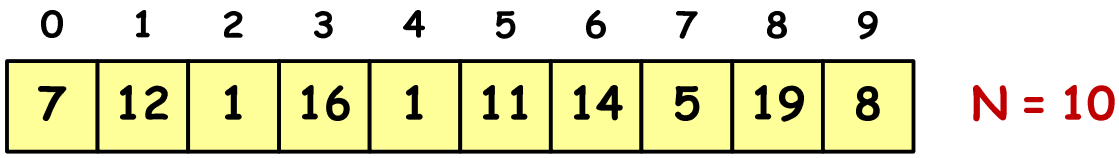
```
Elementy tablicy:
7  12  1  16  1  11  14  5  19  8
```



Język C - operacje na wektorze

```
/* wyświetlenie elementów w odwrotnej kolejności */

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

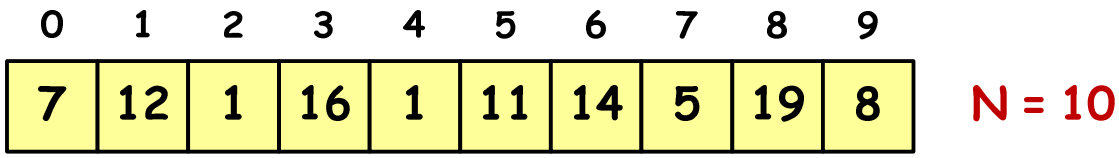
```
Elementy w odwrotnej kolejnosci:
8  19  5  14  11  1  16  1  12  7
```



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: 1
```



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: 2  4
```

0	1	2	3	4	5	6	7	8	9
7	12	1	16	1	11	14	5	19	8

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: 94, srednia: 9.4
```

0	1	2	3	4	5	6	7	8	9
7	12	1	16	1	11	14	5	19	8

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 elementow: %d\n",ile);
```

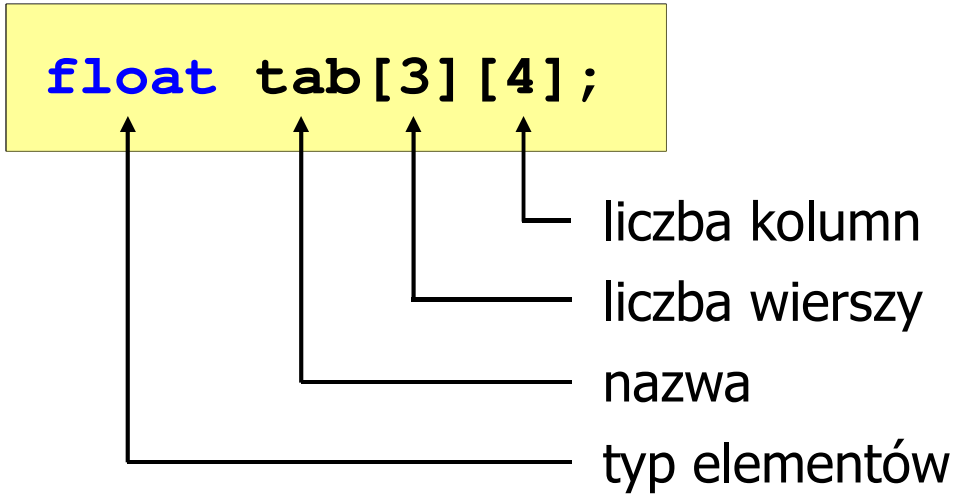
```
Liczba parzystych elementow: 4
```

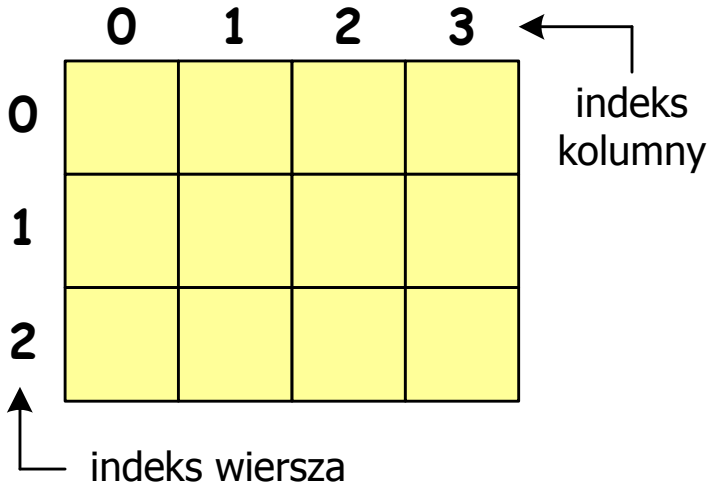
0	1	2	3	4	5	6	7	8	9
7	12	1	16	1	11	14	5	19	8

N = 10

Język C - deklaracja tablica dwuwymiarowej

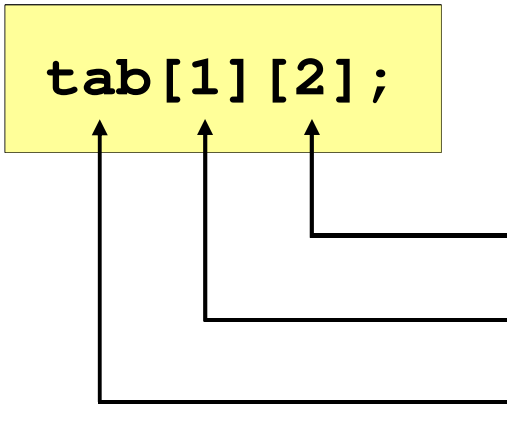
```
float tab[3][4];
```



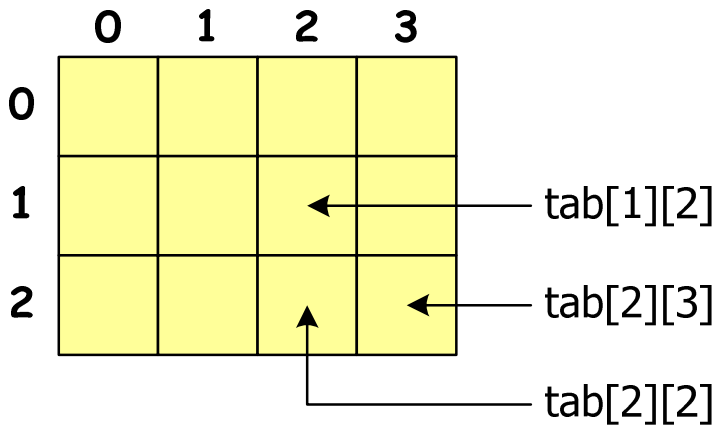


- Rozmiar tablicy (liczb wierszy i kolumn) to wartość:
  - całkowita, dodatnia
  - znana na etapie kompilacji programu
    (stała liczbowa: 5, #define N 5, const int n = 5;)

Język C - odwołania do elementów macierzy



[ ] - 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	5	6

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

	0	1	2
0	1	2	3
1	4	0	0

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

	0	1	2
0	1	0	0
1	4	5	0

Język C - inicjalizacja elementów macierzy

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

wyzerowanie elementów macierzy

	0	1	2
0	0	0	0
1	0	0	0

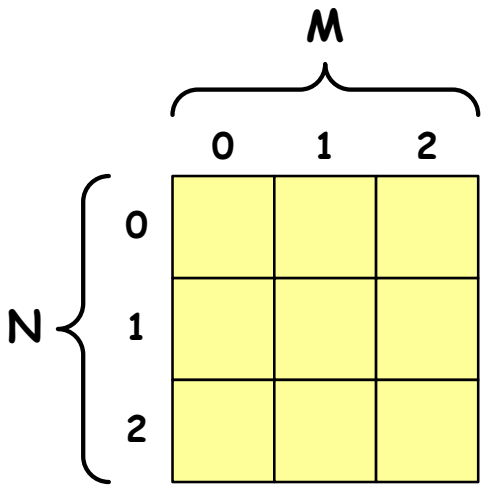
```
int T[][3] = {{1,2,3},{4,5,6}};
```

pominięcie liczby wierszy

	0	1	2
0	1	2	3
1	4	5	6

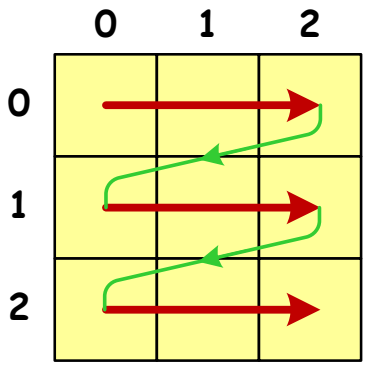
Język C - operacje na macierzy

```
#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;
```

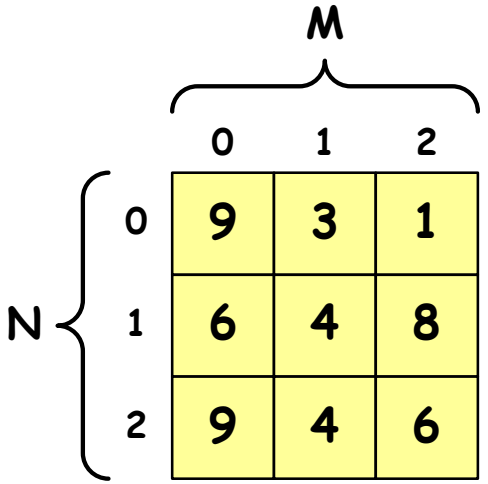


Język C - operacje na macierzy

```
/* 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;
```



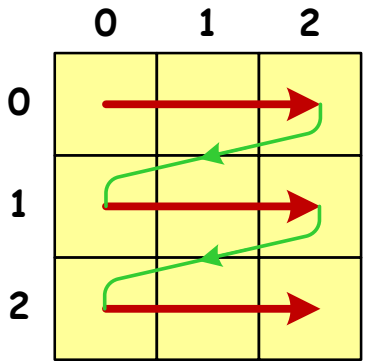
kolejność zapisywania wartości elementów macierzy



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
```

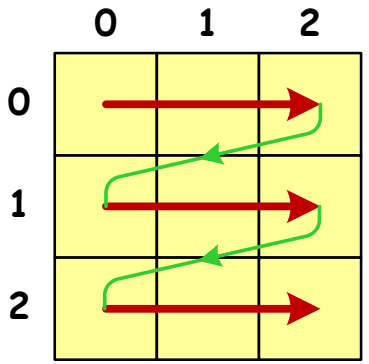


	0	1	2
0	9	3	1
1	6	4	8
2	9	4	6

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);
```

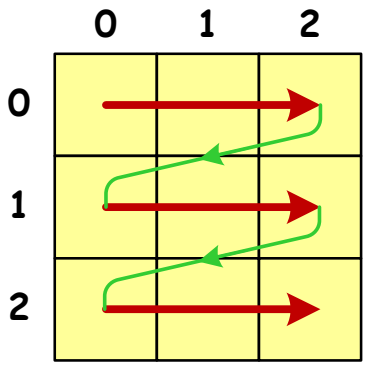
```
Wartosc min: 1
```



	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);
```

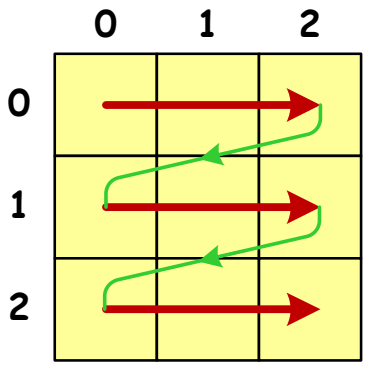


	0	1	2
0	9	3	1
1	6	4	8
2	9	4	6

```
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);
}
```

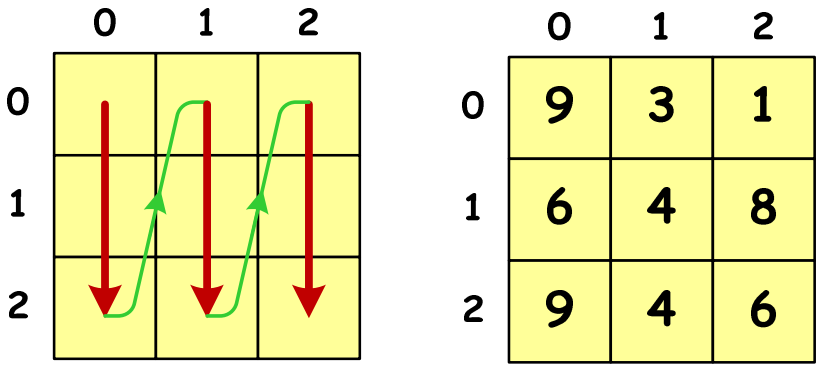


	0	1	2
0	9	3	1
1	6	4	8
2	9	4	6

```
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	9	3	1
1	6	4	8
2	9	4	6

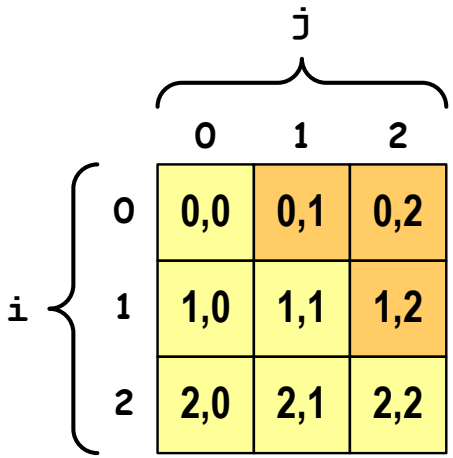
```
Suma kolumny 0 = 24
Suma kolumny 1 = 11
Suma kolumny 2 = 15
```

Język C - operacje na macierzy

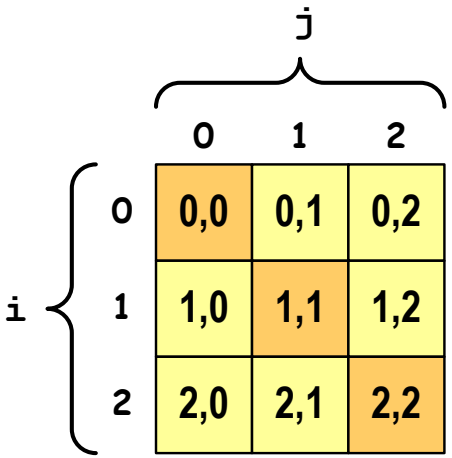
```
/* sumy elementów nad, na i poniżej przekątnej */

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
```

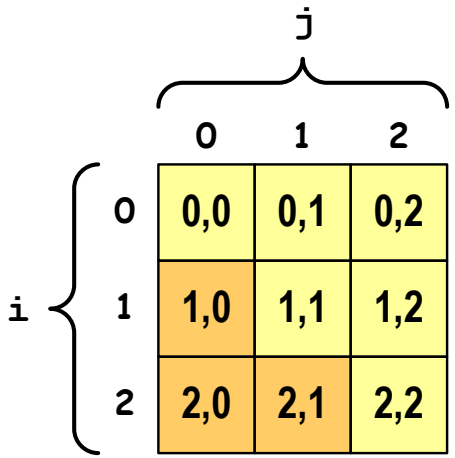
Język C - operacje na macierzy



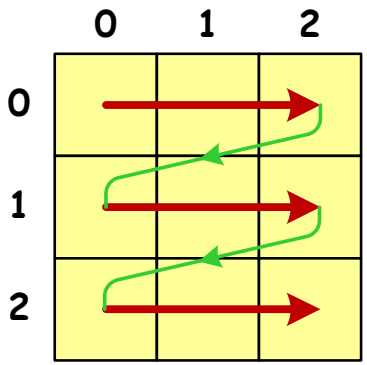
`i < j`

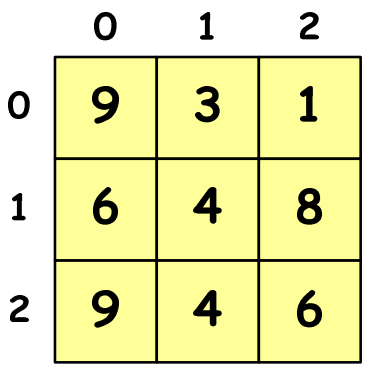


`i = j`



`i > j`





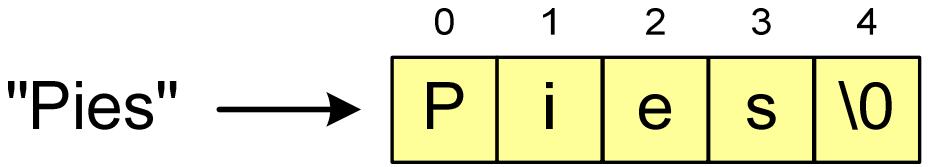

```
Suma nad: 12
Suma na:  19
Suma pod: 19
```

Język C - łańcuchy znaków

- Łańcuch znaków (ciąg znaków, napis, literał łańcuchowy, stała łańcuchowa, C-string) - ciąg złożony z zera lub większej liczby znaków zawartych między znakami cudzysłowu

```
"Pies"
```

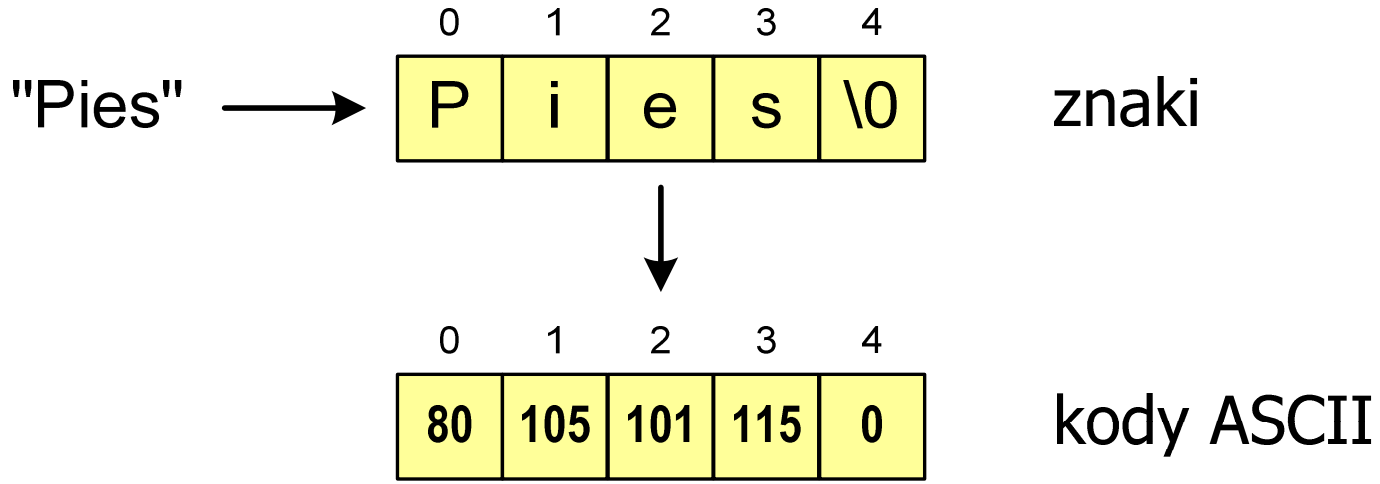
- Implementacja - tablica, której elementami są pojedyncze znaki (typ char)



- Ostatni znak (\0, liczba zero, znak zerowy) oznacza koniec napisu

Język C - łańcuchy znaków

- W rzeczywistości w tablicy zamiast znaków przechowywane są odpowiadające im kody ASCII (czyli liczby)



Język C - deklaracja łańcucha znaków

- Deklaracja zmiennej przechowującej łańcuch znaków

```
char nazwa_zmiennej[rozmiar];
```

Przykład:

```
chat txt[10];
```

- Tablica txt może przechowywać napisy o maksymalnej długości do 9 znaków

Język C - inicjalizacja łańcucha znaków

- Inicjalizacja łańcucha znaków

```
char txt1[10] = "Pies";
char txt2[10] = {'P','i','e','s'};
char txt3[10] = {80,105,101,115};
```

- Pozostałe elementy tablicy otrzymują wartość zero

P	i	e	s	\0	\0	\0	\0	\0	\0

```
char txt4[]   = "Pies";
char *txt5    = "Pies";
```

Język C - inicjalizacja łańcucha znaków

- Inicjalizacja możliwa jest tylko przy deklaracji

```
char txt[10];
txt = "Pies";    /* BŁĄD!!! */
```

- Przypisanie zmiennej txt wartości "Pies" wymaga zastosowania funkcji strcpy() z pliku nagłówkowego string.h

```
char txt[10];
strcpy(txt,"Pies");
```

Język C - stała znakowa

- Stałą znakową tworzy jeden znak ujęty w apostrofy

```
char zn = 'x';
```

- W rzeczywistości stała znakowa jest to liczba całkowita, której wartość odpowiada wartości kodu ASCII reprezentowanego znaku
- Zamiast powyższego kodu można napisać:

```
char zn = 120;
```

- Uwaga:
  - `'x'` - stała znakowa (jeden znak)
  - `"x"` - łańcuch znaków (dwa znaki: x oraz \0)

Język C - stała znakowa

- Niektóre znaki mogą być reprezentowane w stałych znakowych przez sekwencje specjalne, które wyglądają jak dwa znaki, ale reprezentują tylko jeden znak

`'\n'`	- nowy wiersz	`'\\'`	- \ (ang. backslash)
`'\t'`	- tabulator poziomy	`'\''`	- apostrof
`'\v'`	- tabulator pionowy	`'\"'`	- cudzysłów
`'\a'`	- alarm	`'\?'`	- znak zapytania

Język C - wyświetlenie tekstu

- Wyświetlenie tekstu funkcją printf() wymaga specyfikatora %s

```
char napis[15] = "Jan Kowalski";
printf("Osoba: [%s]\n", napis);
```

```
Osoba: [Jan Kowalski]
```

- W specyfikatorze %s: szerokość określa szerokość pola, zaś precyzja - liczbę pierwszych znaków z łańcucha

```
char napis[15] = "Jan Kowalski";
printf("[%10.6s]\n", napis);
```

```
[    Jan Ko]
```

Język C - wyświetlenie tekstu

- Do wyświetlenia tekstu można zastosować funkcję puts()

puts()	int puts(const char *s);

- Funkcja puts() wypisuje na stdout (ekran) zawartość łańcucha znakowego (ciąg znaków zakończony znakiem '\0'), zastępując znak '\0' znakiem '\n'

```
char napis[15] = "Jan Kowalski";
puts(napis);
```

```
Jan Kowalski
```

Język C - wyświetlenie tekstu

- Wyświetlenie znaku funkcją printf() wymaga specyfikatora %c

```
char zn = 'x';
printf("Znak to: [%c]\n", zn);
```

```
Znak to: [x]
```

Język C - wyświetlenie tekstu

- Łańcuch znaków jest zwykłą tablicą - można więc odwoływać się do jej pojedynczych elementów

```
char txt[15] = "Ola ma laptopa";

printf("Znaki: ");
for (int i=0; i<15; i++) printf("%c ",txt[i]);
printf("\n");

printf("Kody:  ");
for (int i=0; i<15; i++) printf("%d ",txt[i]);
printf("\n");
```

```
Znaki: O l a   m a   l a p t o p a
Kody:  79 108 97 32 109 97 32 108 97 112 116 111 112 97 0
```

Język C - wczytanie tekstu

- Do wczytania tekstu funkcją scanf() stosowany jest specyfikator %s

```
char napis[15];
scanf("%s", napis);
```

brak znaku &

- W specyfikatorze formatu %s można podać szerokość

```
char napis[15];
scanf("%10s", napis);
```

- W powyższym przykładzie scanf() zakończy wczytywanie tekstu po pierwszym białym znaku (spacja, tabulacja, enter) lub w momencie pobrania 10 znaków

Język C - wczytanie tekstu

- W przypadku wprowadzenia tekstu "To jest napis", funkcja scanf() zapamięta tylko wyraz "To"
- Zapamiętanie całego wiersza tekstu (do naciśnięcia klawisza Enter) wymaga użycia funkcji gets()

gets()	`char *gets(char *s);`

- Funkcja gets() wprowadza wiersz (ciąg znaków zakończony '\n') ze strumienia stdin (klawiatura) i umieszcza w obszarze pamięci wskazywanym przez wskaźnik s zastępując '\n' znakiem '\0'

```
char napis[15];
gets(napis);
```

Język C - plik nagłówkowy string.h

strcpy()	`char *strcpy(char *s1, const char *s2);`

- Kopiuje łańcuch s2 do łańcucha s1

strlen()	`size_t strlen(const char *s);`

- Zwraca długość łańcucha znaków, nie uwzględnia znaku '\0'

strcat()	`char *strcat(char *s1, const char *s2);`

- Dołącza do łańcucha s1 łańcuch s2

Język C - plik nagłówkowy string.h

strcmp()	`int strcmp(const char *s1, const char *s2);`

- Porównuje łańcuchy s1 i s2 z rozróżnianiem wielkości liter

strcmpi()	`int strcmpi(const char *s1, const char *s2);`

- Porównuje łańcuchy s1 i s2 bez rozróżniania wielkości liter

strchr()	`char *strchr(const char *s, int c);`

- Szuka w łańcuchu s znaku c

Język C - plik nagłówkowy string.h

strlwr()	`char *strlwr(char *s);`

- Zamienia w łańcuchu s wielkie litery na małe

strupr()	`char *strupr(char *s);`

- Zamienia w łańcuchu s małe litery na wielkie

strrev()	`char *strrev(char *s);`

- Odwraca kolejność znaków w łańcuchu s

Język C - plik nagłówkowy string.h (przykład)

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

int main(void)
{
    char napis1[] = "Tekst w buforze", napis2[20];

    printf("napis1: %s \n",napis1);
    int dlugosc = strlen(napis1);
    printf("liczba znakow w napis1: %d \n",dlugosc);
    strcpy(napis2,napis1);
    printf("napis2: %s \n",napis2);
    strrev(napis2);
    printf("napis2 (odwr): %s \n",napis2);

    return 0;
}
```

Język C - plik nagłówkowy string.h (przykład)

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

int main(void)
{
    char napis1[] = "
    printf("napis1: %s \n",napis1);
    int dlugosc = strlen(napis1);
    printf("liczba znakow w napis1: %d \n",dlugosc);
    strcpy(napis2,napis1);
    printf("napis2: %s \n",napis2);
    strrev(napis2);
    printf("napis2 (odwr): %s \n",napis2);

    return 0;
}
```

```
napis1: Tekst w buforze
liczba znakow w napis1: 15
napis2: Tekst w buforze
napis2 (odwr): ezrofub w tskeT
```

Język C - macierz elementów typu char

- Szczególny przypadek tablicy dwuwymiarowej

```
char txt[3][15] = {"Programowanie",
                   "nie jest",
                   "trudne"};
```

- Tablica w pamięci komputera

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	P	r	o	g	r	a	m	o	w	a	n	i	e	\0	\0
1	n	i	e		j	e	s	t	\0	\0	\0	\0	\0	\0	\0
2	t	r	u	d	n	e	\0	\0	\0	\0	\0	\0	\0	\0	\0

Język C - macierz elementów typu char

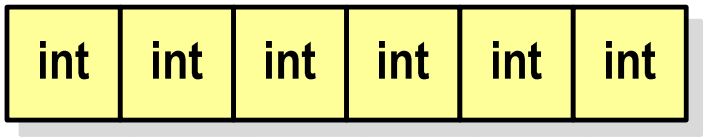
- Używając dwóch indeksów (nr wiersza i nr kolumny) można odwoływać się do jej pojedynczych elementów (znaków)
- Użycie jednego indeksu (numeru wiersza) powoduje potraktowanie całego wiersza jako łańcuch znaków (napisu)

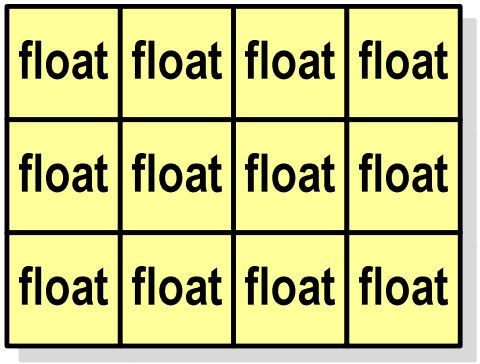
```
char txt[3][15] = {"Programowanie",
                   "nie jest",
                   "trudne"};
printf("%s ",txt[1]);
printf("%s ",txt[2]);
printf("%s ",txt[0]);
```

```
nie jest trudne Programowanie
```

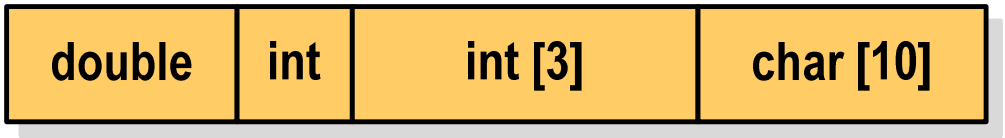
Struktury w języku C

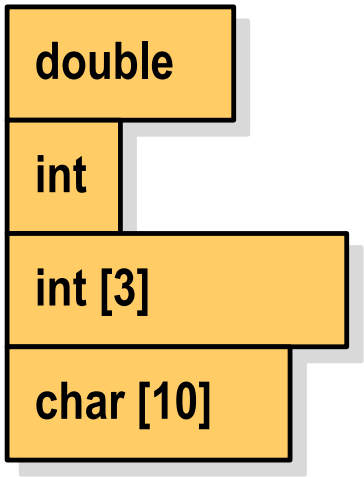
- Tablica - ciągły obszar pamięci zawierający elementy tego samego typu





- Struktura - zestaw elementów różnych typów, zgrupowanych pod jedną nazwą





Deklaracja struktury

```
struct nazwa
{
    opis_pola_1;
    opis_pola_2;
    ...
    opis pola_n;
};
```

```
struct punkt
{
    int x;
    int y;
};
```

- Elementy struktury to pola (dane, komponenty, składowe) struktury
- Deklaracje pól mają taką samą postać jak deklaracje zmiennych
- Deklarując strukturę tworzymy nowy typ danych (struct punkt), którym można posługiwać się tak samo jak każdym innym typem standardowym

Deklaracja struktury

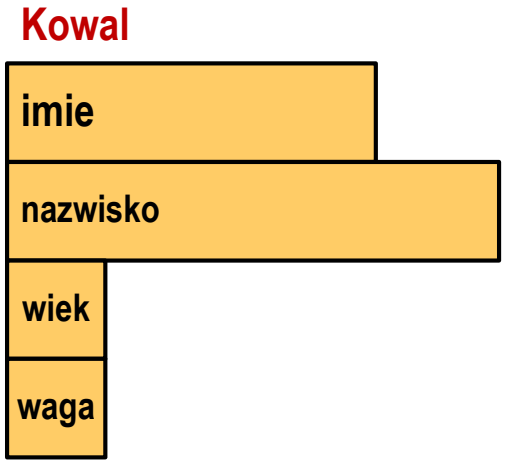
```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};
```

```
struct zesp
{
    float Re, Im;
};
```

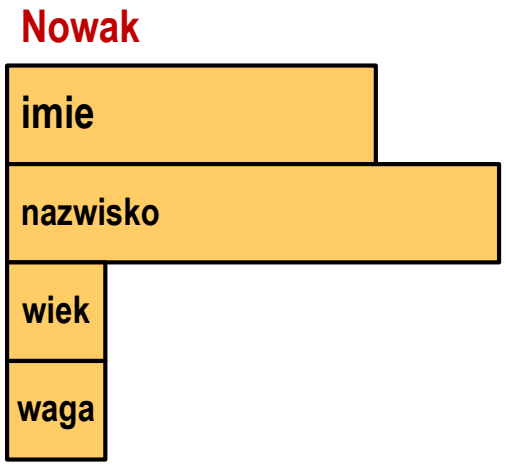
- Deklaracja struktury nie tworzy obiektu (nie przydziela pamięci na pola struktury)
- Zapisanie danych do struktury wymaga zdefiniowania zmiennej strukturalnej

Deklaracja zmiennej strukturalnej

```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
} Kowal, Nowak;
```



- Kowal, Nowak
  - zmienne strukturalne typu struct osoba



Deklaracja zmiennej strukturalnej

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};

int main(void)
{
    struct osoba Kowal;
    struct osoba Nowak;
    ...
    return 0;
}
```

Odwołania do pól struktury

- Dostęp do pól struktury możliwy jest dzięki konstrukcji typu:

```
nazwa_struktury.nazwa_pola
```

- Operator . nazywany jest operatorem bezpośredniego wyboru pola
- Zapisanie wartości 25 do pola wiek zmiennej Nowak ma postać

```
Nowak.wiek = 25;
```

- Wyrażenie Nowak.wiek traktowane jest jak zmienna typu int

```
printf("Wiek: %d\n", Nowak.wiek);
scanf("%d", &Nowak.wiek);
```

Odwołania do pól struktury

- Dostęp do pól struktury możliwy jest dzięki konstrukcji typu:

```
nazwa_struktury.nazwa_pola
```

- Operator . nazywany jest operatorem bezpośredniego wyboru pola
- Zapisanie wartości Jan do pola imie zmiennej Nowak ma postać

```
strcpy(Nowak.imie,"Jan");
```

- Wyrażenie Nowak.imie traktowane jest jak łańcuch znaków

```
printf("Imie: %s\n", Nowak.imie);
gets(Nowak.imie);
```

Struktury - przykład

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int  wiek;
};

int main(void)
{
     struct osoba Nowak;
```

Struktury - przykład

```
    printf("Imie:      ");
    gets(Nowak.imie);

    printf("Nazwisko: ");
    gets(Nowak.nazwisko);

    printf("Wiek:      ");
    scanf("%d",&Nowak.wiek);

    printf("%s %s, wiek: %d\n",Nowak.imie,
            Nowak.nazwisko,Nowak.wiek);

    return 0;
}
```

```
Imie:      Jan
Nazwisko: Nowak
Wiek:      22
Jan Nowak, wiek: 22
```

Inicjalizacja zmiennej strukturalnej

- Inicjalizowane mogą być tylko zmienne strukturalne, nie można inicjalizować pól w deklaracji struktury

```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};

int main(void)
{
    struct osoba Nowak1 = {"Jan","Nowak",25,74};
    ...
}
```

Struktury a operator przypisania (=)

- Struktury tego samego typu można sobie przypisywać (nawet jeśli zawierają tablice)

```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};

int main(void)
{
    struct osoba Nowak1 = {"Jan","Nowak",25,74};
    struct osoba Nowak2;

    Nowak2 = Nowak1;
}
```

operator przypisania

Koniec wykładu nr 3

Dziękuję za uwagę!