

# Informatyka 1 (EZ1F1002)

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

## Wykład nr 10 (22.01.2023)

dr inż. Jarosław Forenc

## Plan wykładu nr 10

- Sieci komputerowe
  - model referencyjny ISO/OSI, model protokołu TCP/IP
- Funkcje w języku C
  - typy funkcji
  - przekazywanie argumentów do funkcji przez wartość i przez wskaźnik
  - przekazywanie wektorów, macierzy i struktur do funkcji

## Model ISO/OSI

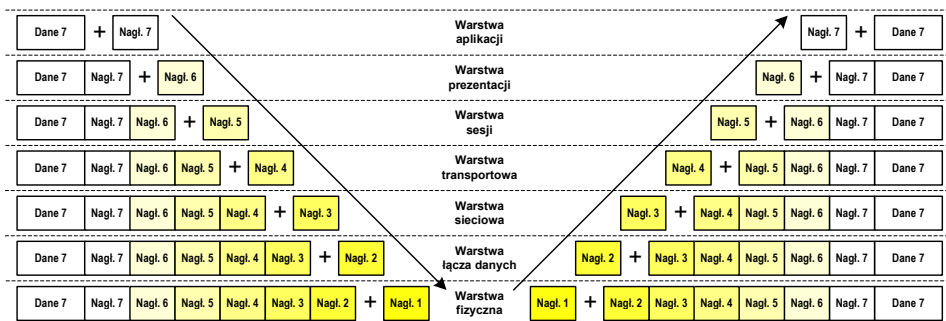
- w latach 70-tych nie istniały ogólne standardy dotyczące sieci komputerowych - każdy producent tworzył własną sieć
- w 1984 roku Międzynarodowa Organizacja Normalizacyjna (ISO) przyjęła model sieciowy, dzięki któremu producenci mogliby opracowywać współpracujące ze sobą rozwiązania sieciowe
- **ISO OSI RM - ISO Open Systems Interconnection Reference Model**
- głównym założeniem modelu jest podział systemów sieciowych na współpracujące ze sobą **7 warstw** (layers)
- struktura tworzona przez warstwy nazywana jest **stosem** protokołu wymiany danych

## Model ISO/OSI



- wierzchołek stosu odpowiada usługom świadczonym bezpośrednio użytkownikowi przez aplikacje sieciowe, zaś dół odpowiada sprzętowi realizującemu transmisję sygnałów
- dane przekazywane są od wierzchołka stosu nadawcy przez kolejne warstwy, aż do warstwy pierwszej, która przesyła je do odbiorcy

## Model ISO/OSI



- przy przechodzeniu do warstwy niższej, warstwa dokleja do otrzymanych przez siebie danych nagłówek z informacjami dla swojego odpowiednika na odległym komputerze (odbiorcy)
- warstwa na odległym komputerze interpretuje nagłówek i jeśli trzeba przekazać dane wyżej - usuwa nagłówek i przekazuje dane dalej

## Model ISO/OSI a model TCP/IP

- w przypadku protokołu TCP/IP tworzącego Internet stosuje się uproszczony model czterowarstwowy

7	Warstwa aplikacji
6	Warstwa prezentacji
5	Warstwa sesji
4	Warstwa transportowa
3	Warstwa sieciowa
2	Warstwa łącza danych
1	Warstwa fizyczna

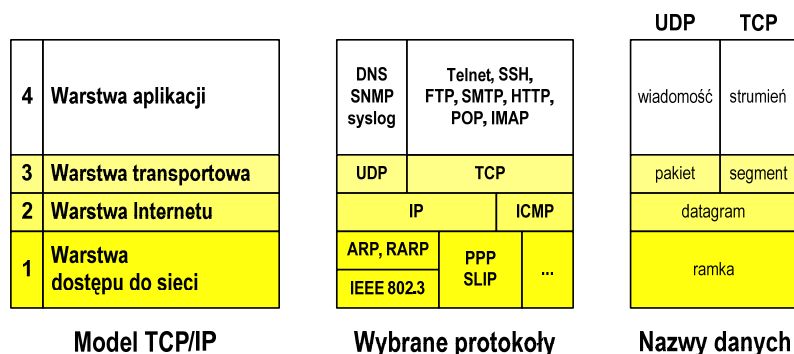
Model ISO/OSI

Warstwa aplikacji	4
Warstwa transportowa	3
Warstwa Internetu	2
Warstwa dostępu do sieci	1

Model TCP/IP

## Model TCP/IP

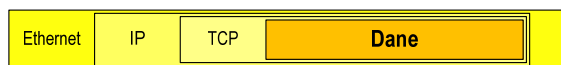
- z poszczególnymi warstwami związanych jest wiele **protokołów**
- **protokół** - zbiór zasad określających format i sposób przesyłania danych



Model TCP/IP

Wybrane protokoły

Nazwy danych



## Warstwa dostępu do sieci

- standard **IEEE 802.3 (Ethernet)** - 1985 r.
- dane przesyłane w postaci ramek Ethernet, format ramki Ethernet II (DIX):

8B	6B	6B	2B	46 - 1500 B	4B
Preambuła	Adres docelowy	Adres źródłowy	Typ	Dane	FCS

- **Preambuła** - naprzemienny ciąg bitów 1 i 0 informujący o ramce
- **Adres docelowy / źródłowy** - 6-bajtowe liczby będące adresami sprzętowymi komunikujących się interfejsów sieciowych (MAC - Media Access Control)

00:23:76:09:41:3B  
 producent karty    numer egzemplarza

FF:FF:FF:FF:FF:FF  
 adres docelowy rozgłoszeniowy

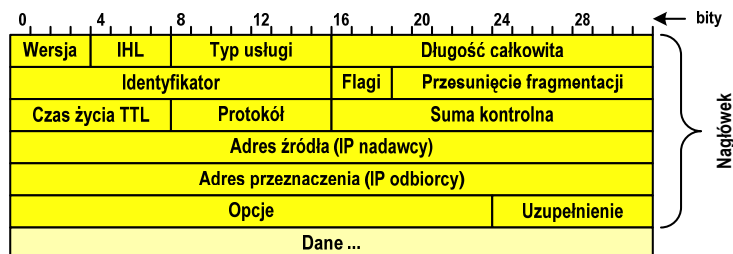
## Warstwa dostępu do sieci

- standard **IEEE 802.3 (Ethernet)** - 1985 r.
- dane przesyłane w postaci ramek Ethernet, format ramki Ethernet II (DIX):

8B	6B	6B	2B	46 - 1500 B	4B
Preambuła	Adres docelowy	Adres źródłowy	Typ	Dane	FCS

- Typ** - numer protokołu warstwy wyższej, która odbierze dane po zakończeniu obróbki przez standard Ethernet
- Dane** - przesyłane dane, jeśli ilość danych jest mniejsza od 46 bajtów, wprowadzane jest uzupełnienie jedynekami (bitowo)
- FCS (Frame Check Sequence)** - 4 bajty kontrolne (CRC - Cyclic Redundancy Check) wygenerowane przez interfejs nadający i sprawdzane przez odbierający

## Warstwa Internetu - datagram IP



- Wersja (Version)** - numer wersji protokołu IP (IPv4, nowsza - IPv6)
- Identyfikator (Identification), Flagi (Flags), Przesunięcie fragmentacji (Fragment offset)** - pola używane w przypadku podziału datagramu na części (fragmenty)
- Adres źródła (Source Address)** - adres IP źródła danych
- Adres przeznaczenia (Destination Address)** - adres IP odbiorcy danych

## Warstwa Internetu

- najważniejsza część Internetu to protokół **IP (Internet Protocol)**:
  - definiuje format i znaczenie pól **datagramu IP**
  - określa schemat adresowania stosowany w Internecie
  - zapewnia wybór trasy przesyłania datagramu (routing)
  - zapewnia podział danych na fragmenty i łączenie ich w całość w przypadku sieci nie akceptujących rozmiaru przenoszonych danych
- cechy protokołu:
  - bezpółłączeniowy** - nie ustanawia połączenia i nie sprawdza gotowości odbiorcy danych
  - niepewny** - nie zapewnia korekcy i wykrywania błędów transmisji

## Warstwa Internetu - adresy IP

- adres IP komputera zajmuje 4 bajty (32-bitowa liczba całkowita)
- najczęściej zapisywany jest w postaci 4 liczb z zakresu od 0 do 255 każda, oddzielonych kropkami, np.

**213.33.95.114**

**11010100.00100001.01011111.01110010**

- adres składa się z dwóch części:
  - identyfikującej daną sieć w Internecie
  - identyfikującej konkretny komputer w tej sieci

## Warstwa Internetu - maska sieci

- do określenia liczby bitów odpowiadających sieci i liczby bitów odpowiadających hostowi stosowana jest **maska sieci**

IP:	212.33.95.114	11010100.00100001.01011111.01110010
Maska:	255.255.255.192	11111111.11111111.11111111.11000000

Adres sieci:	212.33.95.64	11010100.00100001.01011111.01000000
Broadcast:	212.33.95.127	11010100.00100001.01011111.01111111

Pierwszy host:	212.33.95.65	11010100.00100001.01011111.01000001
Ostatni host:	212.33.95.126	11010100.00100001.01011111.01111110

## Warstwa Internetu - adresy IP

- adresy specjalne**

0.0.0.0	- adres sieci dla całego Internetu
255.255.255.255	- adres rozgłoszeniowy dla całego Internetu
127.0.0.1	- adres pętli (loop-back address) - stosowany do komunikacji z lokalnym komputerem (localhost)

- adresy prywatne** (nierutowalne) - nie są przekazywane przez routery

10.0.0.0 – 10.255.255.255	- klasa A
172.16.0.0 – 172.31.255.255	- klasa B
192.168.0.0 – 192.168.255.255	- klasa C

## Warstwa transportowa - porty

- protokoły warstwy transportowej zapewniają dostarczenie danych do **konkretnych aplikacji** (procesów) w odpowiedniej kolejności i formie
- identyfikacja przynależności danej transmisji do procesu odbywa się na podstawie **numeru poru** (liczba 16-bitowa, zakres: 0 ÷ 65535)
- numery portów przydzielane są przez organizację **IANA** (Internet Assigned Numbers Authority):
  - 0 ÷ 1023 - zakres zarezerwowany dla tzw. **dobrze znanych portów** (well-know port number)
  - 1024 ÷ 49151 - porty zarejestrowane (registered)
  - 49152 ÷ 65535 - porty dynamiczne/prywatne (dynamic/private)
- połączenie numeru IP komputera i portu, na którym odbywa się komunikacja, nazywa się **gniazdem** (socket)

## Warstwa transportowa - porty

- wybrane dobrze znane porty:

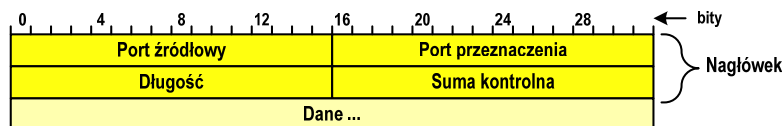
port	protokół
20	FTP (dane)
21	FTP (polecenia)
22	SSH
23	Telnet
25	SMTP (mail)

port	protokół
53	DNS
80	HTTP (www)
110	POP3 (mail)
119	NNTP (news)
143	IMAP (mail)

- w warstwie transportowej funkcjonują dwa podstawowe protokoły:
  - UDP** (User Datagram Protocol)
  - TCP** (Transmission Control Protocol)

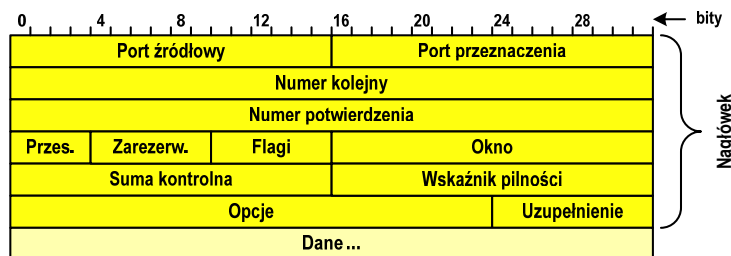
## Warstwa transportowa - protokół UDP

- UDP wykonuje usługę **bezpołączeniowego** dostarczania datagramów:
  - nie ustanawia połączenia
  - nie sprawdza gotowości odbiorcy do odebrania przesyłanych danych
  - nie sprawdza poprawności dostarczenia danych
- jednostką przesyłanych danych jest **pakiet**



- **Port źródłowy (Source port)** - numer portu nadawcy
- **Port przeznaczenia (Destination port)** - numer portu odbiorcy
- **Długość (Length)** - całkowita długość pakietu w bajtach (nagłówek + dane)
- **Suma kontrolna (Checksum)** - tworzona na podstawie nagłówka i danych

## Warstwa Internetu - segment TCP



- **Port źródłowy (Source port)** - numer portu nadawcy
- **Port przeznaczenia (Destination port)** - numer portu odbiorcy
- **Numer kolejny (Sequence number)** - identyfikator określający miejsce segmentu przed fragmentacją
- **Numer potwierdzenia (Acknowledgment number)** - identyfikator będący potwierdzeniem otrzymania danych przez odbiorcę

## Warstwa transportowa - protokoły UDP i TCP

- **UDP** stosowany jest, gdy ilość przesyłanych danych w pakiecie jest niewielka
- pakiet **UDP** zawiera bardzo mało informacji kontrolnych, zatem opłacalne jest jego stosowanie w powiązaniu z aplikacjami samodzielnie dbającymi o kontrolę poprawności transmisji
- **TCP** (Transmission Control Protocol) jest protokołem **niezawodnym** i **połączeniowym**, działa na strumieniach bajtów
- **TCP** sprawdza czy dane zostały dostarczone poprawnie i w określonej kolejności
- jednostką przesyłanych danych stosowaną przez TCP jest **segment**

## Warstwa aplikacji

- zawiera szereg procesów (usług, protokołów) wykorzystywanych przez uruchamiane przez użytkownika aplikacje do przesyłania danych
- większość usług działa w architekturze **klient-serwer** (na odległym komputerze musi być uruchomiony serwer danej usługi)

### DNS (Domain Name System)

- świadczy usługi zamieniania (rozwiązywania) nazwy komputera na jego adres IP

`we.pb.edu.pl` → `213.33.95.2`

- wykorzystuje port o numerze 53

## Funkcje w języku C

```
#include <stdio.h> /* przekatna kwadratu */  
#include <math.h>
```

```
float przekatna(float bok) ← definicja funkcji  
{  
    float wynik;  
    wynik = bok * sqrt(2.0f);  
    return wynik;  
}
```

```
int main(void) ← definicja funkcji  
{  
    float a = 10.0f, d;  
    d = przekatna(a);  
    printf("Bok = %g, przekatna = %g\n", a, d);  
    return 0;  
}
```

## Typy funkcji (1)

- Dotychczas prezentowane funkcje miały argumenty i zwracały wartości
- Struktura i wywołanie takiej funkcji ma następującą postać

```
typ nazwa (parametry)  
{  
    instrukcje;  
    return wartość;  
}
```

```
typ zm;  
zm = nazwa(argumenty);
```

- Można zdefiniować także funkcje, które nie mają argumentów i/lub nie zwracają żadnej wartości

## Typy funkcji (2)

- Funkcja bez argumentów i nie zwracająca wartości:
  - w nagłówku funkcji, typ zwracanej wartości to **void**
  - zamiast parametrów, podaje się słowo **void** lub nie wpisuje się nic
  - jeśli występuje **return**, to nie może po nim znajdować się żadna wartość
  - jeśli **return** nie występuje, to funkcja kończy się po wykonaniu wszystkich instrukcji
- Struktura funkcji:

```
void nazwa(void)  
{  
    instrukcje;  
    return;  
}
```

```
void nazwa()  
{  
    instrukcje;  
    return;  
}
```

## Typy funkcji (2)

- Funkcja bez argumentów i nie zwracająca wartości:
  - w nagłówku funkcji, typ zwracanej wartości to **void**
  - zamiast parametrów, podaje się słowo **void** lub nie wpisuje się nic
  - jeśli występuje **return**, to nie może po nim znajdować się żadna wartość
  - jeśli **return** nie występuje, to funkcja kończy się po wykonaniu wszystkich instrukcji
- Struktura funkcji:

```
void nazwa(void)  
{  
    instrukcje;  
}
```

```
void nazwa()  
{  
    instrukcje;  
}
```

- Wywołanie funkcji: `nazwa ();`

## Typy funkcji (2) - przykład

```
#include <stdio.h>

void drukuj_linie(void)
{
    printf("-----\n");
}

int main(void)
{
    drukuj_linie();
    printf("Funkcje nie sa trudne!\n");
    drukuj_linie();

    return 0;
}
```

```
-----
Funkcje nie sa trudne!
-----
```

## Typy funkcji (3)

- Funkcja z argumentami i nie zwracająca wartości:
  - w nagłówku funkcji, typ zwracanej wartości to **void**
  - jeśli występuje **return**, to nie może po nim znajdować się żadna wartość
  - jeśli **return** nie występuje, to funkcja kończy się po wykonaniu wszystkich instrukcji
- Struktura funkcji:

```
void nazwa (parametry)
{
    instrukcje;
    return;
}
```

```
void nazwa (parametry)
{
    instrukcje;
}
```

- Wywołanie funkcji:

```
nazwa (argumenty);
```

## Typy funkcji (3) - przykład

```
#include <stdio.h>

void drukuj_dane(char *imie, char *nazwisko, int wiek)
{
    printf("Imie:           %s\n", imie);
    printf("Nazwisko:        %s\n", nazwisko);
    printf("Wiek:              %d\n", wiek);
    printf("Rok urodzenia:    %d\n\n", 2022-wiek);
}

int main(void)
{
    drukuj_dane("Jan", "Kowalski", 24);
    drukuj_dane("Barbara", "Nowak", 29);

    return 0;
}
```

## Typy funkcji (3) - przykład

```
#include <stdio.h>

void drukuj_dane(char *imie,
{
    printf("Imie:           %s\n", imie);
    printf("Nazwisko:        %s\n", nazwisko);
    printf("Wiek:              %d\n", wiek);
    printf("Rok urodzenia:    %d\n\n", 2022-wiek);
}

int main(void)
{
    drukuj_dane("Jan", "Kowalski", 24);
    drukuj_dane("Barbara", "Nowak", 29);

    return 0;
}
```

```
Imie:           Jan
Nazwisko:       Kowalski
Wiek:           24
Rok urodzenia: 1998

Imie:           Barbara
Nazwisko:       Nowak
Wiek:           29
Rok urodzenia: 1993
```

## Typy funkcji (4)

- Funkcja bez argumentów i zwracająca wartość:
  - zamiast parametrów, podaje się słowo **void** lub nie wpisuje się nic
  - typ zwracanej wartości musi być zgodny z typem w nagłówku funkcji
- Struktura funkcji:

```
typ nazwa(void)
{
    instrukcje;
    return wartość;
}
```

```
typ nazwa()
{
    instrukcje;
    return wartość;
}
```

- Wywołanie funkcji:

```
typ zm;
zm = nazwa();
```

## Typy funkcji (4) - przykład

W roku jest: 31536000 sekund

```
#include <stdio.h>

int liczba_sekund_rok(void)
{
    return (365 * 24 * 60 * 60);
}

int main(void)
{
    int wynik;

    wynik = liczba_sekund_rok();
    printf("W roku jest: %d sekund\n", wynik);

    return 0;
}
```

## Przekazywanie argumentów do funkcji

- Przekazywanie argumentów przez **wartość**:
  - po wywołaniu funkcji tworzone są lokalne kopie zmiennych skojarzonych z jej argumentami
  - w funkcji widoczne są one pod postacią parametrów funkcji
  - parametry te mogą być traktowane jak lokalne zmienne, którym przypisano początkową wartość
- Przekazywanie argumentów przez **wskaźnik**:
  - do funkcji przekazywane są adresy zmiennych będących jej argumentami
  - wszystkie operacje wykonywane w funkcji na takich argumentach będą odnosiły się do zmiennych z funkcji wywołującej

## Przekazywanie argumentów przez wartość

```
#include <stdio.h>

void fun(int a)
{
    a = 10;
    printf("fun: a = %d\n", a);
}

int main(void)
{
    int a = 20;

    fun(a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()



## Przekazywanie argumentów przez wartość

```
#include <stdio.h>

void fun(int a)
{
    a = 10;
    printf("fun: a = %d\n", a);
}

int main(void)
{
    int a = 20;

    fun(a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()
a	0x0024FAF8	20	fun()

## Przekazywanie argumentów przez wartość

```
#include <stdio.h>

void fun(int a)
{
    a = 10;
    printf("fun: a = %d\n", a);
}

int main(void)
{
    int a = 20;

    fun(a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()
a	0x0024FAF8	10	fun()

fun: a = 10

## Przekazywanie argumentów przez wartość

```
#include <stdio.h>

void fun(int a)
{
    a = 10;
    printf("fun: a = %d\n", a);
}

int main(void)
{
    int a = 20;

    fun(a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()

fun: a = 10  
main: a = 20

## Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

void fun(int *a)
{
    *a = 10;
    printf("fun: a = %d\n", *a);
}

int main(void)
{
    int a = 20;

    fun(&a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()

## Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

void fun(int *a)
{
    *a = 10;
    printf("fun: a = %d\n", *a);
}

int main(void)
{
    int a = 20;

    fun(&a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()
a	0x0024FAF8	0x0024FBDC	fun()

## Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

void fun(int *a)
{
    *a = 10;
    printf("fun: a = %d\n", *a);
}

int main(void)
{
    int a = 20;

    fun(&a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	10	main()
a	0x0024FAF8	0x0024FBDC	fun()

fun: a = 10

## Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

void fun(int *a)
{
    *a = 10;
    printf("fun: a = %d\n", *a);
}

int main(void)
{
    int a = 20;

    fun(&a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	10	main()

fun: a = 10  
main: a = 10

## Parametry funkcji - wektory

- Wektory przekazywane są do funkcji przez wskaźnik
- Nie jest tworzona kopia tablicy, a wszystkie operacje na jej elementach odnoszą się do tablicy z funkcji wywołującej
- W nagłówku funkcji podaje się typ elementów tablicy, jej nazwę oraz nawiasy kwadratowe z liczbą elementów tablicy lub same nawiasy kwadratowe

```
void fun(int tab[5])
{
    ...
}
```

```
void fun(int tab[])
{
    ...
}
```

- W wywołaniu funkcji podaje się tylko jej nazwę (bez nawiasów kwadratowych)

```
fun(tab);
```

## Parametry funkcji - wektory (przykład)

```
#include <stdio.h>

void drukuj(int tab[])
{
    for (int i=0; i<5; i++)
        printf("%3d", tab[i]);
    printf("\n");
}

void zeruj(int tab[5])
{
    for (int i=0; i<5; i++)
        tab[i] = 0;
}
```

```
float srednia(int tab[])
{
    float sr = 0;
    int suma = 0;

    for (int i=0; i<5; i++)
        suma = suma + tab[i];

    sr = (float)suma / 5;

    return sr;
}
```

## Parametry funkcji - wektory (przykład)

```
int main(void)
{
    int tab[5] = {1,2,3,4,5};
    float sred;

    drukuj(tab);

    sred = srednia(tab);
    printf("Srednia elementow: %g\n", sred);
    printf("Srednia elementow: %g\n", srednia(tab));

    zeruj(tab);
    drukuj(tab);

    return 0;
}
```

```
1 2 3 4 5
srednia elementow: 3
srednia elementow: 3
0 0 0 0 0
```

## Parametry funkcji - macierze

- Macierze przekazywane są do funkcji przez wskaźnik
- W nagłówku funkcji podaje się typ elementów tablicy, jej nazwę oraz w nawiasach kwadratowych liczbę wierszy i kolumn lub tylko liczbę kolumn

```
void fun(int tab[2][3])
{
    ...
}
```

```
void fun(int tab[][3])
{
    ...
}
```

- W wywołaniu funkcji podaje się tylko jej nazwę (bez nawiasów kwadratowych)

```
fun(tab);
```

## Parametry funkcji - macierze (przykład)

```
#include <stdio.h>

void zero(int tab[][3])
{
    for (int i=0; i<2; i++)
        for (int j=0; j<3; j++)
            tab[i][j] = 0;
}

void drukuj(int tab[2][3])
{
    for (int i=0; i<2; i++)
    {
        for (int j=0; j<3; j++)
            printf("%3d", tab[i][j]);
        printf("\n");
    }
}
```

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

    drukuj(tab);
    zero(tab);
    printf("\n");
    drukuj(tab);

    return 0;
}
```

## Parametry funkcji - macierze (przykład)

```
#include <stdio.h>

void zero(int tab[][3])
{
    for (int i=0; i<2; i++)
        for (int j=0; j<3; j++)
            tab[i][j] = 0;
}

void drukuj(int tab[2][3])
{
    for (int i=0; i<2; i++)
    {
        for (int j=0; j<3; j++)
            printf("%3d", tab[i][j]);
        printf("\n");
    }
}
```

```
int main
{
    int t
    {
        1 2 3
        4 5 6
    }
    druku
    zero(
    printf("\n");
    drukuj(tab);
    return 0;
}
```

```
1 2 3
4 5 6
0 0 0
0 0 0
```

## Parametry funkcji - struktury

- Struktury przekazywane są do funkcji przez wartość (nawet jeśli daną składową jest tablica)

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

struct pkt
{
    float x, y;
};

float odl(struct pkt pkt1, struct pkt pkt2)
{
    return sqrt(pow(pkt2.x-pkt1.x, 2)+
                pow(pkt2.y-pkt1.y, 2));
}
```

## Parametry funkcji - struktury (przykład)

```
int main(void)
{
    struct pkt p1 = {2,3};
    struct pkt p2 = {-2,1};
    float wynik;

    wynik = odl(p1,p2);

    printf("Punkt nr 1: (%g,%g)\n",p1.x,p1.y);
    printf("Punkt nr 2: (%g,%g)\n",p2.x,p2.y);
    printf("Odleglosc = %g\n",wynik);

    return 0;
}
```

```
Punkt nr 1: (2,3)
Punkt nr 2: (-2,1)
Odleglosc = 4.47214
```

## Koniec wykładu nr 10

Dziękuję za uwagę!