

# Informatyka 1 (EZ1F1002)

---

Politechnika Białostocka - Wydział Elektryczny  
Elektrotechnika, semestr I, studia niestacjonarne I stopnia  
Rok akademicki 2024/2025

**Wykład nr 7 (06.12.2024)**

dr inż. Jarosław Forenc

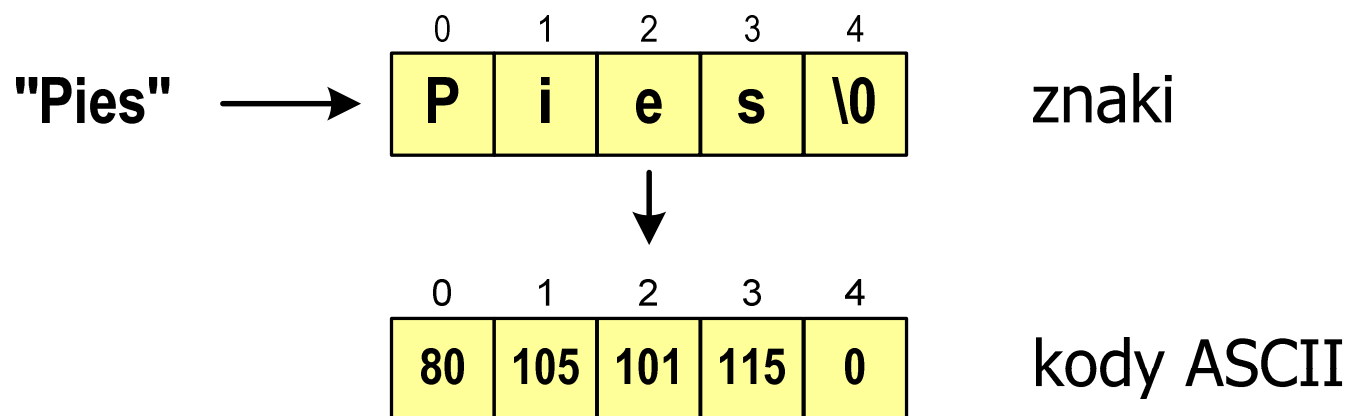
# Plan wykładu nr 7

- Łańcuchy znaków
  - plik nagłówkowy string.h
  - macierz elementów typu char
- Struktury
  - deklaracja struktury i zmiennej strukturalnej
  - odwołania do pól struktury
  - inicjalizacja zmiennej strukturalnej
  - złożone deklaracje struktur
- Pola bitowe, unie

## Język C - łańcuchy znaków

- **łańcuch znaków** - ciąg złożony z zera lub większej liczby znaków zawartych między znakami cudzysłowu

"Pies"



- Deklaracja i inicjalizacja łańcucha znaków

```
char txt[10] = "Pies";
```

## 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

`strncmp()`

```
int strncmp(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)

```
#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[] = "Tekst w buforze";

    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
```



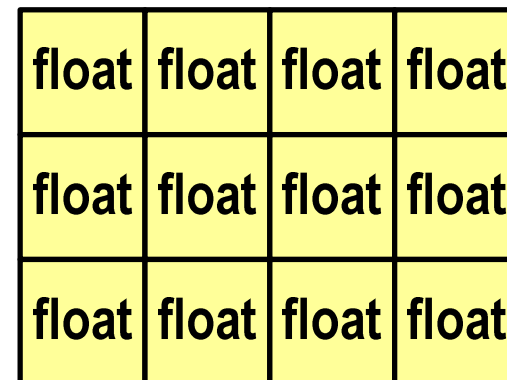
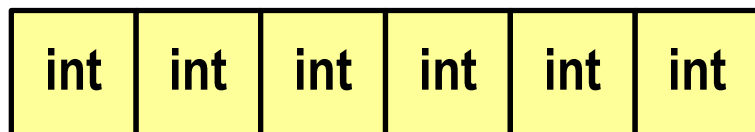




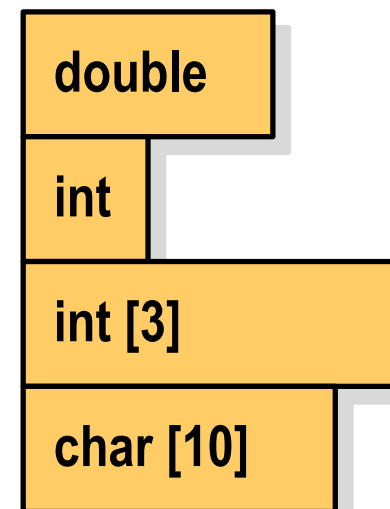


# Struktury w języku C

- **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

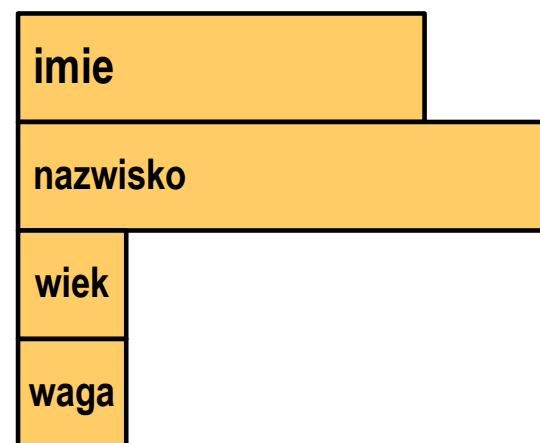
```
#include <stdio.h>

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

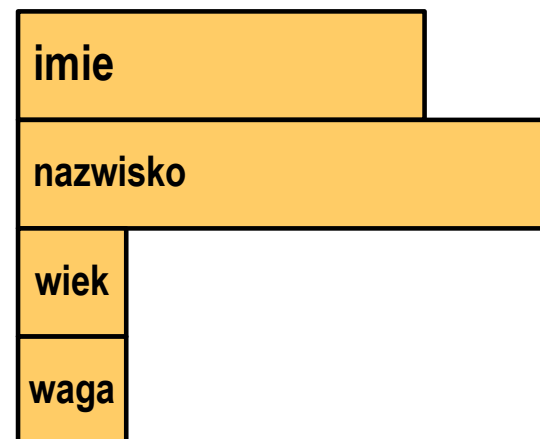
int main(void)
{
    struct osoba Nowak ;
    ...
}
```

- **Kowal, Nowak** - zmienne typu **struct osoba**

**Kowal**



**Nowak**



## Odwołania do pól struktury

- Dostęp do pól struktury możliwy jest dzięki konstrukcji typu:

```
nazwa_zmiennej_strukturalnej.nazwa_pola
```

- Operator `.` nazywany jest **operatorem bezpośredniego wyboru pola**
- Zapisanie wartości do pól zmiennej **Nowak** ma postać

```
Nowak.wiek = 25;  
strcpy(Nowak.imie, "Jan");
```

- Wyrażenie **Nowak.wiek** traktowane jest jak zmienna typu **int**, zaś wyrażenie **Nowak.imie** traktowane jest jak łańcuch znaków

```
printf("%s - wiek %d\n", Nowak.imie, Nowak.wiek);  
scanf("%d", &Nowak.wiek);  
gets(Nowak.imie);
```



## Struktury - przykład (osoba)

```
#include <stdio.h>

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

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

## Struktury - przykład (osoba)

```
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
```

## Struktury w języku C

- **Inicjalizacja** może dotyczyć tylko zmiennych strukturalnych, nie można inicjalizować pól w deklaracji struktury

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

```
struct osoba Nowak = {"Jan", "Nowak", 25, 74};
```

- Do zmiennych strukturalnych można stosować **operator =**

```
struct osoba Kowal = {"Ewa", "Kowal", 21, 54};
struct osoba Kowal1;
Kowal1 = Kowal;
```

# Struktury w języku C

```
#include <stdio.h>

struct date
{
    int day;
    int month;
    int year;
} day1;

int main(void)
{
    struct date day2 = {19, 11, 2018};
}
```

day1

day	?
month	?
year	?

day2

day	19
month	11
year	2018

# Struktury w języku C

```
    day1.day = 1;
    day1.month = 9;
    day1.year = 2018;

    printf("Date1: %02d-%02d-%4d\n",
           day1.day, day1.month, day1.year);
    printf("Date2: %02d-%02d-%4d\n",
           day2.day, day2.month, day2.year);

    return 0;
}
```

```
Date1: 01-09-2018
Date2: 19-11-2018
```

day1

day	1
month	9
year	2018

day2

day	19
month	11
year	2018

## Złożone deklaracje struktur

```
struct punkt  
{  
    int x;  
    int y;  
} tab[3];
```

tab

0	x	y
1	x	y
2	x	y

```
tab[0].x = 10;  
tab[0].y = 20;  
tab[1].x = 15;  
...
```

```
struct trojkat  
{  
    int nr;  
    struct punkt A, B, C;  
} Tr1;
```

Tr1

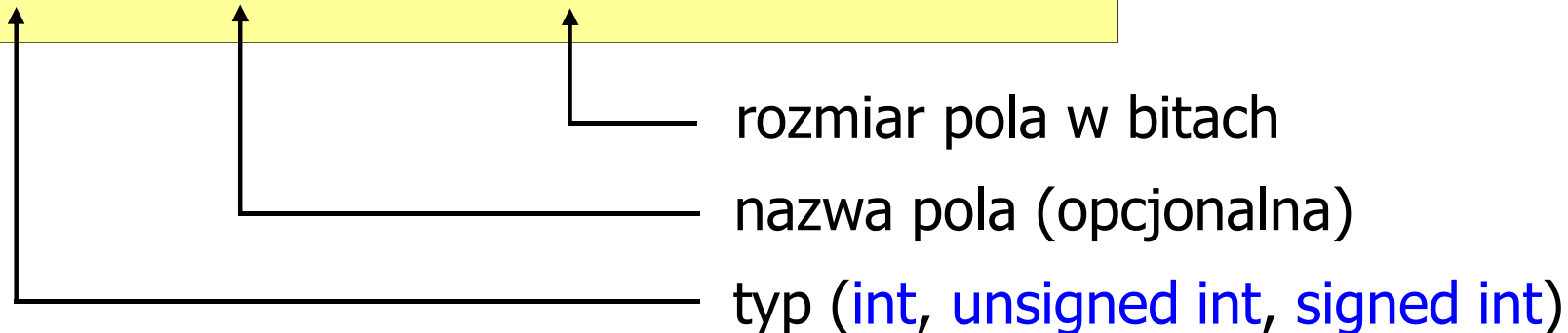
nr		
A	x	y
B	x	y
C	x	y

```
Tr1.nr = 1;  
Tr1.A.x = 10;  
Tr1.A.y = 20;  
Tr1.B.x = 15;  
...
```

## Pola bitowe

- Umożliwiają dostęp do pojedynczych bitów oraz przechowywanie małych wartości zajmujących pojedyncze bity
- Pola bitowe deklarowane są wewnątrz struktur

```
typ id_pola : wielkość_pola;
```



- Wartości zapisane w polach traktowane są jak liczby całkowite
- Zakres wartości pól wynika z `wielkości_pola`

## Pola bitowe

```
struct Bits
{
    unsigned int a : 4;    /* zakres: 0...15 */
    unsigned int b : 2;    /* zakres: 0...3 */
    unsigned int   : 4;
    unsigned int c : 6;    /* zakres: 0...63 */
};
```

- Dostęp do pól bitowych odbywa się na takiej samej zasadzie jak do normalnych pól struktury

```
struct Bits dane;

dane.a = 10;
dane.b = 3;
```



## Pola bitowe

```
struct Bits
{
    unsigned int a : 4;    /* zakres: 0...15 */
    unsigned int b : 2;    /* zakres: 0...3 */
    unsigned int   : 4;
    unsigned int c : 6;    /* zakres: 0...63 */
};
```

- Jeśli pole nie ma nazwy, to nie można się do niego odwoływać
- Pola bitowe nie mają adresów
  - nie można wobec pola bitowego stosować operatora **&** (adres)
  - nie można polu bitowemu nadać wartości funkcją **scanf()**

## Pola bitowe - przykład

```
struct Flags_8086
{
    unsigned int CF : 1;    /* Carry Flag */
    unsigned int   : 1;
    unsigned int PF : 1;    /* Parity Flag */
    unsigned int   : 1;
    unsigned int AF : 1;    /* Auxiliary - Carry Flag */
    unsigned int   : 1;
    unsigned int ZF : 1;    /* Zero Flag */
    unsigned int SF : 1;    /* Signum Flag */
    unsigned int TF : 1;    /* Trap Flag */
    unsigned int IF : 1;    /* Interrupt Flag */
    unsigned int DF : 1;    /* Direction Flag */
    unsigned int OF : 1;    /* Overflow Flag */
};
```

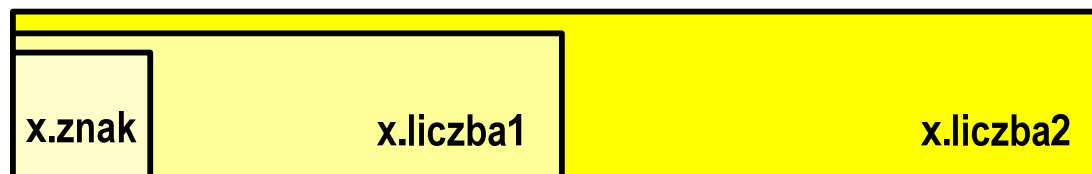
# Unie

- Specjalny rodzaj struktury umożliwiający przechowywanie danych różnych typów w **tym samym obszarze pamięci**
- Do przechowywania wartości w unii należy zadeklarować zmienną

```
union zbior x;
```

```
union zbior  
{  
    char    znak;  
    int     liczba1;  
    double  liczba2;  
};
```

- Zmienna **x** może przechowywać wartość typu **char** lub typu **int** lub typu **double**, ale tylko jedną z nich w danym momencie



- Rozmiar unii wyznaczany jest przez rozmiar największego jej pola

# Unie

- Dostęp do pól unii jest taki sam jak do pól struktury

```
union zbior x;  
x.znak = 'a';  
x.liczba2 = 12.15;
```

```
union zbior  
{  
    char znak;  
    int liczba1;  
    double liczba2;  
};
```

- Unię można zainicjować jedynie wartością o typie jej pierwszej składowej
- Unie tego samego typu można sobie przypisywać

```
union zbior x = {'a'};  
union zbior y;  
y = x;
```

Koniec wykładu nr 7

Dziękuję za uwagę!