# Informatyka 2 (EZ1E3012)

Politechnika Białostocka - Wydział Elektryczny

Elektrotechnika, semestr III, studia niestacjonarne I stopnia

Rok akademicki 2021/2022

## Pracownia nr 3 (23.10.2021)

dr inż. Jarosław Forenc



## 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;
    ...
};
```

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

- Elementy struktury to pola (dane, komponenty, składowe) struktury



## Deklaracja struktury

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

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

- Deklarując strukturę tworzymy nowy typ danych (struct osoba, struct zesp), którym można posługiwać się tak samo jak każdym innym typem standardowym
- Deklaracja struktury nie tworzy obiektu (nie przydziela pamięci)
- Zapisanie danych do struktury wymaga zdefiniowania zmiennej strukturalnej

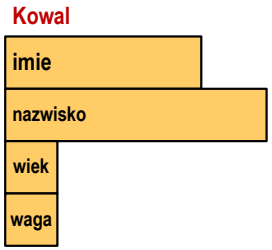


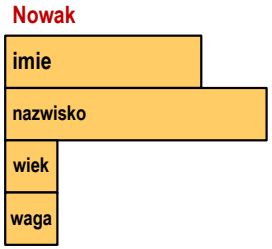
## Deklaracja zmiennej strukturalnej

```
#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

# 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 do pól zmiennej Nowak ma postać

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

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

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



# Odwołania do pól struktury

- Gdy zmienna strukturalna jest wskaźnikiem, to do odwołania do pola struktury używamy operatora pośredniego wyboru pola (->)

```
wskaźnik_do_struktury -> nazwa_pola
```

```
struct osoba Nowak, *Nowak1;
Nowak1 = &Nowak;
Nowak1 -> wiek = 25;
/* lub */
(*Nowak1).wiek = 25;
```

- W ostatnim zapisie nawiasy są konieczne, gdyż operator . ma wyższy priorytet niż operator *



# 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};
```

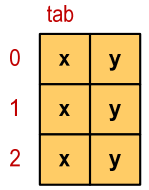
- Do zmiennych strukturalnych można stosować operator przypisania (=)

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



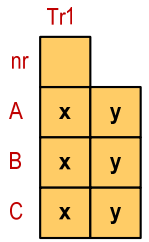
# Złożone deklaracje struktur

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




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

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




```
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;
```

- rozmiar pola w bitach
- nazwa pola (opcjonalna)
- typ (int, unsigned int, signed int)

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



# Pola bitowe

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

- Wartości zapisane w polach traktowane są jak liczby całkowite
- Zakres wartości pól wynika z wielkości_pola
- Dostęp do pól bitowych odbywa się na takiej samej zasadzie jak do normalnych pól struktury

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

- Jeśli pole nie ma nazwy, to nie można się do niego odwoływać

# 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;
```