# Podstawy programowania.

# Wykład 9

# Złożone typy danych: struktury

# Struktury

- Struktury są sposobem na przechowywanie w ramach pojedynczej zmiennej zestawu zmiennych różnych typów
- Struktury, zwane także rekordami, istnieją w wielu językach dla ułatwienia manipulowania złożonymi zestawami danych
- Pojedyncza zmienna będąca strukturą oznacza obszar pamięci, w którym istnieje szereg zmiennych
  - słowem kluczowym do definiowania struktur jest `struct`
  - po słowie `struct`, w nawiasie klamrowym, umieszczone są składowe struktury (pola struktury):
    - `struct{.....}` - definiuje typ
    - `struct{.....} zm_str` – definiuje pojedynczą zmienna takiego typu
  - definicja struktury określa z jakich zmiennych jest złożona, np.

```
struct {  // początek definicji (deklaracji)
    int liczba_lat;      // składowa będąca liczbą int
    char nazwa[25]; // składowa będąca tablicą znaków
}
```

# Struktury

- Część definicji struktury zawierająca jej składowe może mieć własną nazwę:

```
struct punkt_2D { // struktura definiuje punkt w przestrzeni 2D
    double wsp_x; // współrzędna x punktu
    double wsp_y; // współrzędna y punktu
};

struct punkt_2D punkt2d_1, punkt2d_2; // definicja dwóch zmiennych
                                      // typu: "struct punkt_2D"
```

- Składowe struktur mogą mieć takie same nazwy jak składowe innych struktur i zwykłe zmienne:

```
struct punkt_3D { double wsp_x; double wsp_y; double wsp_z; };

struct punkt_3D punkt3d_1, punkt3d_2; // definicja dwóch zmiennych
                                      // typu: "struct punkt_3D"
```

# Struktury

- Zmienna będąca strukturą traktowana jest podobnie jak zmienne innych typów:
  - zmienna może być inicjowana (stałymi wartościami)

    ```
    struct punkt_2D p_srodek = {1.0, 2.0 };
    ```
  - wartość zmiennej może zostać przypisana (zawartość jest kopiowana)

    ```
    struct punkt_2D punkt_1 = p_srodek;
    ```
  - można pobrać adres (początku) zmiennej

    ```
    printf("adres struktury %lu\n", &punkt_1);
    ```
  - zmienna może być przesłana jako argument funkcji

    ```
    void wydrukuj_dane_kola( struct punkt_2D srodek, double promien);
    wydrukuj_dane_kola( p_srodek, 25.0 ); // przesłanie przez wartość
    ```
  - zmienna może zostać zwrócona jako wartość wynikowa funkcji

    ```
    struct punkt_2D punkt_symetryczny( struct punkt_2D );
    ```
    - jest to drugi z możliwych sposobów na przekazanie większej niż 1 liczby wartości z funkcji

      ```
      struct punkt_2D punkt_2 = punkt_symetryczny( punkt_1 );
      ```

# Struktury

- Słowo struct może zostać wyeliminowane z definicji zmiennych poprzez nadanie nowej nazwy utworzonemu typowi
  - służy do tego określenie typedef , które definiuje nową nazwę dla typu
    - typedef może być używane dla dowolnego typu – wbudowanego lub zdefiniowanego przez użytkownika

```
typedef char* Napis; // nazwy char* i Napis znaczą odtąd to samo
Napis wczytana_linia; // definicja zmiennej będącej napisem
```

    - w przypadku zastosowania dla typów utworzonych, typedef pozwala uprościć definicje (deklaracje) zmiennych

```
typedef struct {  // definicja nowego typu będącego strukturą,
                  // określającego punkt w przestrzeni 2D
    double  wsp_x; // współrzędna x punktu
    double  wsp_y; // współrzędna y punktu
} punkt_2D_t;
punkt_2D_t p_srodek = {1.0, 2.0 };
```

  - uwaga: typedef nie tworzy typu lecz nową nazwę dla typu

# Struktury

- Dostęp do pola struktury jako zmiennej odbywa się za pomocą operatora . (kropka)

```
punkt_2D_t p_srodek;
p_srodek.wsp_x = 1.0; p_srodek.wsp_y = 2.0;
printf("wsp x punktu %lf\n", p_srodek.wsp_x);
```

- Przypadki:
  - jeśli elementem struktury jest tablica:

```
struct osoba{ char imie[100]; char nazwisko[100]};
struct osoba zwykly_Polak = { "Jan", "Kowalski" };
char inicjal = zwykly_Polak.imie[0];
```

  - jeśli elementem struktury jest inna struktura

```
struct odcinek { punkt_2D_t poczatek; punkt_2D_t koniec; }
struct odcinek odc_1 = { {0.0, 0.0}, {1.0, 1.0} }
double pocz_odc_1_wsp_x = odc_1.poczatek.x;
```

  - itp. itd.

# Struktury

- Do zmiennych będących strukturami można definiować standardowo wskaźniki posługując się nazwą typu

```
punkt_2D_t * wsk_punkt_2D = &punkt_1;
```

  - wskaźnik do struktury umożliwia dostęp do jej składowych
  - istnieją dwie możliwe notacje:

```
(*wsk_punkt_2D).wsp_x
wsk_punkt_2D->wsp_x
```

- Stosowanie wskaźników do struktur umożliwia m.in. zmianę zawartości struktury wewnątrz wywoływanej funkcji
  - jak zwykle istnieje wiele możliwości wykonania tej operacji

```
void przesun_punkt_o_1_1( struct punkt_2D * punkt_in_out_p ){
    (*punkt_in_out_p).wsp_x += 1.0;
    double wsp_y = punkt_in_out_p->wsp_y;
    wsp_y += 1.0; // zmienne lokalne – wygodne, jeśli jest wiele operacji
    punkt_in_out_p->wsp_y = wsp_y; }
```

# Tablice struktur

- Podobnie jak dla zmiennych typów wbudowanych, także dla struktur można definiować tablice:

```
punkt_2D_t tab_punkt_2D[10] = {{0.0,0.0}};
```

- Dostęp do pól struktury odbywa się za pomocą standardowej składni:

```
tab_punkt_2D[0].wsp_x = 3.14;
```

  - nazwa tablicy jest wskaźnikiem do jej pierwszego elementu

- Operator sizeof może zostać użyty do obliczenia liczby elementów tablicy:

```
int n_el = sizeof tab_punkt_2D / sizeof( punkt_2D_t ) ;
```

- Tablice struktur mogą być alokowane dynamicznie posługując się zmiennymi będącymi wskaźnikami:

```
punkt_2D_t * tab_dyn_punkt_2D = NULL;
tab_dyn_punkt_2D = malloc ( n_el * sizeof( punkt_2D) );
```