

Złożone typy danych, struktury



- **Struktury** to złożone typy danych, które umożliwiają łączenie danych w jeden obiekt o wspólnej nazwie. Mogą być w nich przechowywane dane różnych typów, a ich strukturę definiuje programista.
- **Składowe struktury** mają swoje unikatowe nazwy (etykiety).
- Dostęp do danej składowej otrzymuje się poprzez podanie jej nazwy.

236

Struktury



Deklaracja typu struktury

```
struct nazwa_struktury {  
lista deklaracji składowych struktury  
};
```

Przykłady:

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

```
struct klient {  
char imie [20];  
char nazwisko [20];  
int wiek;  
};
```

237

Struktury



Deklaracja i definicja zmiennej typu strukturalnego

struct typ nazwa;

Deklaracja, definicja i inicjalizacja struktury

struct typ nazwa={dane początkowe – wyrażenia stałe};

Przykłady:

struct point pt;

struct point pt={20,30};

struct klient toyota;

struct klient toyota={"Jan", "Nowak",30};

238

Struktury



Deklaracja i definicja bezpośrednia

struct nazwa_typu_struktury {
lista deklaracji składowych struktury
} nazwa_zmiennej_strukturalnej;

Przykłady:

struct point {
int x;
int y;
} pt1, pt2;

struct klient {
char imie [20];
char nazwisko [20];
int wiek;
} toyota;

239

Struktury



Dostęp do struktury

nazwa-struktury.składowa

Przykłady:

```
printf("%d,%d", pt.x, pt.y);
```

```
printf("%s,%s,%d", toyota.imie, toyota.nazwisko,  
toyota.wiek);
```

240

Unie



- **Unia** to zmienna typu złożonego, która umożliwia przechowywanie obiektów różnych typów, ale nie jednocześnie. W unii w danym momencie można przechować tylko jeden obiekt.
- Kompilator rezerwuje dla unii pamięć o rozmiarze równym rozmiarowi jej największego typu składowego.
- **Składowe unii** mają swoje unikatowe nazwy (etykiety). Dostęp do danej składowej otrzymuje się poprzez podanie jej nazwy.

241

Unie

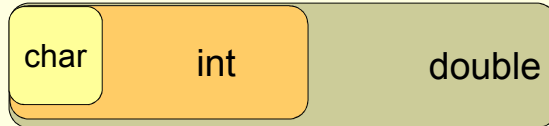


Deklaracja i definicja bezpośrednia

```
union nazwa_typu_unii {  
  lista deklaracji składowych unii  
} nazwa_zmiennej;
```

Przykład:

```
union symbol {  
  int ival;  
  double dval;  
  char cval;  
} u;
```



242

Unie



Dostęp do unii

nazwa-unii.składowa

Przykład:

```
printf("%d", u.ival);
```

243

Typ wyliczeniowy



- Typ wyliczeniowy został wprowadzony od standardu C89. Służy on do tworzenia zmiennych, które mogą przyjmować tylko pewne z góry ustalone stałe wartości reprezentowane za pomocą literałów wyliczeniowych.

```
enum nazwa-typu {wartosc1, wartosc2, wartoscN};
```

Przykłady:

```
enum pora_roku {wiosna, lato, jesien, zima};
```

```
enum symbol_type {INT, DOUBLE, CHAR};
```

244

Typ wyliczeniowy



- Typ wyliczeniowy jest typem całkowitym. Zmienne typu wyliczeniowego przechowują wartości całkowite skojarzone z literałami z listy wyliczeniowej. Domyślnie są to kolejne liczby nieujemne, chyba że zostały zdefiniowane inaczej.

Przykład:

```
enum pomiar {  
    start,          // start: 0 – wartość domyślna  
    odczyt,        // odczyt: 1 – wartość domyślna  
    stop=10};     // stop: 10 – wartość przypisana245
```

Typ wyliczeniowy



Przykład:

```
enum symbol_type utype;  
if (utype==INT)  
    printf("%d", u.ival);  
else if (utype==DOUBLE)  
    printf("%f", u.dval);  
else if (utype==CHAR)  
    printf("%c", u.cval);  
else printf("zły typ");
```

! Do zmiennych typu wyliczeniowego można przypisywać lub porównywać tylko literały z listy wyliczeniowej, a nie skojarzone z nimi liczby.

246

Typy złożone - strukturalne



- Typy strukturalne (tablice, struktury, unie) możemy zagnieżdżać jeden w drugim.

247

Typy złożone - strukturalne



Przykład – definicja tablicy symboli:

```
struct {  
  char *symbol_name;  
  enum symbol_type utype;  
  union symbol {  
    int ival;  
    double dval;  
    char cval;  
  } u;  
} symtab[NSYM];
```

! Każdy symbol bez względu na typ zajmuje tyle samo miejsca w tablicy.

Odwołanie do składowej `ival`:
`symtab[i].u.ival`