Języki programowania

Struktury, unie i pola bitowe

Struktura

- Struktura jest agregatem (różnych) typów danych.
- Składa się z kilku związanych zmiennych, zwanych *składowymi* albo *polami*.
- W przeciwieństwie do tablic, gdzie wszystkie elementy są jednego typu, każda składowa struktury może mieć inny typ.

- Informacje zawarte w strukturze powinny być ze sobą logicznie powiązane.

Definicja struktury

```
struct [nowy_typ] {
    typ pole1;
    typ pole2;
    //...
} [lista_zmiennych];
```

- **nowy_typ** jest nowym typem, który może być używany tak, jak inne nazwy typów.
- **lista_zmiennych** jest listą rzeczywistych obiektów (instancji) nowego typu.
- Jeden z powyższych elementów może być pominięty

Przykład definicji struktury

definicja nowego typu:

```
struct osoba {
   char imie[10];
   char nazwisko[20];
   short wiek;
};

osoba x, *y, z[10];
```

definicja obiektów pewnego typu:

```
struct {
   char imie[10];
   char nazwisko[20];
   short wiek;
} x, *y, z[10];
```

Jeden obiekt typu osoba jest przechowywany w pamięci komputera tak:

Przykład użycia struktury

- Aby sięgnąć do składowej struktury, a dokładniej: do składowej pewnego konkretnego obiektu strukturalnego, należy określić zarówno nazwę tego obiektu oraz nazwy składowej, rozdzielając je kropką, czyli operatorem .

  np. `x.wiek = 25;  z[4].wiek = 50;`

- Każdy typ strukturalny jest wyposażony w operator kopiowania. Można zatem napisać:

  ```
  osoba w = x;
  ```

  W wyniku tej operacji zostaną przekopiowane zawartości poszczególnych pól struktur.

Struktury

- Aby poznać wielkość struktury, tzn. ile obiekt tego typu zajmuje miejsca w pamięci, trzeba użyć operatora **`sizeof`**:

```
printf("%d", sizeof(struct osoba));
```

- Nie dla każdego kompilatora rezultat dla tego samego typu strukturalnego będzie miał tę samą wartość. Przyczyny:
  - Inne rozmiary typów standardowych,
  - Wyrównywanie rozmiary niektórych typów danych do granicy słów,
  - Odmienne zarządzanie pamięcią – różne rozmiary wskaźników.

Struktury

- Jak każdy inny typ może być przekazywana jako parametr do funkcji.
- Obiekt typu strukturalnego może być zwracany przez funkcję – sposób na zwrócenie kilku obiektów!
- Nazwy składowych struktury nie powodują konfliktów z innymi zmiennymi lub składowymi innych struktur o tej samej nazwie. Nazwa składowej struktury jest związana z nazwą struktury.
- Struktury można zagnieżdżać.

Wskaźniki do struktur

- Wskaźnik do struktury definiuje się w sposób analogiczny do wskaźników do innych typów.

  np. `osoba* z = &x;`

- Można teraz sięgnąć do wnętrza obiektu **x** za pomocą wskaźnika **z**: używamy do tego operatora **->**

  ```
  z->wiek = 30;
  ```

- Operator strzałkowy składa się ze znaku minus oraz znaku większy; między znakami nie może być spacji.
- Struktury są przekazywane do funkcji w całości (jeżeli są przekazywane przez wartość).
- W przypadku dużych struktur lepiej jest przekazywać taki obiekt przez wskaźnik lub referencję.

Oszczędzanie pamięci

- Istnieją dwa sposoby na oszczędzanie pamięci:
  - Umieszczenie w bajcie więcej niż jednego małego obiektu -> pola bitowe
  - Używanie tego samego miejsca pamięci do przechowywania w różnym czasie obiektów różnych typów -> unie.
- Obu mechanizmów używa się jedynie do optymalizacji kodu.
- Z reguły oba te mechanizmy są nieprzenośne pomiędzy różnymi środowiskami.

Pola bitowe

- Jest to pewien szczególny rodzaj struktury, której składowymi są pola składające się z określonej liczby bitów.
- Definiujemy je, określając za nazwą pola liczby bitów, które to pole zajmuje.
- Dozwolone są pola bez nazwy.
- Pole musi być typu całkowitego i używa się je, jak każdy obiekt typu całkowitego z tym wyjątkiem, że nie można pobrać adresu tego pola.

Definicja pola bitowego

```
struct sreg {
   unsigned ENABLE : 1;
   unsigned PAGE   : 3;
   unsigned        : 1;
   unsigned MODE   : 2;
   //...
};
```

```
sreg* SR0 = (serg*) 0777572;
//...
if(sreg->ENABLE) {
   //...
   SR0->ENABLE = 0;
}
```

- Przykład użycia: rejestr stanu komputera DEC PDP11/45

Pola bitowe

- To, jak rozmieszczone zostaną w pamięci składowe pola bitowego zależy od implementacji. Mogą być w jednym bajcie, ale nie muszą.
- ZALETA: oszczędność miejsca, bo dane są gęsto upakowane – można wydatnie zmniejszyć redundancję.
- WADA: wzrost wielkości i złożoności kodu potrzebnego na obsługę dostępu do komórki pola bitowego – zwykle dostęp do znaku (char) lub liczby całkowitej jest dużo szybszy.

Unie

- Unia jest pojedynczym obszarem pamięci, dzielonym przez kilka zmiennych.
- Zmienne te mogą być różnych typów.
- W danej chwili może być przechowywana tylko jedna zmienna.

Definicja unii

```
union [nowy_typ] {
    int    n;
    double x;
    char   z;
} [lista_zmiennych];
```

- Podczas przydzielania pamięci wszystkie trzy pola otrzymują ten sam adres.
- Unia ta ma rozmiar równy rozmiarowi największego pola. Rozmiar ten jest ustalany w momencie kompilacji.
- Do pola unii odnosimy się używając operatorów `.` i `->`
- Zawartość unii jest interpretowana przez użycie nazwy jednej ze składowych unii.

Używanie unii

- Unii zazwyczaj używamy w ten sposób, że zapisujemy wartość korzystając z jednego z pól, a odczytujemy używając tego samego pola.
- Unię można czasem używać do „konwersji typów”, ale wymaga to bardzo dużej znajomości danej maszyny i środowiska programistycznego (nie zalecane).
- Można także używać unii do „zaszyfrowywania” danych.
- Inicjalizacja unii: unię inicjalizuje się tylko daną odpowiadającą typowi pierwszego składnika tej unii.

Unie anonimowe

- Unią anonimową jest unia, która nie ma nazwy.

```
union {
   int    n;
   double x;
   char   z;
};
```

- Do składowych takiej unii odnosimy się po prostu podając nazwę tego składnika (bez użycia kropki), np.

```
n = 3;    x = 3.14;
```