Лекція 9. Структури як засіб створення нових типів даних

1. Структура як тип даних визначених користувачем.

2. Ініціалізація та доступ до полів структур.

3. Ключове слово typedef у застосуванні до структур.

4. Використання структур.

1. Структура як тип даних визначених користувачем

Прототипом для даних типу структур вважаються рядки прямокутних таблиць, що містять різні характеристики одного запису, скажімо, в реляційній базі даних. Характерним для таблиць наявність стовпців, в кожному з яких зберігаються однотипні дані. Однак в сусідніх стовпцях типи даних можуть відрізнятися. Якщо специфічною особливістю масивів є використання одного і того ж типу для всіх елементів масиву, то рядки таблиць можна уявити як послідовність полів даних різного типу. Для кожного поля рядка таблиці відомо найменування відповідного стовпця таблиці та тип розташовуваного в цьому полі значення. Наприклад, поле "Прізвище" заповнюється текстовою інформацією, поле "Рік народження" зберігає цілочисельні дані, на полі "Стать" достатньо записувати один символ 'Ч' або 'Ж' і т.д.

Те, що прийнято називати "шапкою" таблиці в програмуванні носить назву шаблону структури. Наприклад, шаблон структури, яка описує дані про книгу, може бути виглядати так:

```
struct book {
    char title[40];              //найменуванння
    char authors[30];            //автори
    char publishing_house[15];   //видавництво
    int year;                    //рік видання
    int pages;                   //кількість сторінок
    float price;                 //ціна
  }
```

Далі ідентифікатор ***book*** можна використовувати для оголошення конкретних змінних:

```
struct book b1,b2,b3;    //змінні типу book
```

В мові С++ службове слово ***struct*** в оголошенні змінних зазвичай опускають (але не в описі шаблону структури!):

```
book b1,b2,b3;
```

Рядковим полям в структурах, зазвичай задають фіксовані розміри. Це істотно спрощує їх обробку, адже робота з полями змінної довжини вимагала б значно більше зусиль пов'язаних з отриманням та вивільненням динамічної пам'яті.

За правилами синтаксису оголошення шаблону структури і змінних, пов'язаних з цією структурою, можна поєднати:

```
struct book {
    char title[40];
    char authors[30];
    char publishing_house[15];
    int year;
    int pages;
    float price;
  } b1,b2,b3;
```

Або без назви типу:

```
struct {
    char title[40];
    char authors[30];
    char publishing_house[15];
    int year;
    int pages;
    float price;
  } b1,b2,b3;
```

Таке оголошення анонімної структури є не надто зручним і не часто використовуються, бо, наприклад, для передачі параметрів у функції воно взагалі непридатне.

2. Ініціалізація та доступ до полів структур

При розгляді стандартних типів даних, статичних масивів і вказівників зазначалося про можливість ініціалізації оголошених змінних відповідних типів. Змінна типу структура також може бути ініціалізована, тобто при оголошенні можна надати значення цій змінній. Для цього після імені (ідентифікатора) оголошуваної змінної через знак дорівнює («=») у фігурних дужках через кому надається початкове значення кожному полю структури,

враховуючи заданий порядок полів структури. Слід дотримуватися відповідності типів полів та типів їх значень. Наприклад:

```
struct detal
{   int nomer_modeli, nomer_detali;
    float vartist_detali;
};
```

Тоді змінні типу detal можна оголошувати так:

```
struct detal det, det1={12089, 245, 187.57f}, det2;
```

Доступ до полів структурної змінної здійснюється через операцію «крапка» (.). Для того щоб отримати доступ до відповідного поля структурної змінної використовують наступну синтаксичну конструкцію:

```
ім'я_змінної.ім'я_поля
```

Така конструкція може входити у вираз, якщо поле вже містить дані, бути аргументом функції, бути лівою частиною оператора присвоєння, якщо це поле не оголошено константним. Над кожним з полів структурної змінної можуть виконуватися операції, які визначені для відповідного типу даних.

Наприклад, щоб вивести вміст структурної змінної на екран, необхідно виводити кожне поле окремо, інструкцією вигляду:

```
cout<<"\nНомер моделі : "<<det1.nomer_modeli;
cout<<"\nНомер деталі :" <<det1.nomer_detali;
cout<<"\nВартість деталі ="
<<det1.vartist_detali<<" грн.\n";
```

Операція доступу до поля структури використовується також при введенні значень полів структури, при чому кожного поля зокрема. Нехай треба ввести нову інформацію про деталь. Використовуючи інструкції виводу та вводу, заповнимо відповідні поля структурної змінної

```
cout<<"\nВведіть дані про деталь det:\n";
cout<<"\n Номер моделі ="; cin>>det.nomer_modeli;
cout<<"\n Номер деталі ="; cin>>det.nomer_detali;
cout<<"\n Вартість деталі ="; cin>>det.vartist_detali;
```

Якщо оголошено вказівник на змінну типу структура, тоді доступ до відповідного поля здійснюється поєднанням операції крапка з операцією розадресації. Та оскільки операція взяття поля структури має вищий пріорітет від операції розадресації, тоді для взяття поля за адресою спершу треба звернутися за цією адресою, а тоді вже доступатися до поля. Це означає, що операцію розадресації слід взяти в дужки, підвищивши її пріорітет:

```
(*вказівник_наструктуру).ім'я_поля_структури.
```

Це не дуже зручно, тому в C/C++ визначена спеціальна операція непрямого доступу до поля, тобто отримання поля за адресою структури. Вона задається двома послідовними символами «-» та «>», хоча в тексті виглядає як «стрілка вліво» (**->**). Наприклад:

```
detal * pd=&det1;

pd->nomer_detali=25;
```

3. Ключове слово typedef та його застосування до структур

У мові С як при створенні структур так і при оголошенні змінних необхідно перед іменем користувацького типу додавати ключове слово struct, що спричинює певну незручність. Граматика мови С передбачає можливість створення власних назв як стандартних, так і анрегованих типів даних з використанням ключового слова ***typedef***. Це ключове слово дозволяє скоротити описові інструкції, а також надати нове псевдо відомому типові. Так, наприклад, при оголошенні беззнакового цілого 8-ми байтного даного, що дозволяє зберігати до 20 десяткових розрядів, використання повного імені типу є досить громіздким:

```
unsigned long long a;
```

Запрвадження нового імені для стандартного типу значно в цьому випадку дозволяє скоротити оголошення:

```
typedef unsigned long long UnLL;

UnLL b, c;
```

Далі цей ідентифікатор типу може використовуватися в якості параметра функцій чи для оголошення змінних та констант у всій області його видимості.

Ще один приклад. Нехай ми опрацьовуємо матриці, тоді можна створити назву типу, що відповідатиме за статичний двовимірний масив. При цьому можна задавати тип елемента масиву, а також максимально можливу розмірність:

```
typedef int chuslo;

const int rd=20, sc=25;

typedef chuslo[rd][sc] matrix;

void vvid_mtr(matrix a,int n,int m,char*ms);

matrix a1,a2;
```

Ключове слово ***typedef*** дозволяє спростити текст програми та зекономити зусилля також і при оглошенні структурних змінних.

Для спрощення подальшого оголошення структурний змінних використовується оператор typedef. Розглянемо застосування для оголошення структури, що задає комплексне число

```
typedef struct COMPLEX
{double Re,Im;}complex;
complex cc={1,2};
```

При цьому можна і не вказувати імені шаблону структури, а просто вказувати нове ім'я оголошуваного типу даних:

```
typedef struct
{int chus, znam; } drib;


typedef struct
{double x, y, z; } Point_3D;
```

Надалі імена оголошених з допомогою описового оператора типів структур можуть використовуватися і при оголошення змінних, і при оголошенні та у визначенні функцій.

4. Використання структур

Для структур, оголошених з використанням одного і того ж шаблону, допустима операція присвоювання:

```
b1=a;    //усі поля структури a копіюються в b1
```

На жаль, однойменні поля стрічкового типу у структур так копіювати не можна - необхідно вдаватися до послуг функцій типу strcpy:

```
strcpy(b1.authors,a.authors);   //копіюємо поле authors
```

Структури можуть бути аргументами функцій. Якщо функція не змінює структуру, то таку структуру можна передати за значенням. Якщо обробка структури в функції пов'язана із зміною вмісту полів, то таку структуру необхідно передавати за вказівником або за посиланням.

У деяких таблицях використовуються багатоповерхові шапки, коли певне поле заголовка, у свою чергу, розпадається на кілька полів. Така ж ситуація може зустрітися і в структурах, коли в якості чергового поля виступає інша структура. І рівень вкладення таких полів нічим не обмежений.

Функції можуть не тільки одержувати структури в якості своїх параметрів, але і повертати результати у вигляді структур. Це означає, що функція, що повертає значення може мати в якості результату своєї роботи сукупність значень полів відповідної структури.