Тема 12
Сложни типове данни. Структури в език С

Типовете данни в програмните езици могат да бъдат класифицирани като:

- ***скаларни -*** типове данни, които се състоят от един елемент (компонент);
- ***структурни*** - типове данни, които се състоят от повече от един елемент.

Типичен пример за данни от структуриран (сложен) тип са масивите.

В език С като структуриран тип данни могат да бъдат класифицирани: ***структурите (struct)*** и ***обединенията (union)***. Тези типове данни могат да се използват за реализация на различни структури от данни като списъци и дървовидни структури. Тъй като структурите и обединенията в С++ са частен случай на класове, разглеждането на тези типове данни в темата, ще бъде представено според стандарта ANSI C.

1. Структури (*struct*)

Структурата в език С е сложен тип данни, състоящ се от повече от един разклонени елемента. Структурата има еквивалентен тип ***Record (запис)*** в програмен език Pascal.

1.1. Декларация на структура

Чрез декларирането на структура се създава нов тип данни, който е дефиниран от програмиста. Декларацията има следният общ вид:

```
struct   Име
         { тип  име1;    //описание на първи елемент
           тип  име2;    //описание на втори елемент
            ...
         };
```

Структурата представлява поредица от елементи, които в общия случай са от различен тип.

Име е валиден идентификатор, служещ за дефиниране на структурата. Името се предшества от ключова дума ***struct***. В блока се описват елементите на структурата, определени чрез техните ***имена*** (***идентификатори***) и техният ***тип***.

Пример за деклариране на структура, описваща точка е:

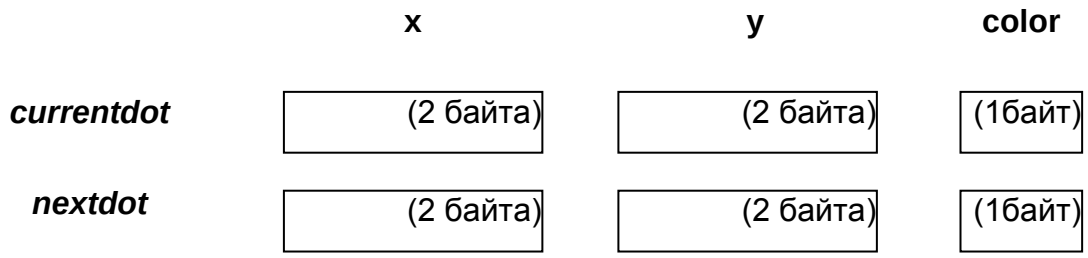
```
struct dot                              // структура dot , описваща точка
{
        int x;                         // с координатите: x,y и цвят:color
         int y ;
         unsigned char color;
};
```

Декларирането на структура не е еквивалентно на заделяне на памет за променлива, а определяне на нов тип т.е. то се явява описание на шаблон за този тип. След като бъде дефинирана дадена структура, в последствие тя може да се използва като всеки един от типовете за дефиниране на променливи от този тип.

Пример:

```
...
struct dot currentdot, nextdot;
```

В случая – дефинирани са 2 променливи **currentdot** , **nextdot** , които са от тип **struct dot**. Съответно, компилаторът заделя за всяко едно от тях следните клетки памет: за **current dot**: елемент **x** - 2 байта; елемент **y** - 2 байта; елемент **color** – 1 байт. (фиг. 12.1)



*Фиг. 12.1. Променливи **currentdot**, **nextdot** от тип **struct dot***

Дефинирането на променлива от тип структура е възможно още при декларацията на структурата. Например, предишните дефиниции на ***struct dot*** и на променливите ***currentdot*** и ***nextdot*** могат да бъдат заменени със следните програмни редове:

```
struct dot
{
        int x;
        int y;
        unsigned char color;
} currentdot, nextdot;
```

Възможно е, при деклариране на структура да не се посочва нейното име. Например:

```
struct { int x;
```

```
        int y;
        unsigned char color } current, next;
```

В този случай, структурата се използва само за променливите ***current*** и ***next*** и ако се наложи използването и за други променливи, необходимо е отново да бъде предефинирана. Подобни структури се наричат ***анонимни***.

1.2. Оператори за работа със структури

Операторите, които са допустими със структури са :

- прилагане на адресен оператор **&**;
- обръщение към елемент от структура (оператори **.** и **->**);
- инициализиране на структури.

Адресна оператор извличане на адрес (**&**) може да се приложи към променлива – структура така, както към обикновена променлива.

Пример:

```
struct dot
{
        int x;
        int y;
} current, *pcurrent;


pcurrent = &current;    //променлива pcurrent сочи (съдържа адреса на)
                        // структура current
```

В случая са дефинирани променлива структура (current) и указател към структура от този тип (pcurrent). На указателя се присвоятва адреса на променливата.

Обръщането към елемент от структура може да се извърши чрез един от следните оператори:

- Оператор **.** (точка)

Операторът служи за **извличане на елемент от структура, чрез променлива структура** т.е. прилага се върху променлива-структура в следния вид:

променливаСтруктура . елемент

Пример :

```
current.x = 10; // задава стойности 10,20 за координатите x и y на
current.y = 20; // текущата точка
```

Чрез оператор точка се извършва обръщение към компонентите **x** и **y** на структура **current**, които са всъщност променливи от тип ***int***.

- Оператор **->** (стрелка)

Операторът служи за **извличане на елемент от структура чрез указател** и се прилага към указател от тип структура в следния вид:

указателКъмСтруктура -> елемент

Пример:

```
pcurrent=&current;
pcurrent->x=10; //задава стойности 10,10 за координатите x и y на
pcurrent->y=10; // текущата точка
```

Променлива структура се инициализира като всяка променлива - след знак за присвояване във фигурни скоби се записват поредица от начални стойности за всеки елемент от структурата.

Пример :

```
struct dot current  = {100, 28};
```

По този начин компонент **x** на структура ***current*** получава стойност 100, а компоненет **y** - стойност 28.

С декларирането на структура се създава нов тип данни, дефиниран от програмиста. Този тип данни е равностоен на вградените типове и може да се изпозлва както за дефиниране на променливи от този тип, така и като тип на параметри на функция и на върнат резултат. Следващият програмен код е пример за използване на структура като параметър на функция и тип на върнат резултат:

```
#include <stdio.h>
#include <math.h>

struct complex {
      double re;
      double im;
};

struct complex add(struct complex, struct complex);

main()
{
      struct complex a,b,c;

      printf("Input complex a:\nre=");
      scanf("%lf",&a.re);
      printf("im=");
      scanf("%lf",&a.im);

      printf("Input complex b:\nre=");
      scanf("%lf",&b.re);
      printf("im=");
      scanf("%lf",&b.im);


      c=add(a,b);
      printf("\nThe result of a + b is: re = %lf im = %lf",c.re,c.im);
```

```
      return 0;
}

struct complex add(struct complex x, struct complex y)
{
      struct complex z;
      z.re=x.re+y.re;
      z.im=x.im+y.im;
      return z;
}
```

В примера е декларирана структура **complex**, описваща комплексно число. Функцията **add()** събира две комплексни числа, които са предадени като параметри и връща като резултат тип **complex**, съответстващ на сбора на комплексните числа.

2. Съвместно използване на структури и масиви

Масивите и структурите могат да се използват съвместно. Възможно е да се дефинира масив от структури или масив да е елемент на структура. В случай, че се дефинира масив от структури, дефиницията има следния общ вид:

struct имеСтруктура имеМасив [граница];

Пример :

```
struct dot d[100];  /* деклариране на масив от 100 елемента структури от точки */
```

Достъпът до компонентите на всеки един елемент от масива може да се осъществи чрез оператори **.** и **->**. Първата се използва съвместно с оператор ***индексиране*** (**[ ]**) за достъп до елемент на масив.

Достъпът до елементите на масива d е по следния начин :

```
d[0].x = 0;     // достъп до елемент 0 на масив d от структури dot
d[0].y = 0;
d[1].x = 0;     // достъп до елемент 1 на масив d от структури dot
d[1].y = 0;
d[i].x = 0;     // достъп до елемент i на масив d от структури dot
d[i].y = 0;
```

Тъй като името на масива е указател към първия елемент, достъпът до компонент на структура може да се осъществи и като:

```
d->x = 0;     // достъп до елемент 0 на масив d от структури dot
d->y = 0;
(d+1)->x = 0;     // достъп до елемент 1 на масив d от структури dot
(d+1)->y = 0;
(d+i)->x = 0;     // достъп до елемент i на масив d от структури dot
(d+i)->y = 0;
```

Възможно е използването на масив като елемент от структура. Пример за такава дефиниция е:

```
struct node
 {
       char city [10];
       float hour;
 } node1;
/* дефиниране на структура node състояща се от символен низ city  и променлива hour */
```

Достъпът до елементите се осъществява по следния начин:

```
node1.city[0]='B'; // достъп до структура съдържаща масив
node1.city[1]='a';
node1.city[2]='p';
node1.city[3]='н';
node1.city[4]='a';
node1.city[5]='\0';
```

Посоченият пример е за достъп до елементи на масив, който е компонент на структура. За посочения пример, в повечето случаи, достъпът до символен низ ***city*** чрез неговото име.

Пример за съвместни използване на структури и масиви е следната програма:

```
#include <stdio.h>

struct dot {  int x;
              int y;
            };

const N=20;

int InputDots( struct dot *current);
void OutputDots( int n, struct dot *current);

void main()


{   int n;
    struct dot d[20];

    n=InputDots(d);
    OutputDots( n, d);

}


int InputDots( struct dot *current)
{
      int n;
    int i;
```

```
    do
    {
            printf("\nБрой точки=");
            scanf("%d",&n);
    } while (n>N || n<5);

    for (i=0;i<n;i++)
    {
            printf("\n X координата на точка %d =", i+1);
      scanf("%d", &current[i].x);
        printf("\n Y координата на точка %d =", i+1);
        scanf("%d", &current[i].y);
     }
    return n;
}

void OutputDots( int n, struct dot *current)

{   int i;


    for (i=0;i<n;i++)
      {
            printf("\nточка %d: X = %d", i+1,current->x);
        printf(" Y = %d", current->y);
        current++;
      }
}
```

В главната функция е дефиниран масив **d** от структури **dot**. Масивът описва поредица точки. Функцията ***InputDots()*** служи за въвеждане на точки от клавиатурата. Тя изисква като параметър началото на масива и връща като резултата броя на въведените точки **n**. Функцията ***OutputDots()*** извежда на екрана стойностите на координатите на точките. Параметри на функцията са: броя на точките и началото на масива от структури.

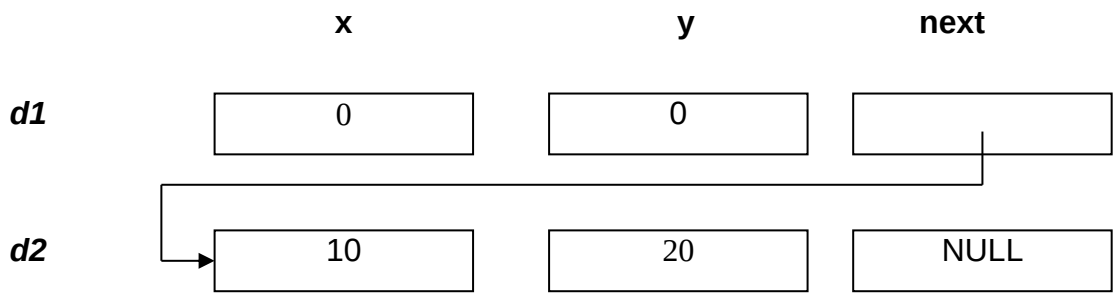
3. Рекурсивно обръщение към структури

Като елемент на структура може да се използва и променлива – указател. Още повече, позволява се като елемент на структура да се дефинира указател към самата нея. По този начин могат да се формират сложни структури от данни като например списъци и дървета.

Пример:

```
struct dot
 { int x;
    int y;
   struct dot *next;                    /* указателят next сочи следващ
елемент от същата структура */
} d1, d2;
...
d1.x=0;
d1.y=0;
d1.next=&d2;
```

```
d2.x=10;
d2.y=20;
d2.next=NULL;
```



*Фиг. 12.2. Променливи **d1**, **d2** от тип **struct dot***

В конкретния пример, стойностите за координатите на първата точка са **0,0**; за втората точка – **10,20**; указателят ***next*** на първата точка сочи адреса на втората точка. Втората точка е последна в списъка – указателят ***next*** има стойност NULL (фиг. 12.2).

Относно въпроси по темата на адрес: ln_zh_st@yahoo.com