17. Обединения. Дефиниране имена на типове. Изборен тип.

Обединения

В езика C/C++ е предвидена възможност за описание на област от паметта, в която могат да се записват в различно време променливи от различен тип. Описания от този тип се наричат *обединения*.

```
union име_на_типа_на_обединението {
      тяло_на_обединението
};
```

Всички правила за дефиниране и работа със структури са в сила и тук. Съществената разлика е, че за променливи от тип обединение, компилаторът разпределя памет, досатъчна за записване на елемент с максимална дължина, тъй като полетата на обединение започват от една и съща граница. Дължината на обединението не е по-малка от дължината на максималната от дължините на отделните полета от типа. В така разпределената памет може да се записва само по един произволен елемент на обединение.

```
union set {
      int k;
      float f;
      char c;
      char string[5];
} example;
```

Обръщението към елемент на обединието е същото, както при структурите – чрез операцията „.“ (точка) или „->“ (стрелка).

```
example.k = 2;
example.f = 2.50;

union set *p;
p = &example;
p->k = 2;
```

Кой елемент от обединението фактически е записан в паметта се контролира единствено от програмиста. За тази цел се въвежда допълнитена променлива от цял тип.

Повечето компилатори поддържат структури с елементи обединения, обединения с елементи структури и масив, чиито елементи са обединение. Използването на такъв масив позволява да се избегне ограничението, че елементите на масива трябва да са от един и същи тип.

```
union first {
      int k;
```

```
        float f;
        char c;
} array[10];

array[0].k = 5;
array[1].f = 7.2;
array[2].c = 'F';
```

Обединението позволява да се обръщаме по различен начин към една и съща област от паметта. Обикновено компилаторите реализират инициализиране на обединение само по първото описано поле.

Дефиниране имена на типове

Една удобна езикова конструкция в C/C++ е дефинирането на нови имена на типове на променливи. Това се извършва по следния начин:

```
typedef стар_тип нов_тип
```

„Нов тип" е ново име, заместващо името, описано в „стар тип". Новото име е произволен, допустим за езика идентификатор. Обикновено се записва с главни букви.

```
typedef float REAL;
REAL x, y;
float x, y;
```

Записите на последните два реда са еквиварлентни.
typedef може да се използва и с масиви, указатели, структури и обединения.

```
typedef double EXS[100]
typedef char* WORD;
```

С **typedef** не се определят нови типове, а само нови имена за съществуващи в езика типове за променливи.

```
struct address {
        char town[20];
        char street[20];
        int num;
};

typedef struct {
        char name[30];
        struct address adr;
        char egn[11];
} officer;

officer f;
```

Ползи:

- увеличава се прегледността на програмата, тъй като се използват имена, отразяващи конкретно предмета на промяна.
- по-добра преносимост на програмите

Изборен тип

```
enum име_на_типа { списък_от_стойности };
```

Името на типа и стойностите от списъка са идентификатори.

```
enum Boolean {false, true} f;
```

Идентификаторите в списъка от стойности се интерпретират като имена на целочислени константи и като такива могат да участват в изрази и операции. Компилаторът присвоява на идентификаторите в списъка от стойности цели числа в нарастващ ред, като започва от 0. Повечето компилатори позволяват идентификаторите в списъка да бъдат инициализирани, като следващите го идентификатори получават нарастващи с 1 стойности, освен ако не са също инициализирани.

```
enum counter {start, first, second, tenth = 10} e;

typedef enum Boolean {Monday = 1, Tuesday, Wednesday,
Thursday, Friday, Saturday, Sunday} DAYS;

DAYS week_day, *ptr;
ptr = &week_day;
*ptr = Friday;
```

Следващи теми: *18. Функции от високо ниво за работа с текстови файлове*
19. Функции от високо ниво за работа с двоични файлове