

Опрацювання структур у функціях

Оскільки мова С інтерпретує структури як звичайні змінні, а не вказівники, можна передавати значення структури у функцію через формальний параметр або повертати структуру як значення результату виконання функції. Шаблон структури повинен бути описаний перед функціями, які використовують даний тип структури. Наприклад:

```
struct Date {  
    int d; /* день */  
    int m; /* місяць */  
    int y; /* рік */  
};  
.....  
struct Date GetCurDate ( void )  
{ .....}
```

Приклад використання функції для заповнення змінної *today* типу *struct Date* даними поточної дати:

```
struct Date today ;
```

```
today = GetCurDate ( ) ;
```

```
printf (“Сьогодні: %d - %d - %d р.”, today.d, today.m,  
today.y );
```

Використовують різні способи передавання структур у функцію для опрацювання:

- цілі структури через відповідні параметри-структури;
- адреси структур через параметри - вказівники на структури;
- передавання окремих полів структур.

Приклад: задано масив структур, в яких зберігаються персональні дані (шаблону структур присвоєно ім'я *PDAT*). Поле *workplace* задає місце праці:

```
typedef struct person_data {  
    ...  
    char workplace [60];  
} PDAT;
```

Треба внести зміни місця праці в масив структур.

Кілька варіантів

1. Передавання у функцію значення всієї структури. Функція замінює найменування *oldname* на нове і повертає опрацьовану структуру.

```
#include <string.h>
```

```
PDAT ChangeWorkPlace1 (PDAT member)
```

```
{
```

```
    char * oldname = “ старе_найменування”;
```

```
    char * newname = “ нове_найменування”;
```

```
    if ( strcmp (member.workplace, oldname) == 0)
```

```
        strcpy (member.workplace, newname );
```

```
    return member;
```

```
        2 -> у буфер обміну
```

```
}
```

Для внесення змін у всю базу даних функцію застосовують циклічно:

```
# define n 100
```

```
PDAT persondat [n];
```

```
... /* заповнення даними масиву persondat [n] */
```

```
for ( i=0; i<n; i++ )
```

```
    persondat [i]= ChangeWorkPlace1 (persondat [i]);
```

3 -> присвоєння

1 -> у форм.

параметр

Ця версія нераціональна, оскільки для кожної структури масиву тричі виконується її копіювання (1,2,3), навіть, якщо поля взагалі не змінюються.

2. Використання вказівника на структуру (формальний параметр *pmemb*). У разі виклику функції цей параметр отримує адресу відповідної структури, тому функція звертається безпосередньо до потрібних полів без копіювання.

```
void ChangeWorkPlace2 (PDAT * pmemb)  
{  
    char * oldname = “ старе_найменування ” ;  
    char * newname = “ нове_найменування ” ;  
    if ( strcmp (pmemb ->workplace, oldname) == 0)  
        strcpy (pmemb ->workplace, newname) ;  
}
```

Для внесення змін у весь масив:

```
for ( i=0; i<n; i++ )
```

```
ChangeWorkPlace2 (persondat +i ); /*або
```

```
@persondat[i]*/
```

3. Використання формального параметру *workname*, що задає адресу рядка, який треба перевірити. Відповідним фактичним параметром у викликах функції має бути адреса поля структури.

```

char * ChangeWorkPlace3 (char * workname )
{
    char * oldname =“ старе_найменування”;
    char * newname=“ нове_найменування”;
    if ( ! strcmp ( workname, oldname ))
        strcpy ( workname, newname );
    return workname;
}

```

Використання функції

```

PDAT *pdat;    /**/
for (pdat = persondat; pdat<persondat + n; pdat++)
    ChangeWorkPlace3 (pdat -> workplace );

```