

# Структури

## Оголошення структури

Структури дозволяють об'єднувати в єдиному об'єкті сукупність значень, які можуть мати різні типи. Синтаксис опису структури

:

```
struct [ім'я_структури]
```

```
{
```

```
    тип1 елемент1;
```

```
    тип2 елемент2;
```

```
    .....
```

```
    типN елементN;
```

```
} [список описів];
```

Опис структури не виділяє місця у пам'яті під елементи структури, а

визначає лише шаблон, що описує характеристики змінних, що будуть

розміщуватися у конкретній структурі. Щоб ввести змінні та зарезер-

*Приклад:* представлення поняття "дата", що складається з декількох частин: число (день, місяць, рік), назва тижня та місяця.

```
struct date {  
    int day ;  
    int month ;  
    int year;  
    char day_name[15];  
    char mon_name[14];  
} arr[100], *pd, date, new_date;
```

В даному прикладі оголошуються:

*date, new\_date* - змінні типу структури *date*; *pd* - покажчик на тип *date*;

*arr* - масив із 100 елементів, кожний елемент якого має тип *date*.

Можливий і наступний опис структури з використанням *typedef*:

```
typedef struct DataTypes {  
    float aFloat;  
    int anInt;  
    char aString[8];  
    char aChar;  
    char aLong;  
} DataTypes;  
DataTypes data;
```

Пам'ять розподіляється у структурі покомпонентно, зліва-направо,  
від молодших до старших адрес пам'яті.



Доступ до окремого елемента структури забезпечується операторами вибору:

. (прямий селектор) та -> (непрямий селектор).

```
struct mystruct {  
    int i;  
    char str[21];  
    double d;  
} s, *sptr=&s;  
s.i =3;  
sptr->d = 1.23;
```

Звертання до окремих елементів структури ( наприклад *date* )

```
date.year=2005;  
printf ("%d-%d-%d", date.day, date.month, date.year);  
scanf ("%d", date.day);  
gets (arr[0].day_name);
```

Ініціалізація структури подібна до тієї, що у масивах, але з урахуванням розміщення даних різного типу.

```
struct person {  
    char frnm[20];  
    char nm[30];  
    int year;  
    char s;  
};  
struct person poet={"Taras", "Shevtchenko",1814, 'M'},  
classics[]={{"Alfred", "Aho", 1939, 'M'},{"Seimour", "Ginzburg"},  
/* ... */ {"Jeffrey", "Ulman", 1938, 'M'}};
```

Ініціалізується змінна *poet* і масив структур *classics*.  
Значення *classics[1].year* і *classics[1].s* мають значення відповідно 0 і '\0'.

Для змінних одного і того ж самого структурного типу визначена операція присвоювання, при цьому здійснюється поелементне копіювання значень полів.

Наприклад, для структури *date*

```
date = new_date;
```

Кожний опис структури вводить унікальний тип структури, тому в наступному фрагменті програми:

```
struct A {  
    int i,j;  
    double d;  
} a, a1;  
struct B {  
    int i,j;  
    double d;  
} b;
```

об'єкти *a* і *a1* мають однаковий тип *struct A*, але об'єкти *a* і *b* мають різні типи структури. Структурам можна виконувати присвоєння тільки в тому випадку якщо і вихідна структура,

і структура, які присвоюється мають один і той же тип.

```
a = a1;    /* можна виконати, так як a і a1 мають  
            однаковий тип */  
a = b;    /* помилка */
```

Для порівняння структур необхідно перевіряти рівність відповідних полів цих структур.

```
struct point  
{ float x,y;  
  char c;  
} point1, point2;  
  if ( point1.x==point2.x && point1.y==point2.y &&  
    point1.c==point2.c)  
{  
  /* ... */  
};
```

Доцільним є зв'язок структур та покажчиків. Так опис *date \*pdate* утворить покажчик на структуру типу *date*. Використовуючи цей покажчик, можна звернутися до будь-якого елемента структури шляхом застосування операції *->*, тобто *date ->year*, або що еквівалентно операції *(\*pdate).year*. Однак слід зауважити, що спільне використання цих типів потребує від програміста достатньо високої кваліфікації.

## ***Масиви структур***

Як і звичайними масивами простих типів, так само можна оперувати масивами структур, елементи якого мають структурований тип.



```
typedef struct Date  
{  
    int d; /* день */  
    int m; /* місяць */  
    int y; /* рік */  
} Date;  
Date arr[100];
```

Оголошено масив *arr*, що складається із 100 елементів, кожний з яких має тип *Date*. Кожний елемент масиву - це окрема змінна типу *Date*, що складається із трьох цілих елементів – *d*, *m*, *y*.

Доступ до полів структури аналогічний доступу до звичайних змінних, плюс використання індексу номеру елементу у квадратних дужках:

```
arr[25].d=24;            arr[12].m=12;
```

Запропонуємо програму, в якій реалізується концепція структурованого типу. Окремими функціями реалізуємо ініціалізацію елементів структури, додавання нового значення, виведення дати на екран, визначення високосного року.

```
#include<stdio.h>  
#include<conio.h>  
typedef struct Date  
{  
    int d; /* день */  
    int m; /* місяць */  
    int y; /* рік */  
} Date;  
void set_date_arr (Date *arr, Date value, int n)  
{  
    int i;  
    for (i=0; i<n; i++)  
    { arr[i].d=value.d;  
      arr[i].m=value.m;  
      arr[i].y=value.y; }  
}
```

```
void print_date_arr (Date *arr, int n)
{
    int i;
    for (i=0; i<n; i++)
    {   printf("%d.%d.%d\n", arr[i].d, arr[i].m, arr[i].y);
    }
}

void print_date (Date &d)
/* виведення на екран дати */
{
    printf("%d.%d.%d\n",d.d,d.m,d.y);
}

void init_date (Date &d, int dd, int mm, int yy)
/* ініціалізація структури типу Date */
{
    d.d=dd;
    d.m=mm;
    d.y=yy;
}

int leapyear (int yy)
/* визначення, чи високосний рік */
{
    if ((yy%4==0 &&yy%100!=0) || (yy%400==0)) return 1;
    else return 0;
}
```

```
void add_year(Date &d,int yy)  
/* додати yy років до дати */  
{  
    d.y+=yy;  
}
```

```
void add_month(Date &d,int mm)  
/* додати mm місяців до дати */  
{  
    d.m+=mm;  
    if (d.m>12)  
        {  
            d.y+=d.m/12;  
            d.m=d.m%12;  
        }  
}
```

```
void add_day(Date &d,int dd)  
/* додати dd днів до дати */  
{  
    int days[]={31,28,31,30,31,30,31,31,30,31,30,31};  
    d.d+=dd;  
    if (leapyear(d.y)) days[1]=29;  
    while ((d.d>days[d.m-1]))  
    {  
        if (leapyear(d.y)) days[1]=29;  
        else days[1]=28;  
        d.d-=days[d.m-1];  
        d.m++;  
    }
```

```
if (d.m>12)
    {
        d.y+=d.m%12;
        d.m=d.m/12;
    }
}
}
void main(void)
{
    Date date1, date2;
    Date array[10]={{12,11,1980}, {15,1,1982}, {8,6,1985},
                   {8,8,1993}, {20,12,2002}, {10,1,2003}};
    clrscr();
    init_date (date1, 15, 12, 2002);
    add_day (date1, 16);
    print_date (date1);
    puts ("");
    init_date (date2, 1, 1, 2003);
    add_month (date2,10);
    print_date (date2);
    puts ("");
    print_date_arr (array, 6);
}
```

# Об'єднання (union)

Об'єднання дозволяють в різні моменти часу зберігати в одному об'єкті значення різного типу. В процесі оголошення об'єднання з ним

асоціюється набір типів, які можуть зберігатися в даному об'єднанні.

В кожний момент часу об'єднання може зберігати значення тільки одного типу з набору. Контроль за тим, значення якого типу зберігається в даний момент в об'єднанні покладається на програміста.

Синтаксис :

```
union [ім'я_об'єднання]  
{  
тип1 елемент1;  
тип2 елемент2;  
.....  
типN елементN;  
} [список описів];
```

Пам'ять, яка виділяється під змінну типу об'єднання, визначається розміром найбільш довгого з елементів об'єднання. Всі елементи об'єднання розміщуються в одній і тій же області пам'яті з однієї й тієї ж адреси.

Значення поточного елемента об'єднання втрачається, коли іншому елементу об'єднання присвоюється значення.

*Приклад 1:*

```
union sign  
{  
    int svar;  
    unsigned uvar;  
} number;
```

*Приклад 2 :*

```
union  
{  
    char *a,b;
```

В першому прикладі оголошується змінна типу об'єднання

*number*. Список оголошень елементів об'єднання містить дві

змінні: *svar* типу *int* і *uvar* типу *unsigned*. Це об'єднання дозволяє запам'ятати ціле значення в знаковому або в беззнаковому вигляді. Тип об'єднання має ім'я *sign*.

В другому прикладі оголошується змінна типу об'єднання з

ім'ям *var*.

Список оголошень елементів містить три оголошення: покажчика *a* на значення типу *char*, змінної *b* типу *char* і масиву *f* з 20 елементів типу *float*. Тип об'єднання не має

імені. Пам'ять, що виділяється під змінну *var*, рівна пам'яті, необхідної для зберігання масиву *f*, так як це

найдовший елемент об'єднання.