Лекція 20

Структури. Використання структур

У програмуванні є багато випадків, коли може знадобитися більше однієї змінної для представлення об'єкта. Наприклад, щоб представити самого себе, ви, швидше за все, захочете вказати своє ім'я, день народження, зріст, вага або будь-яку іншу інформацію. наприклад:

```
string myName;
int myBirthDay;
int myBirthMonth;
int myBirthYear;
int myHeight;
int myWeight;
```

Але тепер у вас **6** окремих незалежних змінних. Якщо ви захочете передати інформацію про себе в функцію, вам доведеться передавати кожну змінну окремо. Крім того, якщо ви захочете зберігати інформацію про когось ще, вам доведеться оголосити ще 6 змінних для кожної людини додатково! Неозброєним оком видно, що така реалізація не дуже ефективна.

На щастя, C ++ дозволяє створювати власні призначені для користувача типи даних. Це типи, які групують кілька окремих змінних разом. Одним з найпростіших користувацьких типів даних є структура. Структура (**struct**) дозволяє групувати змінні різних типів в єдине ціле.

Оголошення і визначення структур

Правило. Оголошення структурної змінної здійснюється в 2 етапи:

- оголошення шаблону структури як нового типу даних. На цьому етапі пам‘ять не виділяється. Формується тільки інформація про вміст структури;
- оголошення самої змінної. На цьому етапі виділяється пам‘ять для кожного поля (змінної), що описується в шаблоні структури.

Синтаксис оголошення структури:

struct <ім'я> {

<Тип1> <поле1>;

<Тип2> <поле2>;

...

<ТипN> <полеN>;};

Оскільки структури визначаються користувачем, то спочатку ми повинні повідомити компілятору, як виглядає наша структура, перш ніж її використовувати. Для цього оголосимо її, використовуючи ключове слово struct. наприклад:

struct Employee

{ **short** id;

int age;

double salary;};

Ми визначили структуру з ім'ям ***Employee***. Вона містить 3 змінні: *id* типу **short**, *age* типу **int** і *salary* типу **double**. Ці змінні, які є частиною структури, називаються членами (або полями). ***Employee*** – це просто оголошена структура, хоч ми і повідомляємо компілятору, що вона має змінні-члени, пам'ять під неї зараз не виділяється. Імена структур прийнято писати з великої літери, щоб відрізняти їх від імен змінних.

Попередження. Одна з найпростіших помилок в С ++ – забути крапку з комою в кінці оголошення структури. Це призведе до помилки компілятора в наступному рядку коду. Сучасні компілятори, такі як Visual Studio 2010 і вище, вкажуть вам, що ви забули крапку з комою в кінці, але старіші компілятори можуть цього і не зробити, через що таку помилку буде важко знайти.

Щоб використовувати структуру ***Employee***, ми просто оголосимо змінну типу ***Employee***:

Employee john // ім’я структури Employee починається з великої літери,

//а мінної john з маленької

Тут ми визначили змінну типу ***Employee*** з ім'ям *john*. Як і у випадку зі звичайними змінними, визначення змінної структури призведе до виділення пам'яті під неї.

Оголосити можна і кілька змінних однієї структури:

Employee john; // створюємо структуру Employee для john

Employee james; // створюємо структуру Employee для james

або

Employee john, james;

Ім'я структурної змінної може бути вказано при оголошенні структури. У цьому випадку воно розміщується після закриваючої фігурної дужки}. Область видимості такої структурної змінної буде визначатися місцем опису структури.

struct complex_type // ім'я структури – комплексне число

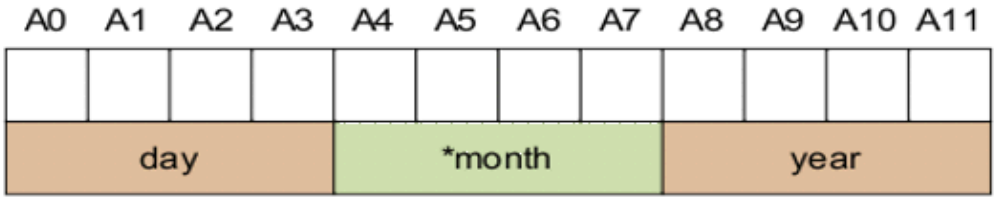
{ double real;

double imag;} Number; // ім'я структурної змінної

Структури зручно використовувати, наприклад, для визначення дати:

```
struct date
{  int day;     // 4 байта
  char *month; // 4 байта
  int year;    // 4 байта};
```

Поля структури розташовуються в пам'яті в тому порядку, в якому вони оголошені:



Полями структури можуть бути будь-які оголошені типи, крім самої структури цього ж типу, але можна зберігати покажчик на структуру цього типу:

```
1.  struct Book
    {    char title[70]; // символьний рядок
```

```
    char author[50];
    int year;
    float price;};
```

2. **struct** x {
    int a [10] [10]; /* массив цілих 10 x 10 */
    float b;} y;

3. У випадку, коли кілька полів мають один тип, то їх можна перерахувати через кому:

```
struct Point3D {
    int x, y, z;};
```

4. struct node {
    void* value;
    struct node *next;};

Структура також може бути анонімною. Тоді ми не зможемо використовувати ім'я структури надалі.

```
struct {
    int x;
    int y; } A;
```

Для змінної типу структура виділяється пам'ять розмір якої можна визначити застосувавши функцію **sizeof()**:

```
Book B;
int size;
size = sizeof(B); // буде виділено 128 б. (70+50+4+4=128)
```

Необхідно пам’ятати, що поля структури вирівнюються до парного значення.

Зауваження. Коли ми визначаємо нову структуру за допомогою службового слова **struct,** в просторі імен структур (воно не має нічого спільного з просторами імен C ++) створюється новий ідентифікатор. Для доступу до нього необхідно використовувати службове слово **struct**. Можна визначити новий тип за допомогою службового слова **typedef.** Тоді буде створений псевдонім для нашої структури, видимий в глобальному контексті.

```
// Визначаємо нову структуру
```

```
struct point_t {
    int x;
    int y;};
// Визначаємо новий тип
typedef struct point_t Point;
void main () {
    // Звернення через ім'я структури
    struct point_t p = {10, 20};
    // Звернення через новий тип
    Point px = {10, 20};
    getch ();}
```

Тепер при роботі з типом Point немає необхідності кожного разу писати слово **struct**. Два оголошення можна об'єднати в одне

```
typedef struct point_t {
    int x;
    int y;
} Point;
```

Зауваження. Якщо ми створюємо новий тип-структуру, полем якого є покажчик на цей же тип, то його необхідно оголошувати явно з використанням службового слова **struct**

```
typedef struct Node {
    int value;
    struct Node * next;} Node;
```

Стандартом поведінку при приведенні однієї структури до іншої не визначено, навіть якщо структури мають однакові поля.

Доступ до членів структур

Коли ми визначаємо змінну, таку як ***Employee*** john, то john посилається на всю структуру. Для того, щоб отримати доступ до окремих елементів-полях, використовується оператор вибору члена (крапка). Нижче наведено

приклад використання оператора вибору членів для ініціалізації кожного поля структури:

```
Employee john; // створюємо структуру Employee для John
john.id = 8; // присвоюємо значення полю id всередині структури john
john.age = 27; // присвоюємо значення полю age всередині структури john
john.salary = 32.17; // присвоюємо значення полю salary всередині структури

Employee james; // створюємо структуру Employee для James
james.id = 9; // присвоюємо значення полю id всередині структури james
james.age = 30; // присвоюємо значення полю age всередині структури james
james.salary = 28.35; // присвоюємо значення полю salary всередині структури
```

Як і у випадку зі звичайними змінними, змінні-поля структури не ініціалізуються автоматично і зазвичай містять сміття. Ініціалізувати їх потрібно вручну.

В наведеному вище прикладі легко визначити, яка змінна відноситься до john, а яка до james. Це забезпечує набагато більш високий рівень організації, ніж у випадку зі звичайними окремими змінними.

Для звернення до поля-масиву з іменем a та індексами 3, 7 структури з іменем у слід написати:

```
y.a [3] [7]
```

Змінні-члени структури працюють так само, як і прості змінні, тому з ними можна і виконувати звичайні операції:

```
int totalAge = john.age + james.age;
if (john.salary> james.salary)
   cout << "John makes more than James \ n";
else if (john.salary <james.salary)
   cout << "John makes less than James \ n";
else
   cout << "John and James make the same amount \ n";
james.salary + = 3.75;
```

++ john.age; // використовуємо інкремент для збільшення віку John-a

Ініціалізація структур

Ініціалізація структур шляхом присвоєння значень кожному члену по порядку – заняття досить громіздке (особливо, якщо цих членів багато, в цих випадках краще задання значень полям оформлювати у вигляді окремих функцій), тому в C ++ є більш швидкий спосіб ініціалізації структур – список ініціалізаторів. Він дозволяє форматувати деякі або всі члени структури під час оголошення.

struct Ім'я_Структури Ім'я_змінної = {ЗначенняЕлемента_1, ЗначенняеЕлемента_2,. . . , ЗначенняЕлемента_n};

Приклад.

```
struct Employee{
   short id;
   int age;
   double salary;};
```

Employee john = {5, 27, 45000.0}; // john.id = 5, john.age = 27, john.salary = 45000.0

Employee james = {6, 29}; // james.id = 6, james.age = 29, james.salary = 0.0 (ініціалізація за замовчуванням)

У C ++ 11 можна також використовувати ініціалізацію uniform:

Employee john {5, 27, 45000.0}; // john.id = 5, john.age = 27, john.salary = 45000.0

Employee james {6, 29}; // james.id = 6, james.age = 29, james.salary = 0.0 (ініціалізація за замовчуванням)

Якщо в списку ініціалізаторів не буде одного або декількох елементів, то їм присвоєно значення за замовчуванням (зазвичай 0). В наведеному вище прикладі змінної james.salary присвоїти значення за замовчуванням – 0.0, так як ми самі не вказали для неї ніякого значення під час ініціалізації.

C ++ 11/14: не статична ініціалізація членів

Починаючи з C ++ 11, з'являється можливість задавати НЕ статистичним (звичайним) членам структури значення за замовчуванням:

```
struct Triangle{
    double length = 2.0;
    double width = 2.0;};
int main ()
{    Triangle z; // довжина = 2.0, ширина = 2.0
    z.length = 3.0; // Ви можете присвоювати також інші значення
    return 0;}
```

На жаль, в C ++ 11 синтаксис ініціалізації не статичних полів структури несумісний з синтаксисом списку ініціалізаторів або ініціалізації uniform. У C ++ 11 наступна програма НЕ скомпілюється:

```
struct Triangle  {double length = 2.0; // не статичною ініціалізація членів
double width = 2.0;};
int main ()
{Triangle z {3.0, 3.0}; // ініціалізація uniform
return 0;}
```

Отже, потрібно буде вирішити, чи хочете ви використовувати не статистичну ініціалізацію полів або ініціалізацію uniform. Ініціалізація uniform більш гнучка, тому ми рекомендуємо дотримуватися її.

Однак в C ++ 14 це обмеження було скасовано, і обидва варіанти можна використовувати. До C ++ 11, якби ми хотіли присвоїти значення полів структури, нам довелося б це робити вручну індивідуально кожному полю. У C++11 можна присвоювати значення членам структур, використовуючи список ініціалізаторів:

```
struct Employee{     short id;
    int age;
    double salary;};
Employee john;
```

john = {5, 27, 45000.0}; // тільки C ++ 11

Вкладені структури

В якості полів можуть виступати і структури. Такі конструкції називають вкладеними структурами, наприклад:

```
struct Employee
{   short id;
    int age;
    double salary;};
struct Company
{   Employee CEO; // Employee - це структура всередині структури Company
    int numberOfEmployees;};
Company myCompany; // опис змінної myCompany типу структура
```

В цьому випадку, якщо б ми хотіли дізнатися, яка зарплата в CEO, нам довелося б використовувати оператор вибору членів двічі:

myCompany.CEO.salary;

Спочатку ми вибираємо поле CEO зі структури myCompany, а потім поле salary зі структури Employee.

Ви можете використовувати вкладені списки ініціалізаторів для вкладених структур:

```
struct Employee
{   short id;
    int age;
    float salary;};
struct Company
{   Employee CEO; // Employee - це структура всередині структури Company
    int numberOfEmployees;};
Company myCompany = {{3, 35, 55000.0f}, 7};
```

Приклад.

Нехай задано два шаблони структур, які оголошуються в окремому файлі "MyStruct.h":

- шаблон Point, що описує точку на координатній площині:

```
struct Point{    int x;
    int y;};
```

- шаблон Triangle, що описує трикутник на площині:

```
struct Triangle
{   Point p1; //  структура
    Point p2; //  структура
    Point p3; //  структура
    char comment [50]; // коментарій до фігурі};
```

У шаблоні Triangle опісується три вкладені структури (точки) типу Point.

Демонстрація роботи зі структурою Triangle.

```
Triangle T; // для змінної T виділяється пам'ять
// Заповнення значень змінної T
T.p1.x = 25;
T.p1.y = 36;
T.p2.x = 100;
T.p2.y = 55;
T.p3.x = 60;
T.p3.y = 56;
strcpy (T.comment, "Triangle # 1");
```

Для використання методу strcpy() та підключення файлу структури потрібно на початку модуля програми вписати:

```
#include <cstring>
#include "MyStruct.h"
```

Вкладені структури ініціалізуються як багатовимірні масиви.

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

```
#include <conio.h>
typedef struct Model {
   int id;
   struct {
      int id;
      char *name; } make;
   char *name;
   unsigned char year; //year is an offset to 1890
} Model;
void main() {
   Model m = {10, {10, "Acura"}, "CL", 112};
   printf("Model name = %s\n", m.name);
   printf("Make name = %s\n", m.make.name);
   getch();}
```

Масиви структур

З структур можна створювати масиви також, як масиви інших типів. І все формати визначення масиву структур будуть аналогічні визначенням масивів інших типів:

```
struct person people [10];
```

В даному випадку визначено масив структур person з 10 елементів.

Використовуємо масив структур в програмі:

```
#include <stdio.h>
person
{    int age;
     char name [20];};
int main (void)
{ struct person people [] = {23, "Tom", 32, "Bob", 26, "Alice", 41, "Sam"};
     int n = sizeof (people) / sizeof (people [0]);
     for (int i = 0; i <n; i ++)
{ printf ("Name:% s \ t Age:% d \ n", people [i] .name, people [i] .age); }
     return 0;}
```

У масиві people визначено 4 об'єкти *person*. Хоча при ініціалізації в масив передається 8 значень {23, "Tom", 32, "Bob", 26, "Alice", 41, "Sam"},

але так як кожна структура складається з двох елементів, то відповідно з цих значень по порядку виходить 4 об'єкти *person*.

Звернення до елементів масиву структур відбувається за індексом people [0]. А щоб звернутися до елементу структури з масиву, після індексу вказується ім'я елемента структури: people [i] .name

Консольний результат виконання програми:

Name: Tom Age: 23

Name: Bob Age: 32

Name: Alice Age: 26

Name: Sam Age: 41

Як з масивами інших типів з масивами структур можна використовувати покажчики:

```
#include <stdio.h>
struct person{
   int age;
   char name [20];};
int main (void)
{  struct person people [] = {23, "Tom", 32, "Bob", 26, "Alice", 41, "Sam"};
   int n = sizeof (people) / sizeof (people [0]);
   for (struct person * p = people; p <people + n; p ++)
{printf ( "Name:% s \ t Age:% d \ n", p-> name, p-> age);}
   return 0;}
```

Тут в масиві *people* ті ж 4 елементи *person*. Для їх перегляду створено покажчик ***p***, який встановлюється на початок масиву *people*. І в циклі отримуємо елементи структур через цей покажчик. Після завершення кожної ітерації покажчик збільшується на одиницю, тобто переміщується в пам'яті на кількість байт, які займає одна структура. І ці дії тривають поки покажчик не дійде до кінця масиву, який можна отримати через вираз people + n.

Покажчики на структуру

Для отримання адреси структурної змінної слід помістити оператор **&** перед ім'ям структури. Нехай є наступний фрагмент

struct bal {

float balance;

char name [80];} ***person***;

struct bal * p; / * Оголошення покажчика на структуру * /

тоді

p = & person;

поміщає адресу структури person в покажчик p.

Для доступу до членів структури за допомогою покажчика на структуру слід використовувати оператор "стрілка". Оператор "стрілка", ->, утворений з знака "мінус" і символу "більше". Наприклад, для доступу до члена balance за допомогою p слід написати:

p-> balance

замість

p.balance

Використання полів бітів в якості полів структур.

Полями структур можуть бути, зокрема, поля бітів. Хоча правила мови не мають обмежень на характер цих полів, крім вимоги, щоб вони містилися в обсязі машинного слова, в типових застосуваннях поля бітів служать для зберігання цілих даних (частіше типу unsigned).

Опис поля бітів складається з опису типу поля, його імені та зазначеного після двокрапки розміру поля в бітах, наприклад:

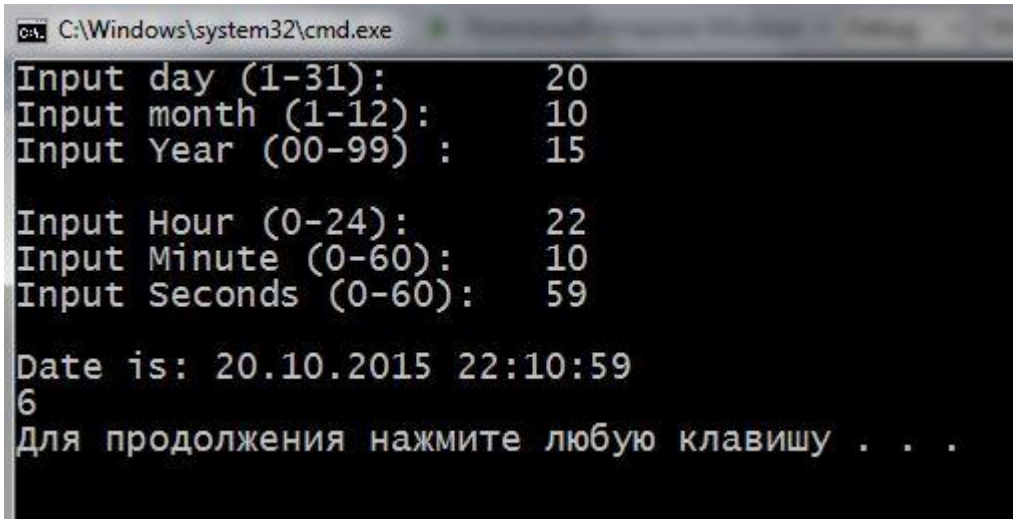
unsigned status: 6 ;.

Якщо ім'я поля опущено, то створюється приховане поле. Якщо розмір поля бітів представлений числом 0, то наступне поле бітів почнеться з кордону машинного слова.

```
#include <iostream>
using namespace std;
```

```
struct DateTime //Опишем структуру с битовыми полями
{ unsigned short Day : 5;   //5 бит для дня
 unsigned short Month : 4; //4 для месяца
 unsigned short Year : 7;  //7 для года от 0 до 99
 unsigned short Hour : 5;  //5 бит для 24-х часов
 unsigned short Minute : 6;//6 для минут
 unsigned short Second : 6;//6 для секунд};

int main()
{  DateTime d; //объявляем переменную этого типа с битовыми полями
 int i;      //И еще одну, в которую будет поступать ввод данных

 //Введем дату
 cout << "Input day (1-31):" << '\t'; cin >> i; d.Day = i;
 cout << "Input month (1-12):" << '\t'; cin >> i; d.Month = i;
 cout << "Input Year (00-99) :" << '\t'; cin >> i; d.Year = i;

 //Введем время
 cout << endl << "Input Hour (0-24):" << '\t'; cin >> i; d.Hour = i;
 cout << "Input Minute (0-60):" << '\t'; cin >> i; d.Minute = i;
 cout << "Input Seconds (0-60):" << '\t'; cin >> i; d.Second = i;

 //И выведем их с показателем размера в памяти
 cout << endl << "Date is: " << d.Day << "." << d.Month << ".20" << d.Year << "
";
 cout << d.Hour << ":" << d.Minute << ":" << d.Second << endl;
 cout << sizeof(d) << endl;
}
```

Результат:



Як бачите, бітові поля зберігають дату і час. Займає ця структура 6 байт, хоча для неї вистачить і п'яти. І на те є свої причини: сам компілятор може вирівнювати відведену пам'ять до парного числа байтів. Наприклад якщо ми замовили 18 біт, компілятор відведе нам не 3 байта, а 4, враховуючи що процесор любить працювати з байтами, а не з битами. Принаймні зберігати в пам'яті або своїх регістрах процесор вважає за краще не біти, а саме байти. Згідно з його розрядності: x32 зберігають 4 байта, x64 вже 8 байт.

Структури і функції

Великою перевагою використання структур, ніж окремих змінних, є можливість передавати всю структуру функції, яка повинна працювати з її полями.

Існує 2 способи передачі структури у функцію:

- передача структури за значенням. При такій передачі робиться копія структурної змінної в пам'яті. Якщо структура має великий розмір, то такий спосіб неефективний. Перевага цього способу в тому, що всі маніпуляції з копією структури у Функції НЕ впливають на вихідну змінну;
- передача покажчика на структуру. У цьому випадку передається тільки покажчик на структуру. Якщо структура займає великий об'єм пам'яті, то такий спосіб забезпечує швидку передачу значень структурної змінної у функцію. Це пов'язано з тим, що НЕ робиться в пам'яті копії структурної змінної. Недоліком цього способу є те, що у функції можна змінити значення вихідної структурної змінної, коли цього НЕ потрібно робити.

Так само, як і при передачі структури у функцію, існують 2 способи повернення:

- повернення структури за значенням;
- повернення покажчика на структуру.

Приклад передачі і привернення структури за значенням

```
#include <iostream>

struct Employee{
```

```
    short id;
    int age;
    double salary;};
void printInformation (Employee employee)
{ cout << "ID:" << employee.id << "\ n";
  cout << "Age:" << employee.age << "\ n";
    cout << "Salary:" << employee.salary << "\ n";}
int main ()
{ Employee john = {21, 27, 28.45};
    Employee james = {22, 29, 19.29};
    // Виводимо інформацію про John
    printInformation (john);
    cout << "\ n";
    // Виводимо інформацію про James
    printInformation (james);
    return 0;}
```

В наведеному вище прикладі ми передали структуру Employee функції printInformation (). Це дозволило нам не передавати кожну змінну окремо. Більш того, якщо ми коли-небудь захочемо додати нових членів в структуру Employee, нам не доведеться змінювати оголошення або виклик функції!

Результат програми вище:

ID: 21

Age: 27

Salary: 28.45

ID: 22

Age: 29

Salary: 19.29

Функція також може повертати структуру. Це один з тих небагатьох випадків, коли функція може повертати декілька змінних.

```
#include <iostream>
struct Point3d{
   double x;
   double y;
   double z;};
Point3d getZeroPoint ()
{ Point3d temp = {0.0, 0.0, 0.0};
   return temp;}
int main ()
{ Point3d zero = getZeroPoint ();
   if (zero.x == 0.0 && zero.y == 0.0 && zero.z == 0.0)
      cout << "The point is zero \ n";
   else
      cout << "The point is not zero \ n";
   return 0;}
```

результат:

The point is zero

Приклад передачі структури у функцію за покажчиком

```
// передача покажчика на структуру MyPoint
bool EqualPointsP(MyPoint * p1, MyPoint * p2)
{  bool f = false;
   if ((p1->x == p2->x) && (p1->y == p2->y))
      f = true;
   return f;}
```

Програмний код, який демонструє використання функції EqualPointsP().

```
// Передача структури за покажчиком
MyPoint p1;
MyPoint p2;
bool f_equal;
p1.x = 28;
p1.y = 35;
p2.x = 28;
```

```
p2.y = 35;
f_equal = EqualPointsP(&p1, &p2); // f_equal = true
p2.y = 100;
f_equal = EqualPointsP(&p1, &p2); // f_equal = false
```

Приклад передачі si повернення структури за покажчиком

Розглянемо приклад зі структурою Size. Треба написати функцію, в якій користувач вносить дані в елементи структури.

```
#include<iostream>
using namespace std;
struct Size{
 int breast;
 int waist;
 int hips;};

void setSizes(Size* Obj);
void showSizes(const Size* Obj);

int main()
{ setlocale(LC_ALL, "rus");
Size* Volume = new Size; // виділення пам’яті
cout << "Введіть обє’ми: ";
 setSizes(Volume);
 showSizes(Volume);
Volume->waist = 62;
 Volume->hips = 92;
 showSizes(Volume);
    delete Volume;
return 0;}

void setSizes(Size* Obj)
{ cin >> Obj->breast;
 cin >> Obj->waist;
 cin >> Obj->hips;
 cout << endl;}

void showSizes(const Size* Obj)
{  cout << "Объемы = ";
 cout << Obj->breast << '/' << Obj->waist << '/' << Obj->hips << endl;}
```

Оголошуючи покажчик на структуру, не забувайте про те, що треба виділити пам'ять під неї

Приклад повернення з функції покажчика на структуру

```
// структура
struct MyPoint
{    int x;
   int y;};
```

Нехай реалізовано функцію GetCenterLineP(), яка отримує 2 точки і повертає покажчик на структуру. У тілі функції реалізовано виділення пам’яті для структури оператором new.

```
MyPoint * GetCenterLineP(MyPoint p1, MyPoint p2)

{   MyPoint * resP; // покажчик на структуру MyPoint

   // виділення пам'яті для структури динамічно
   resP = new MyPoint;
   // обчислення середини відрізку - заповнення значеннями
   resP->x = (p1.x + p2.x)/2;
   resP->y = (p1.y + p2.y)/2;
   return resP;}
```

Демонстрація роботи з функцією GetCenterLineP().

```
// повернення native-структури за покажчиком
MyPoint * resPt; // покажчик на структуру –
                 // пам’ять для структури ще не  виділена
MyPoint pt1, pt2; // екземпляри структури
pt1.x = 28;
pt1.y = 40;
pt2.x = 38;
pt2.y = 50;
// виклик функції GetCenterLineP()
// для покажчика resPt пам'ять виділяється всередині функції
resPt = GetCenterLineP(pt1, pt2);
// перевірка
int d;
d = resPt->x; // d = 33
d = resPt->y; // d = 45
```

Покажчики на поля структури і на вкладені структури

Покажчики на поля структури визначаються також, як і звичайні покажчики. Покажчики на вкладені структури можливі тільки тоді, коли структура визначена. Трохи переробимо попередній приклад і візьмемо покажчики на поля структури Model:

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define YEAR_OFFSET 1890
//Окремо опишемо структуру "Марка"
typedef struct Make {
   int id;
   char *name;
} Make;

//Тепер полем структури "Модель" є структура "Марка"
typedef struct Model {
   int id;
   Make make;
   char *name;
   unsigned char year; //year is an offset to 1890
} Model;

char* mallocByString(const char *str) {
// Виділення блоку пам’яті розміром strlen(str) і встановлення покажчика на
// його початок
   char* p = (char*) malloc(strlen(str) + 1);
//Копіювання рядка str у рядок на який вказує покажчик p
   strcpy(p, str);
   return p;}

void freeModel(Model* model) {
   free(model->make.name);
   free(model->name);}

void main() {
   Make *make = NULL;
   Model cl;
   int *id;
   cl.id = 2;
   cl.make.id = 1;
   cl.make.name = mallocByString("Acura");
   cl.name = mallocByString("CL");
   cl.year = (2003 - YEAR_OFFSET);
   // Отримання покажчика на вкладену структуру
   make = &cl.make;
   //Отримання покажчика на поле структури
   id = &cl.id;
   printf("make.name = %s\n", make->name);
```

```
printf("make.id = %d\n", make->id);
printf("model.id = %d\n", *id);
freeModel(&cl); //Вивільнення пам’яті
scanf("1");}
```

Операції над структурами

Допустимою є операція присвоєння даних типу структура одного типу, наприклад:

Mystruct x,y;

x=y;

В даному випадку значення полів структури ***y*** копіюються у відповідні поля структури ***x***.

Як ми вже розглядали вище можна створювати покажчики на структуру та брати адресу.

Розглянемо виконання деяких найпоширеніших операцій над елементами структури на приклады наступного опису:

```
struct {
int x;
int *y;} *p; // покажчик на структуру
```

Тоді:

(++p)->x – операція збільшує p до доступу до x

(p++)->x – операція збільшує p після доступу до x (круглі дужки необов’язкові так як спочатку за пріоритетом виконується операція ->)

*p->y – вибирає значенняобєкту на який вказує y

*p->y++ – збільшує y після обробки того, на що він вказує (аналогічно до c**)

*p++->y – збільшує p після вибірки того, на що вказує y

(*(p).y)++ – збільшує те, на що вказує y