Тема: МЕХАНІЗМ ІНІЦІАЛІЗАЦІЇ ПОКАЖЧИКІВ

1. Домовленість про використання нульових покажчиків

Після того, як покажчик був оголошений, але до того, як йому було присвоєне якесь значення, покажчик містить невідоме значення. Спроба використати покажчик до присвоєння йому якогось значення є неприємною помилкою, оскільки вона може порушити роботу не тільки Вашої програми, але й операційної системи. Навіть якщо цього не сталося, результат роботи програми буде неправильним і знайти цю помилку буде достатньо складно.

Вважають, що покажчик, який вказує в «нікуди», має мати значення null, однак і це не робить його «безпечним». Після того, як він потрапить в праву або ліву частину оператора присвоєння, то він знову може стати «небезпечним» . З іншого боку нульовий покажчик можна використати, наприклад, для позначення кінця масиву покажчиків.

Якщо була спроба присвоїти яке-небудь значення тому, на що вказує покажчик з нульовим значенням, система видає попередження, що з'являється під час роботи програми (або після завершення роботи програми) «Null pointer assignment». Поява цього повідомлення є мотивом для пошуку використання неініціалізованого покажчика в програмі.

Оголошений, але не ініціалізований покажчик міститиме довільне значення. Під час спроби використати покажчик до присвоєння йому конкретного значення можна зруйнувати не тільки власну програму, але навіть і операційну систему (найгірший, треба сказати, тип помилки!). Оскільки не існує гарантованого способу уникнути використання неініціалізованого покажчика, то С++-програмісти прийняли процедуру, яка дає змогу уникати таких жахливих помилок. За домовленістю, якщо покажчик містить нульове значення, то вважається, що він ні на що не посилається. Це означає, що у випадку присвоєння нульового значення всім невживаним покажчикам і уникати його використання, то можна уникнути випадкового використання

неініціалізованого покажчика. Рекомендуємо і Вам дотримуватися цієї практики програмування.

Під час оголошення покажчик будь-якого типу можна ініціалізувати нульовим значенням, наприклад, як це робиться в такій настанові.

```
float *p = 0; // p -- тепер нульовий покажчик.
```

Для тестування покажчика використовують настанову Її(будь-який з таких її варіантів):

```
if(p)     // Виконуємо щось, якщо p -- не нульовий покажчик.
if(!p)    // Виконуємо щось, якщо p -- нульовий покажчик.
```

Дотримуючись згаданої вище домовленості про нульові покажчики, Ви можете уникнути багатьох серйозних проблем, що виникають під час їх використання.

2. Покажчики і 16-розрядні середовища

На сьогодні більшість обчислювальних середовищ є 32-розрядними, однак раніше немало користувачів працювали в 16-розрядних (в основному, це DOS і Windows 3.1) і, природно, з 16-розрядним кодом. Ці операційні системи були розроблені для процесорів серії Intel 8086, які містять такі модифікації, як 80286, 80386, 80486 і Pentium (під час роботи в режимі емуляції процесора 8086). І хоча під час написання нового коду програми програмісти, як правило, орієнтуються на використання 32-розрядного середовища виконання, однак раніше створені програми підтримують компактні 16-розрядні середовища. Позаяк деякі теми актуальні тільки для 16-розрядних середовищ, то програмістам, які працюють в них, буде корисно отримати інформацію про те, як адаптувати «старий» програмний код до нового середовища, тобто переорієнтовувати 16-розрядний код на 32-розрядний.

Під час написання 16-розрядноші коду програми для процесорів серії Intel 8086 програміст має право розраховувати на шість способів компілювання програм, які відрізняються організацією комп’ютерної пам’яті. Програми можна компілювати для мініатюрної, малої, середньої, компактної, великої і

величезної моделей пам’яті. Кожна з цих моделей по-своєму оптимізує простір, що резервується для даних, коду програми і стека. Відмінність в організації комп’ютерної пам’яті пояснюється використанням процесорами серії Intel 8086 сегментованої архітектури у процесі виконання 16-розрядного коду програми. У 16-розрядному сегментеваному режимі процесори серії Intel 8086 ділять пам’ять на 16К сегментів.

У деяких випадках модель пам’яті може вплинути на поведінку покажчиків і Ваші можливості з їх використання. Основна проблема виникає під час інкрементування покажчика за межі сегмента. Головне, слід знати, коли доведеться працювати в 16-розрядному середовищі і орієнтуватися на процесори серії Intel 8086, програміст повинен вивчити документацію, що додається до використовуваного Вами компілятора, і детально розібратися в моделях пам’яті та їх впливі на покажчики.

Під час написання програм для сучасного 32-розрядного середовища необхідно знати, що в ній використано єдину модель організації пам’яті, яка називається *однорівневою, несегментованою* або *лінійною* (flat model).

3. Багаторівнева непряма адресація

Можна створити покажчик, який посилатиметься на інший покажчик, а той – на кінцеве значення. Цю ситуацію називають *багаторівневою непрямою адресацією* (multiple indirection) або використанням *покажчика на покажчик*. Ідея багаторівневої непрямої адресації схематично проілюстрована на рис. 6.2. Як бачимо, значення звичайного покажчика (при однорівневій непрямій адресації) є адресою змінної, яка містить певне значення. У разі застосування покажчика на покажчик перший містить адресу другого, а той посилається на змінну, що містить певне значення.

Рис. Однорівнева і багаторівнева непряма адресація

Під час використання непрямої адресації можна організувати будь-яку бажану кількість рівнів, але, як правило, обмежуються тільки двома, оскільки збільшення їх кількості часто веде до виникнення концептуальних помилок Змінну, яка є покажчиком на покажчик, потрібно оголосити відповідним чином. Для цього достатньо перед її іменем поставити додатковий символ «зірочка» (*). Наприклад, таке оголошення повідомляє компіляторові про те, що balance - покажчик на покажчик на значення типу int:

```
int **balance;
```

Необхідно пам’ятати, що змінна balance тут – не покажчик на цілочисельне значення, а покажчик на покажчик на int-значення.

Щоб отримати доступ до значення, яке адресується покажчиком на покажчик, необхідно двічі застосувати оператор як це показано в такому короткому прикладі.

Код програми 6.12. Демонстрація механізму використання багаторівневої непрямої адресації

```
#include <iostream>          // Потокове введення-виведення
using namespace std;         // Використання стандартного простору імен

int main()
```

```
{
        int x, *p, **q;
        x = 10; p = &x; q = &p;
        cout << **q;                    // Виводимо значення змінної x.
        getch(); return 0;
}
```

У цьому коді програми змінна **p** оголошена як покажчик на int-значення, а змінна **q** - як покажчик на покажчик на int-значення. У процесі виконання цієї програми ми набудемо значення змінної **x**, тобто число 10.

Контрольні запитання:

1. Яка є домовленість про використання нульових покажчиків?
2. Охарактеризуйте покажчики і 16-розрядні середовища.
3. Що таке багаторівнева непряма адресація?