1.6 Поняття вказівника

Будь-який об'єкт програми, будь то змінна базового або похідного типу, займає в пам'яті певну область пам'яті. Місце розташування об'єкта в пам'яті визначається його **адресою**. При оголошенні змінної для неї резервується місце в пам'яті, розмір якого залежить від типу даної змінної, а для доступу до вмісту об'єкта служить його ім'я (ідентифікатор). Для того щоб дізнатись адресу конкретної змінної, служить унарна операція взяття адреси. При цьому перед іменем змінної ставиться знак амперсанта (**&**). Наступний нижче приклад програми виведе на друк спочатку значення змінних **x** і **y**, а потім їх адреси.

Приклад:

```
#include <iostream.h>
void main ()
{
int x = 10, y = 20;
// виводить на екран Value x = 10
cout << "Value x = " << x << endl;
// виводить на екран Adress x = 0x0012FF7C
cout << "Adress x = " << &x << endl;
// виводить на екран Value y = 20
cout << "Value y = " << y << endl;
// виводить на екран Adress y = 0x0012FF78
cout << "Adress y = " << &y << endl;
}
```

Як видно із прикладу, локальні змінні розташовуються у стеці у зворотному порядку (стек росте в напрямку молодших адрес). Результат може відрізнятися навіть при повторному запуску програми, тому що неможливо вгадати, по якій адресі почнуть розміщатися змінні. Важливо інше: різниця в адресах першої й другої змінної завжди буде однакова й складе 4 байта.

Потужним засобом розробника програмного забезпечення на С++ є можливість здійснення безпосереднього доступу до пам'яті. Для цієї мети передбачається спеціальний тип змінних – вказівники.

Вказівник (**pointer**) – це змінна, яка в якості значення зберігає адресу іншої змінної. Вказівник може посилатися на змінну (базового або похідного типу) або на функцію. Найбільша ефективність застосування вказівників у розробці додатків досягається при використанні їх з масивами й символьними рядками.

Синтаксис оголошення вказівника:

тип_об'єкта* ідентифікатор ;

Тут **тип_об'єкта** визначає тип даних, на які посилається вказівник з іменем **ідентифікатор**. Символ 'зірочка' (*****) повідомляє компілятор, що оголошена змінна є вказівником і не залежно від того, скільки пам'яті потрібно відвести під сам об'єкт, для вказівника резервується два або чотири байти залежно від моделі пам'яті, що використовується.

Оскільки вказівник являє собою посилання на деяку область пам'яті, йому може бути присвоєна тільки адреса деякої змінної (або функції), а не саме її значення. У випадку некоректного присвоєння компілятор видасть відповідне повідомлення про помилку.

Приклад оголошення і ініціалізації вказівника:

```
char symbol = 'Y';
char *psymbol = &symbol;
long capital = 304L;
long *plong;
plong = &capital;
```

У наведеному фрагменті оголошується символьна змінна **symbol** і ініціалізується значенням **'Y'**, потім визначається вказівник на символьний тип даних **psymbol**, значення якого призначається рівним адресі змінної **symbol**. Слідом за цим оголошується змінна **capital** типу **long** і вказівник на цей же тип **plong**, після чого проводиться ініціалізація вказівника адресою змінної **capital**.

Зверніть увагу на те, що символ "*" в операторі оголошення змінної вказує, що ця змінна є вказівником. Слід уникати запису оператора в рядку 4. Краще відразу проіцізіалізувати вказівник `long* plong =NULL;`

1.6.1 Розіменування вказівників

Вказівники допомагають здійснювати безпосередній доступ до пам'яті. Для того, щоб одержати (прочитати) **значення**, записане в деякій області, на яку посилається вказівник, використовують операцію **розіменування** (*****). При цьому використовується ім'я вказівника із зірочкою перед ним:

```
double d1, d2 = 10;
double *ptr = &d2;
d1 = *ptr;
cout << d1;
```

У наведеному фрагменті оголошуються дві змінні типу double: **d1** і **d2** і вказівник (**ptr**) на тип **double**, проініціалізований адресою змінної **d2**. Після цього за допомогою непрямого доступу до змінної **d1** присвоюється значення, що зберігається за адресою, зазначеною

в **ptr**, тобто фактично значення змінної **d2**, що й підтверджує вивід на друк.

Ще раз зверніть увагу на те, що символ "*" в операторі оголошення змінної вказує, що ця змінна є вказівником. В усіх інших випадках наявність "*" – це операція розіменування.

1.6.2 Порожній вказівник

На практиці досить широко застосовується так званий **порожній вказівник** (типу **void**), який може вказувати на об'єкт будь-якого типу. Для отримання доступу до об'єкта, на який посилається вказівник **void**, його необхідно попередньо привести до того ж типу, що і тип самого об'єкта.

Приклад:

```
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
   char A = 'q';
   void *ptr;
   ptr = &A;
   *(char*) ptr = 'a';
   cout << "ptr=" << *(char*)ptr << "\na="<<A;
   system("pause");
   return 0;
}
```

1.6.3 Арифметика вказівників

До вказівників (крім вказівників на змінні типу **void**) можуть застосовуватися арифметичні операції:

– вказівники можна віднімати один від одного, тим самим, визначаючи кількість елементів (того ж типу, на який указують вказівники), розташованих між ними. Це корисно при операціях з масивами і символьними рядками;

– від вказівника можна віднімати (або додавати до нього) якенебудь число (ці операції мають сенс тільки для масивів);

– до вказівників можна застосовувати операції інкремента або декремента (ці операції мають сенс тільки для масивів);

– вказівники можна використовувати в операціях порівняння.

Для зміни порожнього вказівника він повинен бути попередньо приведений до якого-небудь типу (не **void**).

Компілятор, знаючи тип вказівника, обчислює розмір змінної цього ж типу, після чого модифікує адресу, що зберігається у вказівнику, відповідно до заданої арифметичної операції, але з урахуванням обчисленого для даного типу розміру. Це означає, що якщо оголошений вказівник типу **double**, на змінну, що займає в пам'яті 8 байт, операція, наприклад, інкремента вказівника збільшить значення адреси не на один, а на вісім байт:

Приклад 1:

Оголошено вказівник

```
double *ptr=NULL;
```

якимсь чином у вказівник **ptr** було записано адресу 100, після виконання оператора інкремента

```
ptr++;
```

ptr буде містити адресу `100 + sizeof(double)`
(або 108 для наших комп'ютерів)

Приклад 2:

Оголошено вказівник

```
int *ptr=NULL;
```

якимсь чином у вказівник **ptr** було записано адресу 100, після виконання оператора інкремента

```
ptr++;
```

ptr буде містити адресу адресу 100 + **sizeof**(**int**) ;
(або 104 для наших комп'ютерів)

Приклад 3:

```
double d;
double *ptr1 = &d;
cout << "\nptr1=" << ptr1;                    //0012ff50
ptr1++;
cout << "\nptr1+1=" << ptr1<<'\n';           // 0012ff58
int a = 4;
int *ptr1 = &a,*ptr2 = &a;
if (ptr1 == ptr2) cout<<"yes";          //буде надруковано yes
```

1.6.4 Константні вказівники та вказівники на константу

При оголошенні вказівників можна використовувати ключове слово **const**. Застосування даного специфікатора трактується компілятором у такий спосіб.

Синтаксис вказівника на константу:

const тип* ідентифікатор ;

Розіменоване значення незмінне. Спроба модифікації вміст вказівника на константу приведе до повідомлення про помилку, тобто не можна написати *ptr=3;

Синтаксис константного вказівника:

тип* const ідентифікатор ;

Завжди вказує на ту саму адресу. Застосування будь-якої арифметичної операції до константного вказівника, приведе до повідомлення про помилку.

Приклад:

```
double f, f1;
//обов'язково присвоїти початкове значення
double *const ptr = &f;
ptr = &f1;                                  // не можна
```

1.6.5 Застосування до вказівників оператора sizeof

Як і до будь-якої змінної або типу даних, до вказівників можна застосовувати операцію визначення розміру **sizeof**. Вище зазначалося, що розмір вказівника може приймати одне із двох значень: два або чотири байти, що дозволяє вказівнику адресувати $2^{(2\times8)}$ = 65 Кбайт або $2^{(4\times8)}$ = 4 Гбайта пам'яті відповідно. На розмір вказівника (2 або 4 байта) впливає обрана модель пам'яті і ряд інших причин.

Приклад:

```
long X = 546213L ;
long* px = &X ;
// виводить на екран розмір вказівника 2 або 4 байта
cout << sizeof ( px ) << '\n' ;
```

1.6.6 Вказівники на вказівники

Вказівники можуть самі посилатися на інші вказівники. При цьому в пам'яті, на яку посилається вказівник, утримується не значення, а адреса якого-небудь об'єкта. Сам об'єкт може також бути вказівником і т. д.

Синтаксис вказівника на вказівник:

тип ідентифікатор ;**

При оголошенні вказівник на вказівник може ініціалізуватися адресою об'єкта. Число символів "зірочка" (*) при оголошенні говорить про "порядок" вказівника.

Щоб одержати доступ до значення, такий вказівник повинен бути розіменований відповідну кількість разів (по числу символів ' * ').

Приклад:

```
// оголошення змінної типу char
char s = 'H' ;
// оголошення вказівника типу char і його ініціалізація
char* ps = &s ;
```

```
// оголошення вказівника на вказівник і його ініціалізація
   char** pps = &ps ;
// оголошення вказівника третього порядку і його ініціалізація
   char*** ppps = &pps ;
// модифікує значення змінної s
   ***ppps = 'L' ;
// виводить на екран значення змінної s
   cout << ***ppps ;
```

1.6.7 Вказівники на функції

У С++ вказівники можуть посилатися на функції. Ім'я функції саме по собі представляє константний вказівник на цю функцію, тобто містить адресу входу в неї. Однак можна задати свій власний вказівник на дану функцію:

тип (*ім'я_вказівника) (список_типів_аргументів)

Приклад:

```
bool ( *funcPtr ) ( char, long );
```

оголошує вказівник **funcPr**, що посилається на функцію, яка повертає логічне значення, та приймає в якості параметрів одну символьну і одну цілу довгу змінну.

Виклик функції через вказівник здійснюється так, начебто ім'я вказівника є просто іменем функції, що викликається. Тобто, після імені вказівника вказується список аргументів, очікуваних функцією. Так, для наведеного вище вказівника на функцію **funcPtr** виклик може виглядати, наприклад, у такий спосіб:

```
bool flag;
char symbol = 'x';
long number = 202L;
flag = funcPtr ( symbol, number );
```

Вказівники на функції використовуються часто в якості аргументів інших функцій. У такий спосіб створюються універсальні функції, що значно спрощують текст складної програми (наприклад, чисельне рішення рівнянь, диференціювання, інтегрування). Деякі бібліотечні функції, як параметр, також приймають вказівники на функції.

Приклад функції (**common**), яка приймає в якості параметра вказівник на іншу функцію. Таким чином, результат роботи функції залежить від того, яка функція передається в якості параметра. Головна вимога – сігнатури функцій, які передаються в якості параметра, повинні співпадати.

Приклад:

```
#include <iostream>
```

```
using namespace std;
void common(int(*arifmetic)(int a,int b),int a, int b)
{
   cout << arifmetic(a,b) << '\n';
}

int sum(int a, int b)
{
return a+b;
}

int subtraction(int a, int b)
{
return a-b;
}

int _tmain(int argc, _TCHAR* argv[])
{
   common(sum,2,3);
   common(subtraction,2,3);
   system("pause");
   return 0;
}
```

1.6.8 Посилання

Посилання – особливий тип даних, що є схованою формою вказівника, який при використанні автоматично розіменовується. Іншими словами, він може використовуватися просто як інше ім'я, або псевдонім об'єкта. При оголошенні посилання перед іменем змінної ставиться знак амперсанта, а сама вона повинна бути відразу проініціалізована іменем того об'єкта, на який посилається:

тип &ім'я_посилання = ім'я_змінної;

Тип об'єкта, на який вказується посилання, може бути будь-яким. Оголошення неініціалізованого посилання викличе повідомлення компілятора про помилку (крім ситуації, коли посилання оголошується як **extern**). Будь-яка зміна значення посилання спричинить зміну того об'єкта, на який дане посилання вказує.

Приклад:

```
int v = 0;          // оголошення змінної типу int
// оголошення посилання на змінну типу int і її ініціалізація
int &ref = v;
ref += 10;          // те ж, що й v += 10
```

Після виконання наведеного фрагмента значення обох змінні **v** і **ref** буде дорівнювати 10. Використання посилань не пов'язане з додатковими витратами пам'яті.

Посилання не можна перепризначати. Спроба перепризначити наявне посилання якій-небудь іншій змінній приведе до присвоєння оригіналу об'єкта значення іншої змінної.

Приклад:

```
// оголошення змінних a і b типу char їх ініціалізація
   char a = 'A', b = 'B';
// оголошення посилання на змінну a і її ініціалізація
   char &refa = a;
// присвоєння посиланню refa значення змінної b
   refa = b;
   cout << refa;              // виводить на екран символ 'B'.
```

Крім того, слід врахувати, що посилатися можна тільки на сам об'єкт. Не можна оголосити посилання на тип об'єкта. Ще одне обмеження, що накладається на посилання, полягає в тому, що вони не можуть вказувати на нульовий об'єкт, тобто на об'єкт, який має значення **null**. Таким чином, якщо є ймовірність того, що об'єкт в результаті роботи додатка стане нульовим, від посилання слід відмовитися на користь застосування вказівника.

1.6.9 Функції. Передача параметрів за посиланням та за значенням

Параметри у функцію можуть передаватися одним із наступних способів:

- за значенням;
- за посиланням.

При передачі аргументів **за значенням** компілятор створює тимчасову копію об'єкта, який повинен бути переданий, і розміщує її в області стекової пам'яті, призначеної для зберігання локальних об'єктів. Функція, що викликається, оперує саме із цією копією, не впливаючи на оригінал об'єкта.

Приклад:

```
void swap(int x, int y)
{
   int temp = x;
   x = y;
   y = temp;
   cout << "x=" << x << "y=" << y << '\n';
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
   int a = 3, b = 4;
   swap(a,b);
   cout << "a=" << a << "b=" << b << '\n';
   system("pause");
   return 0;
}
```

У результаті виконання оператора

```
cout<<"x="<<x<<"y="<<y<<'\n';
```

у функції **swap** на друк буде виведено:

```
x = 4 y = 3
```

а після повернення у функцію **_tmain** в результаті виконання оператора

```
cout << "a=" << a << "b=" << b << '\n';
```

на друк буде виведено:

```
a = 3 b = 4
```

оскільки у функцію передавались не змінні **a** та **b** , а їх копії.

Якщо ж необхідно, щоб функція модифікувала оригінал об'єкта, використовується передача параметрів **за посиланням**. При цьому в функцію передається не сам об'єкт, а тільки його адреса. Таким чином, всі модифікації в тілі функції переданих їй за посиланням аргументів впливають на об'єкт. Беручи до уваги той факт, що функція може повертати лише єдине значення, використання передачі адреси об'єкта виявляється досить ефективним способом роботи із великим за обсягом числом даних. Крім того, тим, що передається адреса, а не сам об'єкт, суттєво заощаджується стекова пам'ять.

У C++ передача за посиланням може здійснюватися двома способами:

- використовуючи безпосередньо посилання;
- за допомогою вказівників.

Синтаксис передачі з використанням посилань має на увазі застосування як аргумент посилання на тип об'єкта. **Наприклад**, функція

```
double glue ( long& x, int& y );
```

одержує два посилання на змінні типу **long** і **int**. При передачі у функцію параметра за посиланням компілятор автоматично передає у функцію адресу змінної, зазначеної в якості аргумента. Ставити знак амперсанта перед аргументом у виклику функції не потрібно. Наприклад, для попередньої функції виклик з передачею параметрів за посиланням виглядає в такий спосіб:

```
c = glue (a, b);
```

Приклад прототипу функції при передачі параметрів через вказівник:

```
void setnumber ( int*, long* );
```

Тоді виклик функції має наступний вигляд:

```
setnumber (&n, &a);
```

Приклад функції, яка приймає в якості параметра дві змінні, та міняє їх місцями (параметри передаються з використанням посилань):

```
void swap(int &x, int &y)
{
   int temp = x;
   x = y;
   y = temp;
   cout << "x=" << x << "y=" << y << '\n';
}

int _tmain(int argc, _TCHAR* argv[])
{
   int a = 3,b = 4;
   swap(a, b);
   cout << "a=" << a << "b=" << b << '\n';
   system("pause");
   return 0;
}
```

Приклад функції, яка приймає в якості параметра дві змінні, та міняє їх місцями (параметри передаються з використанням вказівників):

```
void swap (int *x, int *y)
{
   int temp = *x;
   *x = *y;
   *y = temp;
   cout << "x=" << *x << "y=" << *y << '\n';

}
int _tmain(int argc, _TCHAR* argv[])
{
   int a = 3, b = 4;
   swap (&a, &b);
   cout << "a=" << a << "b=" << b << '\n';
   system("pause");
   return 0;
}
```

Якщо передати параметри за значенням, зміни не будуть збережені, бо в функцію буде передано копії змінних, а не їх адреси.

Крім того, функції можуть повертати не тільки значення деякої змінної, але й вказівник або посилання. Наприклад, функції, прототип яких:

```
int* count ( int );
int& increase ( );
```

повертають вказівник і посилання відповідно на цілу змінну типу **int**. Слід мати на увазі, що повернення посилання або вказівника з функції може привести до проблем, якщо змінна, на яку робиться посилання, вийшла з області видимості.

Приклад:

```
int& func () { int x ; return x ; }
```

У цьому випадку спроба повернути посилання на локальну змінну **x** приведе до помилки, яка з'ясується тільки в ході виконання програми.

Ефективність передачі адреси об'єкта замість самої змінної відчутна і у швидкості роботи, особливо, якщо використовуються великі об'єкти, зокрема масиви.

Якщо потрібно у функцію передати досить великий об'єкт, однак його модифікація не передбачається, на практиці використовується передача константного вказівника. Даний тип виклику припускає використання ключового слова **const**, наприклад, функція

```
int* const fname ( int* const number )
```

приймає і повертає вказівник на константний об'єкт типу **int**. Будь-яка спроба модифікувати такий об'єкт у межах тіла функції, що викликається, викличе повідомлення компілятора про помилку.

Приклад, що ілюструє використання константних вказівників:

```
#include <iostream>
using namespace std;
int* const call ( int* const );
/* прототип функції, яка приймає й повертає константний вказівник типу int*/
void main ()
{
int x = 13;                    // оголошення змінної x, її ініціалізація
// оголошення вказівника на змінну x, його ініціалізація
int* px = &x;
call ( px );                   // виклик функції call
system("pause");
}
```

```
int* const call ( int* const x )      // заголовок функції
{
   int k = 88;
// виводить на екран значення змінної, на яку вказує вказівник x
   cout << *x;
//x = &k;                  // не можна модифікувати об'єкт!
   *x = 99;                 //все добре
   return x;
}
```

Замість наведеного вище синтаксису константного вказівника як альтернативи при передачі параметрів можна використовувати константні посилання, що мають той же зміст, що й константні вказівники.

Приклад:

```
/* заголовок функції, яка приймає й повертає константне посилання типу int */
const int& call ( const int& x )
{
/* виводить на екран значення змінної, на яку посилається посилання */
cout << x ;
// x++ ;                    // не можна модифікувати об'єкт!
return x ;
}

void main ()
{
SetConsoleOutputCP(1251);
int y = 13 ;                // оголошення змінної x, її ініціалізація
// оголошення посилання на змінну x і її ініціалізація
int& rx = y ;
// виклик функції call
cout << "\nПовертаємо значення=" << call ( rx ) ;
system("pause");
}
```

1.6.10 Контрольні питання

1. Дайте визначення вказівника.
2. Для чого використовується розіменування вказівника?
3. Синтаксис оголошення вказівників.
4. Що таке ім'я функції в C++?
5. Чи можна передати функцію в якості параметра?

6. Які арифметичні операції можна виконувати над вказівниками?
7. Що таке константний вказівник?
8. Що таке вказівник на константу?
9. Якими способами можна передавати параметри у функцію?

1.6.11 Завдання для самостійної роботи

1. Напишіть фрагмент коду, який демонструє 3 способи передачі параметрів у функцію.
2. Наведіть приклад коду, який демонструє передачу функції в якості параметру.
3. Наведіть приклад коду, який демонструє арифметику вказівників.