Лекція №7. Вказівники та посилання. Динамічні масиви

ПРОГРАМУВАННЯ ТА ПРИКЛАДНІ ІНФОРМАЦІЙНІ СИСТЕМИ

Вказівник

Вказівники – це змінні, значеннями яких є адреси пам’яті. Якщо змінна безпосередньо посилається на своє значення, то вказівник посилається на значення змінної не безпосередньо або непрямо. Він тільки володіє значенням пам’яті імені відповідної йому змінної. Посилання на значення змінної через вказівник називається непрямою адресацією.

Вказівники, перед тим як будуть використовуватися в ході програми, повинні бути визначені. Опис змінних типу вказівник здійснюється за допомогою операторів наступної форми:

<тип> *<ім'я вказівника на змінну заданого типу>;

Опис вказівників

Приклад 1. Опис вказівників.

```
char *ptrc; //вказівник на змінну символьного типу

float *ptrf; //вказівник на змінну з плаваючою точкою

int *countPtr,  count;
```

Такий спосіб оголошення вказівників виник внаслідок того, що змінні різних типів займають різну кількість комірок пам'яті. При цьому для деяких операцій з вказівниками необхідно знати об'єм відведеної пам'яті. Операція * в деякому розумінні є оберненою до операції &.

Спочатку вказівник ініціалізується нульом, або макросом NULL, який знаходиться в директиві процесора стандартної бібліотеки С - <stddef.h>, яка включається в інші директиви, наприклад: в директиву <stdio.h>.

Змінні типу вказівник

В мові С++ є операція визначення адреси — **&**, за допомогою якої визначається адреса комірки пам’яті, що містить задану змінну. Наприклад, якщо **vr** — ім’я змінної, то **&vr** — адреса цієї змінної

В С++ також існують і змінні типу вказівник. Значенням змінної типу вказівник є адреса змінної або об'єкта. Нехай змінна типу вказівник має ім'я **ptr**, тоді в якості значення їй можна присвоїти адресу за допомогою наступного оператора:

```
ptr=&vr;
```

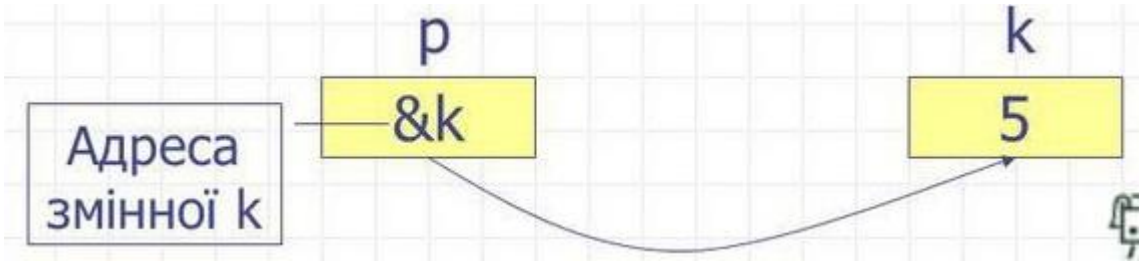
В мові С++ при роботі з вказівниками велике значення має операція непрямої адресації — *. Операція * дозволяє звертатися до змінної не напряму, а через вказівник, який містить адресу цієї змінної. Нехай **ptr** — вказівник, тоді ***ptr** — це значення змінної, на яку вказує **ptr**.

Змінні типу вказівник

- для типу T тип T* - вказівник на T
- змінна T* містить адресу об’єкту T

```
int k=5;

int *p = &k;
```



Розіменування вказівника

int *a // оголошення змінної типу вказівник

*a=4 // розіменування вказівника

Приклад 2.

```
int i = 3; //змінна типу int
int *p; // вказівние на змінну типу int
p = &i; // вказівник p посилається на змінну i
*p = 10; // змінюється комірка за адресою p

cout << i;
```

Посилання

Посилання (reference) являє собою видозмінену форму вказівника, яка використовується в якості псевдоніму (другого імені) змінної. У зв’язку з цим посилання не потребують додаткової пам’яті. Для визначення посилання використовують символ **&** (амперсант), який ставиться перед змінною-посиланням.

Приклад 3. Використання посилань.

```
int t = 13;
int &r = t;// ініціалізація посилання на t
   // тепер r синонім імені t
cout << "Початкове значення t:" << t; // виводить 13
r += 10; // зміна значення t через посилання
cout << "\n Остаточне значення t:" << t; // виводить 23
```

Вказівники та масиви

Вказівники використовуються для роботи з масивами. розглянемо оголошення одномірного масиву:

```
int mas[5];
int *ptr;
```

Тоді вираз ptr=mas вказує на перший елемент масиву. Записи mas і &mas[0] рівносильні. Вираз ptr+1 вказує на mas[1], далі йдуть елементи: mas[2], mas[3], mas[4].

ptr	ptr+1	ptr+2	ptr+3	ptr+4
mas[0]	mas[1]	mas[2]	mas[3]	mas[4]

Розміщення двовимірного масиву в пам’яті

Двовимірні масиви розташовані в пам’яті так само, як і одновимірні масиви, займаючи послідовні комірки пам’яті.

```
int mas[4][2];

int *ptr;
```

ptr	ptr+1	ptr+2	ptr+3	ptr+4	ptr+5
mas[0][0]	mas[0][1]	mas[1][0]	mas[1][1]	mas[2][0]	mas[2][1]

Приклад 4. Ініціалізація масива через вказівник

```
const short int size = 10;
int a[size];
int *ptr;

ptr = a;
cout << ptr << endl;

cout << "array: " << endl;
for (int i = 0; i<size; ++i){
    *(ptr + i) = rand() % 10;
    cout << *(ptr + i) << "\t";
}
cout << endl;
for (int i = 0; i<size; ++i){
    cout << a[i] << "\t";
}
```

Динамічне виділення пам'яті

Всі об'єкти в C++ можуть розміщатися в пам'яті або статично – пам'ять виділяється під час компіляції, або динамічно – під час виконання програми, за допомогою виклику функцій із стандартної бібліотеки. Статичне розміщення більш ефективне, оскільки виділення пам'яті відбувається до виконання програми, однак воно не є гнучким, оскільки ми повинні заздалегідь знати тип і розмір об'єкту.

Основні відмінності між статичними і динамічними об'єктами такі:

- статичні об'єкти визначаються іменованими змінними, і дії над цими змінними проводяться напряму, з використанням їх імен. Динамічні об'єкти не мають власних імен і дії над ними виконуються за допомогою вказівників.
- виділення і звільнення пам'яті під статичними об'єктами виконується компілятором автоматично. Виділення і звільнення пам'яті під динамічні об'єкти цілком залежить від програміста. Ця задача досить складна і під час цього виникають помилки.

Для динамічного виділення пам'яті існують оператори ***new*** і ***delete***.

Оператор new

int *ptr_int = new int(1024);

Тут оператор new виділяє пам'ять під безіменний об'єкт типу int і ініціалізує його значенням 1024 і повертає адресу створеного об'єкта. Ця адреса використовується для ініціалізації вказівника *ptr_int*. Всі дії над таким об'єктом виконуються шляхом розіменування даного вказівника, оскільки явно проводити операції з динамічним об'єктом не можна.

```
int *ptr_int = new int(1024);
cout << *ptr_int << endl;
*ptr_int = 10;
cout << *ptr_int << endl;
delete ptr_int;
```

Динамічні масиви

Динамічним називається масив, розмірність якого стає відомою в процесі виконання програми.

Приклад 5. Виділення пам’яті під динамічний масив.

Нехай розмірність динамічного масиву вводиться з клавіатури. Спочатку необхідно виділити пам’ять під цей масив, а потім створений динамічний масив треба видалити.

…

```
int n;
cin>>n;  // n – розмірність масиву
int *mas=new int[n]; // виділення пам’яті під масив
delete [] mas; // звільнення пам’яті
```

…

В цьому прикладі mas є вказівником на масив з n елементів. Оператор **int *mas=new int[n]** виконує дві дії: оголошується змінна типу вказівник, далі вказівнику надається адреса виділеної області пам’яті у відповідності з заданим типом об’єкта.

Якщо за допомогою операції **new** неможливо виділити потрібний об’єм пам’яті, то результатом операції **new** є 0.

Приклад 6. Вивести на екран масив із заданою кількістю елементів.

```
int *a;
int n;
cout << "Enter number of elements in your massive:" << endl;
cin >> n;
cout << "Your massive: " << endl;
a = new int[n];
for (int i = 0; i<n; i++){
    *(a + i) = i + 1;
    cout << *(a + i) << endl;
}

delete[]a;
```

Багатовимірні динамічні об’єкти

Іноді при програмуванні виникає необхідність створення багатовимірних динамічних об’єктів. Програмісти-початківці за аналогією з поданим способом створення одновимірних динамічних масивів для двовимірного динамічного масиву розмірності n*k запишуть наступне

```
mas=new int[n][k];  // Невірно! Помилка!
```

Такий спосіб виділення пам’яті не дасть вірного результату.

Приклад 5. Створення двовимірного масиву.

```
int n, m;
int** a; //a - вказівник на масив вказівників на рядки
cout << "input the n:";
cin >> n;
cout << "input the n:";
cin >> m;
a = new int*[n]; //виділення пам’яті для масиву вказівників на n рядків
for (int i = 0; i<n; i++)
     a[i] = new int[m]; //виділення пам’яті для кожного рядка масиву розмірністю nxm
for (int i = 0; i<n; i++){
     for (int j = 0; j < m; j++) {
          a[i][j]= rand() % 10;
          cout << a[i][j] << "   ";
     }
     cout << endl;
}

for (int i = 0; i<n; i++)
     delete[] a[i]; //звільнення пам’яті від кожного рядка
delete[] a; //звільнення пам’яті від масиву вказівників
```