ШАГ
комп'ютерна
АКАДЕМІЯ

ОСНОВИ ПРОГРАМУВАННЯ НА МОВІ C++

Урок № 12

Покажчики

Зміст

Додаткові матеріали уроку прикріплені до даного PDF-файлу. Для доступу до матеріалів, урок необхідно відкрити в програмі Adobe Acrobat Reader.

1. Статичне та динамічне виділення пам'яті

Статична пам'ять — це область зберігання всіх глобальних і статичних змінних. Змінні статичної пам'яті оголошуються лише один раз і знищуються після завершення програми.

Динамічна пам'ять, або пам'ять вільного зберігання відрізняється від статичної тим, що програма має явно запросити пам'ять для елементів, що зберігаються в цій області, а потім звільнити пам'ять, якщо вона більше не потрібна. При роботі з динамічною пам'яттю програма може, наприклад, дізнатися кількість елементів масиву на етапі виконання.

2. Покажчики

Покажчик — це змінна, яка містить адресу іншої змінної. Покажчики дуже широко використовуються в мові C++. За допомогою покажчиків ви можете звертатися до осередків оперативної пам'яті. Для цього вам потрібно знати адресу осередку.

Через те, що покажчик містить адресу об'єкта, це дає можливість «непрямого» доступу до цього об'єкта через покажчик. Припустимо, що x — змінна, наприклад, типу int, а px — покажчик, створений способом, який ми ще не розглядали. Унарна операція & видає адресу об'єкта, отже, оператор:

```
px = &x;
```

привласнює адресу x змінній px; кажуть, що px «вказує» на x. Операція & застосовна тільки до змінних і елементів масиву, конструкції виду: &(x-1) и &3 є незаконними.

Унарна операція * розглядає свій операнд як адресу кінцевої мети і звертається за цією адресою, щоб витягти вміст. Отже, якщо y теж має тип int, то:

```
y = *px;
```

привласнює y вміст того, на що вказує px. Так послідовність:

```
px = &x;
y = *px;
```

привласнює y те саме значення, що й оператор:

```
y = x;
```

Змінні, які беруть участь у коді вище необхідно оголосити:

```
int x, y;
int *px;
```

З описом для x і y ми вже неодноразово зустрічалися. Опис покажчика:

```
int* px;
```

є новим для нас і каже, що ми оголосили покажчик на осередок пам'яті цілого типу. Оголошення покажчика відрізняється від оголошення змінної наявністю * в оголошенні. Після оголошення покажчик ні на що не вказує. У ньому випадкове значення. Покажчик може зберігати тільки адресу осередку. Нагадаємо, що для присвоєння адреси в покажчик використовується & (отримання адреси), а для доступу до осередку пам'яті необхідно застосовувати * (розіменування).

Покажчики можуть входити у вираз. Наприклад, якщо px вказує на ціле x, то *px може з'являтися в будь-якому контексті, де може трапитися x. Наприклад:

```
y = *px + 1; // привласнює у значення, на 1 більше
             // ніж значення x;
cout<< *px;  // виводить поточне значення x;
d = sqrt((double) *px) // отримає в d квадратний
// корінь з x, причому до передачі функції sqrt
// значення x перетворюється на тип double
```

У виразах виду:

```
y = *px + 1;
```

унарні операції * і & пов'язані зі своїм операндом міцніше, ніж арифметичні операції, отже, такий вираз бере те значення, на яке вказує px, додає 1 і привласнює результат змінній y. Ми незабаром повернемося до того, що може означати вираз:

```
y = *(px + 1);
```

Ви можете використовувати розіменування й у лівій частині присвоювання. Якщо px вказує на x, то:

```
*px = 0;
```

вважає, що x дорівнює нулю (тому що використання * записує значення в той осередок, на який вказує px), а:

```
*px += 1;
```

збільшує його на одиницю, тому що це перетвориться на

```
*px = *px + 1;
```

І, нарешті, через те, що покажчики є змінними, то з ними можна поводитися як і з іншими змінними. Якщо py — інший покажчик на змінну типу int, то:

```
py = px;
```

копіює вміст px у py, унаслідок чого py вказує на те саме, що і px.

Розглянемо вивчені механізми на прикладі:

```
#include <iostream>
using namespace std;
int main()
```

```
{
    int x = 10;
    int y = 5;
    int* px;
    int* py;
    // Записуємо адресу змінної x у px
    px = &x;

    // Відображаємо адресу x через покажчик
    // та операцію взяття адреси
    cout << px << " " << &x << endl;

    // Відображаємо значення x через змінну
    // й операцію розіменування
    // на екрані 10 10
    cout << *px << " " << x << endl;

    // Змінюємо значення змінної x
    // використовуємо операцію розіменування
    *px = 99;

    /* Відображаємо нове значення x через змінну
       й операцію розіменування
       на екрані відобразиться 99 99
     */
    cout << *px << " " << x << endl<<endl;

    // Записуємо адресу змінної y в py
    py = &y;

    // Відображаємо адресу y через покажчик
    // та операцію взяття адреси
    cout << py << " " << &y << endl;

    // Відображаємо значення y через змінну
    // й операцію розіменування на екрані 5 5
    cout << *py << " " << y << endl;
```

```
    // записуємо значення адреси з px у py
    // тепер обидва покажчики вказують на x
    py = px;

    // Відображаємо значення x через змінну
    // і два покажчики на екрані 99 99 99
    cout << *px << " " <<*py<<" "<<x << endl << endl;

    return 0;
}
```

У цьому прикладі ми використовували різні можливості по роботі з покажчиками. Поекспериментуйте з кодом цього прикладу, щоб краще зрозуміти механіку покажчиків.

3. Покажчики та масиви

У мові C++ є сильний взаємозв'язок між покажчиками й масивами, настільки сильний, що покажчики й масиви дійсно треба розглядати одночасно. Будь-яку операцію, яку можна виконати за допомогою індексів масиву, можна зробити і за допомогою покажчиків. Покажемо, як це можна виконати. Опис:

```
int a[10];
```

визначає масив розміру 10, тобто набір з 10 послідовностей об'єктів, що називаються a[0], a[1], ..., a[9]. Запис a[i] відповідає елементу масиву через і позицій від початку. Якщо pa — покажчик на ціле, описаний як:

```
int *pa;
```

то присвоєння:

```
pa = &a[0];
```

призводить до того, що pa вказує на нульовий елемент масиву a. Це означає, що pa містить адресу елемента a [0].

Тепер присвоєння:

```
x = *pa;
```

копіюватиме вміст a[0] у x (змінна цілого типу).

Якщо pa вказує на деякий певний елемент масиву a, то вираз pa+1 вказує на наступний елемент. Вираз pa-1 вказує на попередній елемент. Вираз виду pa-i вказує на елемент, що стоїть на i позицій до елемента, чия адреса

знаходиться в pa. Вираз pa+i на елемент, що стоїть на i позицій після. Отже, якщо pa вказує на a[0], то:

```
*(pa+1)
```

посилається на вміст a[1], pa+i — адреса a[i], а *(pa+i) — вміст a[i].

Ці зауваження справедливі незалежно від типу змінних у масиві a. Суть визначення «додавання 1 до покажчика», а також його поширення на всю арифметику покажчиків, полягає в тому, що приріст масштабується розміром пам'яті, займаної об'єктом, на який вказує покажчик. У такий спосіб, i у pa+i перед додаванням множиться на розмір об'єктів, на які вказує pa.

Очевидно, є дуже тісна відповідність між індексацією й арифметикою покажчиків. Насправді, компілятор перетворює посилання на масив у покажчик на початок масиву. Унаслідок цього ім'я масиву є вказівним виразом. Звідси випливає декілька вельми корисних наслідків. Через те, що ім'я масиву є синонімом місця розташування його нульового елемента (фактично ім'я масиву — це покажчик на його нульовий елемент), то присвоєння:

```
pa = &a[0]
```

можна записати як pa = a.

Ще більш дивним, принаймні на перший погляд, здається той факт, що посилання на a[i] можна записати у вигляді *(a+i). Під час аналізу виразу a[i] у мові C++ він негайно перетвориться на *(a+i); ці дві форми абсолютно еквівалентні. Якщо застосувати операцію & до обох частин

такого співвідношення еквівалентності, то ми отримаємо, що &a[i] та a+i теж ідентичні: a+i — адреса i-го елемента від початку a. З іншого боку, якщо pa є покажчиком, то у виразах його можна використовувати з індексом pa[i] ідентично до *(pa+i). Підсумуємо: будь-який вираз, що включає масиви й індекси, може бути записаний через покажчики і зміщення і навпаки, причому навіть в одному і тому ж твердженні.

Є одна відмінність між ім'ям масиву і покажчиком, яке необхідно мати на увазі. Покажчик є змінною, отже, операції pa=a та pa++ мають сенс і компілюються. Однак, ім'я масиву є константою, а не змінною: конструкції типу a=pa або a++ або p=&a будуть незаконними.

Розглянемо декілька прикладів для закріплення матеріалу:

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

int main()
{
    const int size = 5;
    int arr[size] = { 33, 44, 7, 8, 9 };

    /* записуємо адресу нульового елемента масиву
       в покажчик */
    int* ptr = &arr[0];

    /*
       Показуємо значення нульового елемента масиву
       через розіменування покажчика.
       На екрані 33
    */
    cout << *ptr << endl;
```

```
    /*
       Показуємо значення першого елемента масиву
       через розіменування покажчика.
       Ми додаємо зсув за адресою на один елемент
       після чого виконуємо розіменування.
       Адреса всередині покажчика не змінюється.
       На екрані 44
    */
    cout << *(ptr + 1) << endl;

    /*
       Виконуємо зсув на один елемент цілого типу
       вперед і записуємо нову адресу в покажчик ptr.
       Фактично ця операція виглядає так
       ptr = ptr + 1 * sizeof(int)
       Тепер у покажчику міститься адреса першого
       елемента. Можна було також написати ptr++;
    */
    ptr = ptr + 1;

    /*
        Показуємо значення першого елемента масиву
        через розіменування покажчика.
        На екрані 44
    */
    cout << *ptr << endl;
    return 0;
}
```

У цьому прикладі ми розглянули ази використання покажчиків для доступу до елементів масиву: запис адреси масиву в покажчик, відображення значення, перехід на наступний елемент масиву.

Розглянемо ще один приклад. У ньому ми будемо використовувати покажчик для відображення елементів масиву і їхні зміни.

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

int main()
{
    const int size = 5;
    int arr[size] = { 33, 44, 7, 8, 9 };

    // записуємо адресу нульового елемента масиву
    // в покажчик
    int* ptr = arr;

    // відображаємо весь масив через покажчик
    // на екрані 33 44 7 8 9
    for (int i = 0; i < size; i++) {
        cout << *(ptr + i)<<" ";
    }

    // змінюємо значення першого елемента
    *(ptr + 1) = 55;

    // змінюємо значення другого елемента
    *(ptr + 2) = 12;
    cout << endl << endl;

    // відображаємо весь масив через покажчик
    // на екрані 33 55 12 8 9
    for (int i = 0; i < size; i++) {
        cout << *(ptr + i) << " ";
    }
    return 0;
}
```

А тепер модифікуємо цей приклад і застосуємо синтаксис покажчиків до масиву. Нагадаємо, ім'я масиву — це адреса нульового елемента. Однак, на відміну від значення адреси

в покажчику, її міняти не можна. Це константний покажчик. Про це ми поговоримо пізніше. Приклад звернення до імені масиву за допомогою синтаксису покажчиків:

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

int main()
{
    const int size = 5;
    int arr[size] = { 33, 44, 7, 8, 9 };
    // відображаємо весь масив на екрані 33 44 7 8 9
    for (int i = 0; i < size; i++) {
        cout << *(arr + i) << " ";
    }
    // змінюємо значення першого елемента
    *(arr + 1) = 55;
    // змінюємо значення другого елемента
    *(arr + 2) = 12;
    cout << endl << endl;
    // відображаємо весь масив на екрані 33 55 12 8 9
    for (int i = 0; i < size; i++) {
        cout << *(arr + i) << " ";
    }
    return 0;
}
```

А чи можемо ми використовувати синтаксис масивів до покажчика? Звісно! Розглянемо на прикладі:

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

int main()
{
    const int size = 5;
```

```
    int arr[size] = { 33, 44, 7, 8, 9 };
    int* ptr = arr;

    /* відображаємо весь масив через покажчик
       на екрані 33 44 7 8 9 */
    for (int i = 0; i < size; i++) {
        cout << ptr[i] << " ";
    }

    // змінюємо значення першого елемента
    ptr[1] = 55;

    // змінюємо значення другого елемента
    ptr[2] = 12;

    cout << endl << endl;
    /* відображаємо весь масив через покажчик
       на екрані 33 55 12 8 9 */
    for (int i = 0; i < size; i++) {
        cout << ptr[i] << " ";
    }
    return 0;
}
```

Ми вже знаємо, що ім'я масиву — це покажчик (адреса) на нульовий елемент. Що ж відбувається, коли ім'я масиву передається функції? Відповідь очевидна — їй передається розташування початку цього масиву. Усередині викликаної функції такий аргумент є такою самою змінною, як і будь-яка інша, тож ім'я масиву як аргумент дійсно є покажчиком, тобто змінною, що містить адресу.

Це означає, що ми можемо застосовувати арифметику покажчиків до імені масиву і всередині функції. Наприклад:

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

void ShowArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << *(arr + i) << " ";
    }
}
// обчислення суми елементів масиву
int GetAmount(int* ptr, int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
         sum += *(ptr + i);
    }
    return sum;
}

int main()
{
    const int size = 5;
    int arr[size] = { 33, 44, 7, 8, 9 };

    // відображаємо весь масив на екрані 33 44 7 8 9
    ShowArray(arr, size);
    cout << endl << endl;

    // показуємо суму елементів масиву
    // 101
    cout << "Amount of array elements: " <<
            GetAmount(arr, size)<<endl<<endl;

    /* відображаємо весь масив на екрані
       33 44 7 8 9
    */
    ShowArray(&arr[0], size);
    return 0;
}
```

Ми можемо навіть змінювати покажчик всередині функції. Наприклад, функцію GetAmount можна було б змінити так:

```
// обчислення суми елементів масиву
int GetAmount(int* ptr, int size) {
    int sum = 0;
    for (int i = 0; i < size; i++,ptr++) {
         sum += *ptr;
    }
    return sum;
}
```

Операція збільшення ptr абсолютно законна, оскільки ця змінна є покажчиком, ptr++ ніяк не впливає на оригінальну адресу масиву, а тільки збільшує локальну для функції GetAmount копію адреси.

Описи формальних параметрів у визначенні функції у вигляді:

```
int m[];
```

та

```
int *m;
```

абсолютно еквівалентні; якому виду опису треба віддати перевагу, визначається переважно тим, які вислови будуть використані при написанні функції. Якщо функції передається ім'я масиву, то залежно від того, що зручніше, можна вважати, що функція оперує або з масивом, або з покажчиком, і діяти далі відповідним чином. Можна навіть використовувати обидва види операцій, якщо це здається доречним і ясним.

4. Покажчики — аргументи функцій. Передача аргументів за покажчиком

Через те, що в C++ передача аргументів функцій здійснюється «за значенням» (передається копія змінної, а не сама змінна), викликана функція не має безпосередньої можливості змінити змінну у викликаючій програмі. Що ж робити, якщо вам дійсно треба змінити аргумент? Наприклад, програма сортування захотіла б поміняти два елементи, що порушують порядок, за допомогою функції з ім'ям swap. Для цього замало написати:

```
swap(a, b);
```

визначивши функцію swap при цьому в такий спосіб:

```
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Через виклик за значенням swap не може впливати на аргументи a та b у викликаючій функції.

На щастя, усе ж є можливість отримати бажаний ефект. Викликаюча програма передає покажчики значень, що підлягають зміні:

```
swap(&a, &b);
```

Через те, що операція & видає адресу змінної, то &a є покажчиком на a. У самій swap аргументи описуються як покажчики, і доступ до фактичних операндів здійснюється через них:

```
void swap(int* px,int* py)
{
    int tmp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```

Розглянемо створення функції Swap з передачею за покажчиком:

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

/* Передача змінних через покажчик
   Усередині міняємо значення змінних, на які
   вказують покажчики */
void Swap(int* a, int* b) {
    int c = *a;
    *a = *b;
    *b = c;
}

int main()
{
```

```
    int av = 5, bv = 9;
    Swap(&av, &bv);
    // 9 5
    cout << av << " " << bv << endl;
    return 0;
}
```

5. Аналіз використання NULL та nullptr

При створенні покажчика, якщо ви його не ініціалізуєте, у нього поміщається випадкове значення (його також називають сміттєвим значенням). Наявність сміттєвого значення в покажчику — це вкрай небезпечно. Адже це значення не має сенсу. Якщо ми спробуємо по ньому звернутися, наслідки будуть дуже смутні.

```
int*  ptr;
cout<<*ptr;
```

На щастя, Microsoft Visual Studio 2019 при спробі компіляції такої програми видасть помилку. Однак, будь-який інший компілятор C++ може повести себе інакше.

Тому краще при створенні покажчика форматувати його нейтральною адресою. Потрібно показати в коді програми, що покажчик порожній. Для цього використовують нульові покажчики. Нульовий покажчик показує, що він нікуди не вказує. Як показати, що покажчик нульовий? Є три шляхи.

Перший шлях — вписати значення 0 у покажчик:

```
int* ptr = 0;
```

Потім у коді ми можемо перевіряти, чи є покажчик нульовим на поточний момент часу:

```
if (ptr != 0){
…..
}
```

Такий шлях створення нульового покажчика можна використовувати, проте значення 0 — це числове значення, і воно не відображає, що ми маємо справу з адресою.

Другий шлях: ми можемо скористатися макросом NULL, який прийшов у C++ з мови C.

У разі C++ у Visual Studio замість NULL підставляється 0.

```
int* ptr = NULL;
……
if(ptr != NULL){
      ……
}
```

Читаючи цей код, стає зрозуміліше, що маєш справу з нульовим покажчиком. Однак у C++ є рідний спосіб створення нульового покажчика. Для цього потрібно використати константу nullptr.

```
int* ptr = nullptr;
……
if(ptr != nullptr){
      ……
}
```

Саме цей спосіб є найкращим у C++ для створення нульового покажчика. Розглянемо всі три підходи в загальному прикладі:

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

int main()
{
```

```
    // Покажчик не ініціалізовано
    // У ньому випадкове значення
    int* ptr;

    // зараз ptr — нульовий покажчик
    // так робити не рекомендується
    ptr = 0;
    cout << ptr<< endl;

    // Спадок, отриманий від мови C
    // так робити не рекомендується
    ptr = NULL;
    cout << ptr << endl;

    // сучасний спосіб C++
    // створення нульового покажчика
    ptr = nullptr;
    cout << ptr << endl;

    if (ptr == nullptr) {
       cout << "\n\nNull pointer was found!" << endl;
    }
    return 0;
}
```

Записуйте значення nullptr у покажчик, щоб не отримати неприємні моменти при пошуку багів у ваших програмах.

6. Домашнє завдання

1. Дано масив цілих чисел. Скориставшись покажчиками, поміняйте місцями елементи масиву з парними й непарними індексами (тобто ті елементи масиву, які стоять на парних місцях, поміняйте з елементами, які стоять на непарних місцях).
2. Дано два масиви, упорядковані за зростанням: A[n] та B[m]. Сформуйте масив C[n+m], що складається з елементів масивів A і B, упорядкований за зростанням. Використовуйте синтаксис покажчиків.
3. Дано два масиви: A[n] і B[m]. Необхідно створити третій масив, у якому потрібно зібрати:

- Елементи обох масивів;
- Загальні елементи двох масивів;
- Елементи масиву A, які не включаються до B;
- Елементи масиву B, які не включаються до A;
- Елементи масивів A і B, які не є загальними для них (тобто об'єднання результатів двох попередніх варіантів).

Обов'язково використовуйте синтаксис покажчиків для цього завдання.

Урок № 12
Покажчики