Лабораторна робота № 4

Динамічний розподіл пам’яті. Покажчики

Покажчики у C/C++

Основні поняття

Покажчик — змінна, значенням якої є адреса комірки пам`яті. Тобто покажчик посилається на блок даних з області пам`яті, причому на саме його початок. Покажчик може посилатися на змінну або функцію. Для цього потрібно знати адресу змінної або функції. Так ось, щоб дізнатися адресу конкретної змінної в C++ існує унарна операція взяття адреси &. Така операція витягує адресу оголошених змінних, для того, щоб його присвоїти покажчикові.

Покажчики використовуються для передачі за посиланням даних, що набагато прискорює процес обробки цих даних (в тому випадку, якщо обсяг даних великий), через те що їх не треба копіювати, як при передачі за значенням, тобто, використовуючи ім’я змінної. В основному покажчики використовуються для організації динамічного розподілу пам’яті, наприклад при оголошенні масиву, не треба буде його обмежувати в розмірі. Адже програміст заздалегідь не може знати, якого розміру потрібен масив тому чи іншому користувачеві, в такому випадку використовується динамічне виділення пам’яті під масив. Будь-який покажчик необхідно оголосити перед використанням, як і будь-яку змінну.

```
//оголошення покажчика
/*тип даних*/  * /*ім'я покажчика*/;
```

Принцип оголошення покажчиків такий само, як і принцип оголошення змінних. Відмінність полягає лише в тому, що перед ім’ям ставиться символ зірочки *. Візуально покажчики відрізняються від звичайних змінних тільки одним символом.

При оголошенні покажчиків компілятор виділяє декілька байт пам’яті, які відводяться в залежності від типу даних для зберігання деякої інформації в пам’яті.

Щоб отримати значення, записане в деякій області, на яке посилається покажчик потрібно скористатися операцією розіменування покажчика *. Необхідно поставити зірочку перед ім’ям і отримаємо доступ до значення покажчика. Розглянемо програму, яка буде використовувати покажчики.

```
#include <iostream>

using namespace std;

int main()
{
    int var = 123; // ініціалізація змінної var числом 123
    int *ptrvar = &var; // покажчик на змінну var (присвоїли адресу змінної покажчику)
    cout << "&var    = " << &var << endl; // адреса змінної var, що міститься в
пам'яті, отримана операцією взяття адреси
    cout << "ptrvar  = " << ptrvar << endl; // адреса змінної var, є значенням
покажчика ptrvar
    cout << "var     = " << var << endl; // значення у змінній var
    cout << "*ptrvar = " << *ptrvar << endl; // виведення значення, що міститься в
змінній var через покажчик, операцією розіменування покажчика

    return 0;
}
```

В програмуванні заведено додавати до імені покажчика префікс ptr, таким чином, отримаємо змістовне ім’я покажчика, та вже зі звичайною змінною такий покажчик не сплутати. Результат роботи програми:

```
&var    = 0x22ff08
ptrvar  = 0x22ff08
var     = 123
*ptrvar = 123
```

Покажчики можна порівнювати не тільки на рівність чи нерівність, адже адреси можуть бути менше або більше один відносно одного. Нижче наведено програму, яка буде порівнювати адреси покажчиків.

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

int main()
{
    int var1 = 123; // ініціалізація змінної var1 числом 123
    int var2 = 99; // ініціалізація змінної var2 числом 99
    int *ptrvar1 = &var1; // покажчик на змінну var1
    int *ptrvar2 = &var2; // покажчик на змінну var2
    cout << "var1    = " << var1 << endl;
    cout << "var2    = " << var2 << endl;
    cout << "ptrvar1 = " << ptrvar1 << endl;
    cout << "ptrvar2 = " << ptrvar2 << endl;
    if (ptrvar1 > ptrvar2) // порівнюємо значення покажчиків, тобто адреси змінних
        cout << "ptrvar1 > ptrvar2" << endl;
    if (*ptrvar1 > *ptrvar2) // порівнюємо значення змінних, на які посилаються
покажчики
        cout << "*ptrvar1 > *ptrvar2" << endl;

    return 0;
}
```

Результат роботи програми:

```
var1    = 123
var2    = 99
ptrvar1 = 0xffffcc2c
ptrvar2 = 0xffffcc28
ptrvar1 > ptrvar2
*ptrvar1 > *ptrvar2
```

У першому випадку, ми порівнювали адреси змінних, і, можна помітити, що адреса другої змінної завжди менше адреси першої змінної. При кожному запуску програми адреси можуть виділяютися різні. У другому випадку ми порівнювали значення цих змінних використовуючи операцію розіменування покажчика.

З арифметичних операцій, найчастіше використовуються операції додавання, віднімання, інкремент і декремент, бо за допомогою цих операцій, наприклад в масивах, обчислюється адреса наступного елементу.

Покажчики на покажчики

Покажчики можуть посилатися на інші покажчики. При цьому в комірках пам'яті, на які будуть посилатися перші покажчики, будуть міститися не значення, а адреси інших покажчиків. Число символів * при оголошенні покажчика показує порядок покажчика.

Щоб отримати доступ до значення, на яке посилається покажчик його необхідно

розіменовувати відповідну кількість разів. Розглянемо програму, яка буде виконує деякі операції з покажчиками порядки вище першого:

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

int main()
{
    int var = 123; // ініціалізація змінної var числом 123
    int *ptrvar = &var; // покажчик на змінну var
    int **ptr_ptrvar = &ptrvar; // покажчик на покажчик на змінну var
    int ***ptr_ptr_ptrvar = &ptr_ptrvar;
    cout << " var\t\t= " << var << endl;
    cout << " *ptrvar\t= " << *ptrvar << endl;
    cout << " **ptr_ptrvar   = " << **ptr_ptrvar << endl; // два рази розіменовуємо
покажчик, через те, що він другого порядку
    cout << " ***ptr_ptrvar  = " << ***ptr_ptr_ptrvar << endl; // покажчик третього
порядку
    cout << "\n ***ptr_ptr_ptrvar -> **ptr_ptrvar -> *ptrvar ->      var -> "<< var <<
endl;
    cout << "\t  " << &ptr_ptr_ptrvar<< " -> " << "    " << &ptr_ptrvar << " ->" <<
&ptrvar << " -> " << &var << " -> " << var << endl;

    return 0;
}
```

Результат роботи програми наведений нижче

```
 var        = 123
 *ptrvar    = 123
 **ptr_ptrvar   = 123
 ***ptr_ptrvar  = 123

 ***ptr_ptr_ptrvar -> **ptr_ptrvar -> *ptrvar ->      var -> 123
      0xffffcc20 ->     0xffffcc28 ->0xffffcc30 -> 0xffffcc3c -> 123
```

Дана програма доводить той факт, що для отримання значення кількість розіменування покажчика має збігатися з його порядком. Логіка n-кратного розіменування полягає в тому, що програма послідовно перебирає адреси всіх покажчиків аж до змінної, в якій міститься значення. У програмі показана реалізація покажчика третього порядку. І якщо, використовуючи такий покажчик (третього порядку) необхідно отримати значення, на яке він посилається, робиться 4 кроки:

1. За значенням покажчика третього порядку отримати адресу покажчика другого порядку;
2. За значенням покажчика другого порядку отримати адресу покажчика першого порядку;

3. За значенням покажчика першого порядку отримати адресу змінної;
4. За адресою змінної отримати доступ до її значення.

Покажчики на функції

Покажчики можуть посилатися на функції. Ім’я функції, як і ім’я масиву само по собі є покажчиком, тобто містить адресу входу

```
// оголошення покажчика на функцію
/* Тип даних */ (* /* ім'я покажчика */) (/* список аргументів функції */);
```

Тип даних визначаємо такий, який буде повертати функція, на яку буде посилатися покажчик. Символ покажчика і його ім’я беруться в круглі дужки, щоб показати, що це покажчик, а не функція, яка повертає покажчик на певний тип даних. Після імені покажчика йдуть круглі дужки, в цих дужках перераховуються всі аргументи через кому як в оголошенні прототипу функції. Аргументи успадковуються від тієї функції, на яку буде посилатися покажчик.

Розглянемо програму, яка використовує покажчик на функцію. Програма знаходить НСД — найбільший спільний дільник. НСД — це найбільше ціле число, на яке без залишку діляться два числа, введених користувачем. Вхідні числа також повинні бути цілими.

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

int gcd(int, int ); // прототип вказаної функції

int main()
{
    int (*ptrgcd)(int, int); // оголошення покажчика на функцію
    ptrgcd=gcd; // присвоюємо адресу функції покажчику ptrgcd
    int a, b;
    cout << "Enter first number: ";
    cin >> a;
    cout << "Enter second number: ";
    cin >> b;
    cout << "GCD = " << ptrgcd(a, b) << endl; // звертаємось до функції через покажчик

    return 0;
}

int gcd(int number1, int number2) // рекурсивна функція знаходження найбільшого
спільного дільника GCD
{
    if ( number2 == 0 ) // базове розв'язання
        return number1;
    return gcd(number2, number1 % number2); // рекурсивне розв'язання НОД
}
```

Результат роботи програми:

```
Enter first number: 1001
Enter second number: 65
GCD = 13
```

Динамічні масиви в C++

Динамічне виділення пам’яті необхідно для ефективного використання пам’яті комп’ютера. Наприклад, ми написали якусь програму, яка обробляє масив. Під час написання даної програми необхідно було оголосити масив, тобто задати йому фіксований розмір (наприклад, від 0 до 100 елементів). Тоді ця програма буде не універсальною, адже може обробляти масив розміром не більше 100 елементів. А якщо нам знадобляться всього 20 елементів, але в пам’яті виділиться місце під 100 елементів, адже оголошення масиву було статичним, а таке використання пам’яті вкрай неефективне.

В C+&#43 операції `new` і `delete` призначені для динамічного розподілу пам’яті комп’ютера. Операція `new` виділяє пам’ять з області вільної пам’яті, а операція `delete` звільняє виділену пам’ять. Пам’ять, яка виділяється, після її використання повинна вивільнятися, тому операції `new` і `delete` використовуються парами. Навіть якщо не вивільняти пам’ять явно, то

вона звільниться ресурсами ОС по завершенню роботи програми. Рекомендується все ж таки не забувати про операцію `delete`.

```
// приклад використання операції new
int * ptrvalue = new int;
// де ptrvalue - покажчик на виділену ділянку пам'яті типу int
// new - операція виділення вільної пам'яті під створюваний об'єкт.
```

Операція `new` створює об’єкт заданого типу, виділяє йому пам’ять і повертає покажчик правильного типу на дану ділянку пам’яті. Якщо пам’ять неможливо виділити, наприклад, в разі відсутності вільних ділянок, то повертається нульовий покажчик, тобто покажчик зі значенням 0 (nullptr). Виділення пам’яті можливо під будь-який тип даних: `int`, `float`, `double`, `char` і т.д.

```
// приклад використання операції delete:
delete ptrvalue;
// де ptrvalue - покажчик на виділену ділянку пам'яті типу int
// delete - операція вивільнення пам'яті
```

Розглянемо програму, в якій буде створюватися динамічна змінна

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

int main()
{
    int *ptrvalue = new int; // динамічне виділення пам'яті під об'єкт типу int
    *ptrvalue = 9; // ініціалізація об'єкта через покажчик
    //int *ptrvalue = new int (9); ініціалізація може виконуватися одразу при
оголошенні динамічного об'єкта
    cout << "ptrvalue = " << *ptrvalue << endl;
    delete ptrvalue; // вивільнення пам'яті

    return 0;
}
```

Створення динамічних масивів

Найчастіше операції `new` і `delete` застосовуються для створення динамічних масивів, а не для створення динамічних змінних. Розглянемо фрагмент коду створення одновимірного динамічного масиву.

```
// оголошення одновимірного динамічного масиву на 10 елементів:
float * ptrarray = new float [10];
// де ptrarray - покажчик на виділену ділянку пам'яті під масив дійсних чисел типу
float
// в квадратних дужках вказуємо розмір масиву
```

Після того як динамічний масив став непотрібним, потрібно звільнити ділянку пам’яті, яка під нього виділялась.

```
delete [] ptrarray;
```

Після оператора `delete` ставляться квадратні дужки, які говорять про те, що вивільняється ділянка пам’яті, що відводилась під одновимірний масив. Розглянемо програму, в якій створюється одновимірний динамічний масив, заповнений випадковими числами.

```
#include <iostream>
#include <ctime>
#include <iomanip>

using namespace std;

int main()
{
    srand(time(0)); // генерація випадкових чисел
    float *ptrarray = new float [10]; // створення динамічного масиву дійсних чисел на
десять елементів
    for (int count = 0; count < 10; count++)
        ptrarray[count] = (rand() % 10 + 1) / float((rand() % 10 + 1)); // заповнення
масиву випадковими числами з масштабуванням від 1 до 10
    cout << "array = ";
    for (int count = 0; count < 10; count++)
        cout << setprecision(2) << ptrarray[count] << "    ";
    delete [] ptrarray; // вивільнення пам'яті
    cout << endl;

    return 0;
}
```

Створений одновимірний динамічний масив заповнюється випадковими числами, отриманими за допомогою функцій генерації випадкових чисел, причому числа генеруються в інтервалі від 1 до 10, інтервал задається так - `rand()% 10 + 1`. Щоб отримати випадкові дійсні числа, виконується операція ділення, з використанням явного приведення до дійсного типу знаменника - `floatrand ()% 10 + 1`. Щоб показати тільки два знаки після коми використовуємо функцію `setprecision(2)`, прототип даної функції знаходиться в заголовку `<iomanip>`. Функція `time(0)` засіває генератор випадкових чисел тимчасовим значенням, таким чином, виходить, відтворювати випадковість виникнення чисел

```
array = 1.6    1    0.5    1    4    0.22    4.5    1.3    3    1.6
```

По завершенню роботи з масивом, він видаляється, таким чином, вивільняється пам’ять, відведена під його зберігання.

Тепер розглянемо фрагмент коду, в якому показано, як оголошується двовимірний динамічний масив.

```
// оголошення двовимірного динамічного масиву на 10 елементів:
float **ptrarray = new float* [2]; // два рядки в масиві
     for (int count = 0; count < 2; count ++)
         ptrarray [count] = new float[5]; // і п'ять стовпців
// де ptrarray - масив покажчиків на виділену ділянку пам'яті під масив дійсних чисел
типу float
```

Спочатку оголошується покажчик другого порядку `float **ptrarray`, який посилається на масив покажчиків `float* [2]`, де розмір масиву дорівнює двом. Після чого в циклі for кожному рядку масиву оголошеного в рядку 2 виділяється пам’ять під п’ять елементів. В результаті виходить двовимірний динамічний масив `ptrarray[2][5]`. Розглянемо приклад вивільнення пам’яті відводиться під двовимірний динамічний масив.

```
// вивільнення пам'яті яка відводиться під двовимірний динамічний масив:
     for (int count = 0; count < 2; count ++)
         delete [] ptrarray[count];
// де 2 - кількість рядків в масиві
```

Оголошення та видалення двовимірного динамічного масиву виконується за допомогою циклу, тому необхідно зрозуміти та запам’ятати те, як це робиться. Розглянемо програму, в якій створюється двовимірний динамічний масив.

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

int main ()
{
    srand(time (0)); // генерація випадкових чисел
    // динамічне створення двовимірного масиву дійсних чисел на десять елементів
    float **ptrarray = new float* [2]; // два рядки в масиві
    for (int count = 0; count < 2; count ++)
        ptrarray [count] = new float[5]; // і п'ять стовпців
    // заповнення масиву
    for (int count_row = 0; count_row < 2; count_row ++)
        for (int count_column = 0; count_column < 5; count_column ++)
            ptrarray [count_row] [count_column] = (rand ()% 10 + 1) / float ((rand ()%
10 + 1)); // заповнення масиву випадковими числами з масштабуванням від 1 до 10
    // вивід масиву
    for (int count_row = 0; count_row < 2; count_row ++)
    {
        for (int count_column = 0; count_column < 5; count_column ++)
            cout << setw(5) << setprecision(2) << ptrarray [count_row] [count_column]
<< "";
        cout << endl;
    }
    // видалення двовимірного динамічного масиву
    for (int count = 0; count < 2; count ++)
        delete [] ptrarray[count];

    return 0;
}
```

При виведенні масиву була використана функція `setw()`, вона відводить місце заданого розміру під виведені дані. У нашому випадку, під кожен елемент масиву по чотири позиції, це дозволяє вирівняти у стовпцях числа різної довжини

```
   6    1  1.1    1    6
0.62 0.25  1.3  1.1    1
```

Завдання

Завдання 5.1 (C++).

Створити структуру, що була визначена у попередній лабораторній роботі. Визначити функції, що дозволяють створювати змінну типу створеної структури та вводити дані з клавіатури. Реалізувати зберігання даних в оперативній пам’яті у вигляді динамічного

масиву та на диску — у файлі.

Програма повинна при старті перевірити наявність файлу з даними на диску, якщо файл є — прочитати його вміст у динамічний масив, якщо файл відсутній — створити його.

При завершенні роботи програма повинна зберегти всі дані, розміщені у динамічному масиві структур у файл.

Програма має містити функцію меню, що дозволяє користувачеві обирати потрібну йому дію: додавання запису, вилучення запису за вказаним id, виведення всіх даних, що зберігаються у масиві, в табличній формі, виведення даних, відповідно до запитів (параметри запитів вводити з клавіатури)

1. **Student:** id, Прізвище, Ім’я, По батькові, Дата народження, Адреса, Телефон, Факультет, Курс, Група.
   Запити:
   a. список студентів вказаного факультету;
   b. список студентів, що народились після вказаного року;
   c. список студентів, чиї номери телефонів починаються із вказаної послідовності цифр;
   d. список навчальної групи в алфавітному порядку.
2. **Customer:** id, Прізвище, Ім’я, По батькові, Адреса, Номер кредитної картки, Номер банківського рахунку.
   Запити:
   a. список покупців в алфавітному порядку;
   b. список покупців, у яких номер кредитної картки знаходиться в заданому інтервалі;
   c. список покупців, у яких адреса включає в себе вказану послідовність літер (наприклад, назву міста);
   d. список покупців, у яких номер банківського рахунку закінчується на вказану цифру.
3. **Patient:** id, Прізвище, Ім’я, По батькові, Адреса, Телефон, Номер медичної картки, Діагноз.
   Запити:
   a. список пацієнтів, що мають вказаний діагноз;
   b. список пацієнтів, чий номер медичної картки містить указану послідовність цифр;
   c. список пацієнтів, у яких адреса починається із вказаної послідовності символів;
   d. список пацієнтів, номер медичної карти яких знаходиться в заданому інтервалі.
4. **Abiturient:** id, Прізвище, Ім’я, По батькові, Адреса, Телефон, Оцінки.
   Запити:
   a. список абітурієнтів, що мають незадовільні оцінки;
   b. список абітурієнтів, у яких сума балів вище заданої;
   c. список абітурієнтів, у яких номер телефону починається із заданої послідовності цифр (інші символи номера ігноруються)
   d. вибрати вказану кількість n абітурієнтів, що мають найбільшу суму балів.

5. **Book:** id, Назва, Автор(и), Видавництво, Рік видання, Кількість сторінок, Ціна, Тип палітурки.
   Запити:
   a. список книг заданого автора;
   b. список книг, що видані вказаним видавництвом;
   c. список книг, кількість сторінок у яких належить указаному діапазону;
   d. список книг, що видані після заданого року.
6. **House:** id, Номер квартири, Площа, Поверх, Кількість кімнат, Вулиця, Тип будівлі, Термін експлуатації.
   Запити:
   a. список квартир, які мають задану кількість кімнат;
   b. список квартир, що мають вказану кількість кімнат і розташованих між вказаними поверхами;
   c. список квартир, якф експлуатуються не більше R (ввести з клавіатури) років, що знаходяться на вказаній вулиці;
   d. список квартир, які мають площу, що більше заданої.
7. **Phone:** id, Прізвище, Ім’я, По батькові, Адреса, Номер кредитної картки, Час міських розмов, Час міжнародних розмов.
   Запити:
   a. відомості про абонентів, у яких час міських розмов перевищує вказаний;
   b. відомості про абонентів, які користувались міжнародним зв’язком;
   c. відомості про абонентів, номер кредитної картки яких закінчується на вказану послідовність цифр;
   d. відомості про абонентів в алфавітному порядку.
8. **Car:** id, Марка, Модель, Рік випуску, Колір, Ціна, Реєстраційний номер.
   Запити:
   a. список автомобілів заданої марки;
   b. список автомобілів заданої моделі, які експлуатуються більше n років;
   c. список автомобілів вказаного кольору, реєстраційний номер яких містить указану послідовність цифр;
   d. список автомобілів вказаного року випуску, ціна яких більше вказаної.
9. **Product:** id, Найменування, Тип, Виробник, Ціна, Термін зберігання, Кількість.
   Запити:
   a. список товарів заданого найменування;
   b. список товарів заданого найменування, ціна яких не більше заданої;
   c. список товарів указаного типу заданого виробника;
   d. список товарів, термін зберігання яких більше заданого.
10. **Train:** id, Пункт призначення, Номер поїзда, Час відправлення, Число місць (загальних,

плацкарт, купе, люкс).
Запити:

a. список поїздів, які прямують до заданого пункту призначення;
b. список поїздів, які прямують до заданого пункту призначення та відправляються після вказаної години;
c. список поїздів, у яких кількість плацкартних місць більше ніж усіх інших разом;
d. список поїздів, які відправляються до заданого пункту призначення та мають загальні місця.

Завдання 5.2 (Kotlin).

Виконати завдання 5.1, застосувавши мову програмування Kotlin. Замість структур (C++) використовувати Data Classes Замість динамічних масивів використовувати списки