Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра ЕОМ

**ЛАБОРАТОРНА РОБОТА № 3.**

**ЗАСОБИ РОБОТИ З ДИНАМІЧНОЮ ПАМ’ЯТТЮ. ДИНАМІЧНІ МАСИВИ**

**Варіант 16**

Виконав: ст.гр. КІ-15

Леземезюк Р.Т.

Прийняв: асистент

Козак Н.Б.

Львів – 2020

***Мета:*** *познайомитися із динамічними масивами.*

***КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ***

*Динамічне виділення пам’яті*

*В С++ об‘єкти можна розміщати статично – під час компіляції, або динамічно – під час виконання програми, шляхом виклику функцій зі стандартної бібліотеки. Основна відмінність у використанні даних методів – в їхній ефективності та гнучкості. Статичне розміщення більш ефективне, так як виділення пам‘яті відбувається до виконання програми, проте воно менш гнучке, тому що необхідно наперед знати тип і розмір об‘єкту. Задачі, в яких необхідно зберігати та обробляти наперед не відому кількість елементів, зазвичай потребують динамічного виділення пам‘яті. Динамічне виділення пам‘яті у мові С++ здійснюється за допомогою оператора new. Оператор new здійснює пошук неперервної області пам‘яті в області пам‘яті, що зветься некерована куча. Некерована куча – це стрктура даних за допомогою якої реалізована пам‘ять, що може бути виділена динамічно в ході виконання програми, а також це область пам‘яті, зарезервована під цю структуру. З іншої сторони куча - це довгий відрізок адрес пам'яті, поділений на блоки різних розмірів, що йдуть підряд. Пам‘ять у кучі поділяється на заняту і вільну. Перед початком роботи програми вся пам‘ять у кучі позначається як вільна. При виклику оператора динамічного виділення пам‘яті у кучі відбувається пошук неперервного сегменту вільної пам‘яті заданого розміру. Час такого пошуку є значним і займає більшу частину часу, що необхідна для виконання операції динамічного виділення пам‘яті. Якщо такий сегмент було знайдено в кучі, то він помічається як занятий і програмі повертається адреса його початку, інакше – програмі повертається ознака відсутності такого сегменту в пам‘яті, найчастіше NULL. Якщо в ході виконання програми значення адреси початку цього сегменту втрачається, то заняту пам‘ять звільнити буде неможливо. Якщо ця ситуація проявляється неодноразово, то це може призвести до вичерпання вільної пам‘яті в системі. Коли динамічно виділена область пам‘яті стає непотрібною,*

*то її потрібно звільнити за допомогою оператора звільнення динамічно виділеної пам‘яті. При його виклику область пам‘яті на яку вказує вказівник і яка була попередньо динамічно виділена з кучі позначається як вільна і її можна буде в подальшому використовувати заново.*

*Оператор динамічного виділення пам‘яті new може мати дві форми:*

*1. Виділення пам’яті під одиничний об’єкт має наступний синтаксис:*

*тип \*вказівник = new тип;*

*Наприклад, при виконанні оператора*

*іnt \*іp = new іnt;*

*створюються 2 об'єкти: динамічний безіменний об'єкт розміром 4 байти (значення типу int займає 4 байти) і вказівник на нього з ім'ям іp розміром також 4 байти (у 32-ох бітній системі адреса займає 32 біти), значенням якого є адреса у пам‘яті динамічного об'єкта. Можна створити й інший вказівник на той же динамічний об'єкт:*

*іnt \*other = іp;*

*Якщо вказівникові іp присвоїти інше значення, то можна втратити доступ до динамічного об'єкта:*

*іnt \*іp = new іnt; іnt і = 0; іp = &і;*

*У результаті динамічний об'єкт як і раніше буде існувати, але звернутися до нього буде вже не можна. При виділенні пам'яті об'єкт можна ініціалізувати певним значенням (окрім масивів):*

*іnt \*іp = new іnt (3);*

*В даному випадку об‘єкт типу іnt буде ініціалізований значенням 3.*

*2. Виділення пам’яті під масив заданого розміру має наступний синтаксис:*

*тип \*вказівник = new тип\*розмір\_масиву+;*

*Наприклад, при виконанні оператора*

*double \*mas = new double [50];*

*виділяється пам‘ять під масив з 50 елементів типу double. Тепер з цією динамічно виділеною пам'яттю можна працювати як зі звичайним масивом:*

*\*(mas+5) = 3.27; mas[6] = mas[5] + sіn(mas[5]);*

*У випадку успішного завершення операція new повертає вказівник зі значенням, відмінним від нуля. Результат операції, рівний NULL, говорить про те, що безперервний вільний фрагмент пам'яті потрібного розміру не знайдено.*

*Оператор звільнення динамічної пам'яті delete звільняє для подальшого використання в програмі ділянку пам'яті, яка була раніше виділена оператором new. Синтаксис оператора delete має наступний вигляд:*

*delete вказівник; // видалення одиничного динамічного об‘єкту*

*delete[] вказівник; // видалення динамічного масиву*

*Операції new і delete дозволяють створювати і видаляти багатомірні динамічні масиви, підтримуючи при цьому ілюзію довільної розмірності.*

*Для створення динамічного двовимірного масиву використовуються наступні елементи:*

*1. вказівник на вказівник, який містить адресу початку допоміжного масиву адрес розмір якого рівний висоті двовимірного масиву (кількості рядків);*

*2. допоміжний масив адрес, що зберігає адреси одновимірних масивів, які власне міститимуть дані; розмір цих масивів рівний розміру ширини двовимірного масиву (кількості стовпців);*

*3. множина масивів, що зберігають дані (реалізують рядки масиву).*

*Якщо вимірів більше, то використовується більша кількість допоміжних масивів до яких приєднуватимуться інші масиви, завдяки чому власне і утворюватимуться нові виміри. Загалом можна сказати: скільки зірочок при оголошенні базового вказівника на багатовимірний масив, стільки вимірів міститиме цей масив.*

**Код програми**

#include <iostream>

#include <math.h>

using namespace std;

void func(int\*\* arr, int n)

{

int l = n / 5;

if (n>=5)

{

for (int i = l; i < n-l; i++)

{

for (int q = 0; q < l; q++)

{

arr[i][l + q] = 0;

arr[i][n - l - 1 - q] = 0;

}

}

for (int i = l; i < n - l; i++)

{

for (int q = 0; q < l; q++)

{

arr[l + q][i] = 0;

arr[n - l - 1 - q][i] = 0;

}

}

}

for (int i = 0; i < n; i++)

{

for (int q = 0; q < n; q++)

{

cout << arr[i][q] << " ";

}

cout << endl;

}

}

int main()

{

int n1;

cout << "Enter size of matrix: ";

cin >> n1;

int\*\* arr;

arr = new int\* [n1];

for (int i = 0; i < n1; i++)

{

arr[i] = new int[n1];

}

for (int i = 0; i < n1; i++)

{

for (int q = 0; q < n1; q++)

{

arr[i][q] = 1;

}

}

func(arr, n1);

delete[] arr;

system("pause");

return 0;

}

**Результат**

