Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



Звіт

З лабораторної роботи №3

З дисципліни «Об’єктно-орієнтоване програмування »

На тему:

ЗАСОБИ РОБОТИ З ДИНАМІЧНОЮ ПАМ’ЯТТЮ. ДИНАМІЧНІ МАСИВИ

Виконав: Куцевол В.С. KI-109

Прийняв: асис. каф. ЕОМ

Гузинець Н.В

Львів – 2024

**Тема роботи:** Засоби роботи з динамічною пам’яттю. Динамічні масиви

**Мета:** познайомитися із динамічними масивами.

**Теоретичні відомості**

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, 38 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.

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

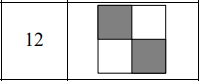
Якщо нам необхідно прочитати скажімо 3 комірку 2-го масиву даних (комірку 3-го стовпця у другому рядку за умови нумерації рядків і стовпців починаючи з нуля), то нам слід записати наступний рядок коду \*(\*(ptr+2)+3). Цей рядок коду аналогічний запису ptr[2][3]. При роботі з масивами форми з дужками і вказівниками є еквівалентними, тому можна застосовувати будь-яку з них, яка є зручнішою для конкретного випадку. Проте слід зазначити що форма з вказівниками є низькорівневішою і значно потужнішою, ніж дужкова форма, проте часто вона є джерелом помилок. Для роботи з багатовимірними масивами допускається і змішана форма: \*(ptr[2]+3).

Створюючи масиви з даними різного розміру у другому вимірі динамічних двовимірних масивів можна створювати так звані зубчаті масиви, які в окремих випадках можуть значно зекономити пам‘ять в порівнянні з використанням двовимірних масивів. Приклад зубчатого масиву зображено на рис. 3.3. Принцип створення зубчатих масивів відрізняється від принципу створення двовимірних динамічних масивів лише етапом створення одновимірних масивів з даними, де на відміну від двовимріних динамічних масивів масиви з даними можуть мати різний розмір утворюючи не матрицю, а зубці, завдяки чому даний вид масивів і одержав свою назву.

**Хід виконання роботи**

**Варіант 12**

Завдання : Задано квадратну матрицю, всі елементи якої рівні одиниці. Написати функцію void func (int\*\* arr, int n), котра заповняє заштриховану область матриці (згідно варіанту) нулями. Розмір масиву вводиться з клавіатури.



//file ArrayWork.h

#pragma once

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

int\*\* CreateMatrix(int numberOFElements) {

int\*\* matrix = new int\*[numberOFElements];

for (int i = 0; i < numberOFElements; i++) {

matrix[i] = new int[numberOFElements];

}

return matrix;

}

void FillMatrixWithOne(int\*\* matrix, int numberOfElements) {

for (int i = 0; i < numberOfElements; i++) {

for (int j = 0; j < numberOfElements; j++) {

matrix[i][j] = 1;

}

}

}

int CentreOfMatrix(int numberOfElements) {

if (numberOfElements % 2 == 0) {

return numberOfElements / 2;

}

if (numberOfElements % 2 != 0) {

return (numberOfElements / 2) + 1;

}

}

void func(int\*\* matrix, int numberOfElements , int centreOfMatrix) {

for (int i = 0; i < centreOfMatrix; i++) {

for (int j = 0; j < centreOfMatrix; j++) {

matrix[i][j] = 0;

}

}

for (int i = centreOfMatrix; i < numberOfElements; i++) {

for (int j = centreOfMatrix; j < numberOfElements; j++) {

matrix[i][j] = 0;

}

}

}

void PrintMatrix(int\*\* matrix, int numberOfElements) {

for (int i = 0; i < numberOfElements; i++) {

cout << endl;

for (int j = 0; j < numberOfElements; j++) {

cout << matrix[i][j] << "\t";

}

}

}

void DeleteMatrix(int\*\* matrix, int numberOfElements) {

for (int i = 0; i < numberOfElements; i++) {

free(matrix[i]);

}

free(matrix);

}

//file ProgramComp.h

#pragma once

#include <iostream>

#include"ArrayWork.h"

using std::cin;

using std::cout;

using std::endl;

void Program() {

int numberOfElements, centreOfMatrix;

int\*\* matrix;

cout << "Enter number of elements of matrix" << endl;

cin >> numberOfElements;

matrix = CreateMatrix(numberOfElements);

FillMatrixWithOne(matrix, numberOfElements);

centreOfMatrix = CentreOfMatrix(numberOfElements);

func(matrix, numberOfElements, centreOfMatrix);

PrintMatrix(matrix, numberOfElements);

DeleteMatrix(matrix, numberOfElements);

}

//file main.cpp

#include <iostream>

#include "ProgramComp.h"

using std::cin;

using std::cout;

using std::endl;

int main(void) {

Program();

return 0;

}

**Висновок**

У ході цієї лабораторної роботи я занурився у світ засобів роботи з динамічною пам'яттю на мові програмування C++. Особливу увагу я приділив динамічним масивам, оскільки вони відіграють важливу роль у створенні програм, які потребують гнучкості у використанні пам'яті.

Вивчення динамічних масивів дозволило мені краще зрозуміти, як ефективно керувати розмірами даних у програмі, коли передбачити статичний розмір масиву не можливо. Я вивчив процес виділення пам'яті за допомогою оператора new, а також вивільнення пам'яті з використанням оператора delete.

Ця лабораторна робота стала важливим кроком у моєму професійному розвитку, оскільки дозволила мені отримати поглиблені знання про роботу з динамічною пам'яттю в мові програмування C++. Я впевнений, що ці знання стануть корисними у подальшій роботі над проектами та вирішенні завдань, пов'язаних з програмуванням.