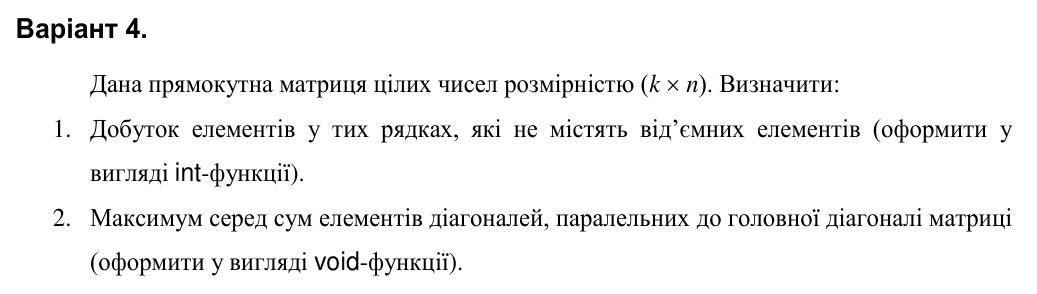
**ЗВІТ  
про виконання лабораторної роботи № 07.3.іт  
Опрацювання динамічних багатовимірних масивів  
з дисципліни  
"Алгоритмізація та програмування"  
студента групи РІ-12  
Грушевського Івана Олександровича**

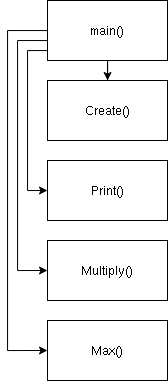
# Мета: Навчитися опрацьовувати динамічні двовимірні та багатовимірні масиви.

# Умова завдання:

Розмірності масивів слід задати за допомогою змінних а їх значення – вводити з  
клавіатури під час виконання програми (масиви мають бути динамічними). Усі необхідні  
дані мають передаватися функціям як параметри; всі величини, які використовуються лише  
всередині функцій, мають бути описані як локальні. Використання глобальних змінних у  
функціях не допускається. Виведення результатів роботи функцій має виконуватися в  
головній функції.  
Кожна функція має виконувати лише одну роль, і ця роль має бути відображена у  
назві функції.  
Рекурсивний та ітераційний способи – це 2 різні проекти, для яких потрібно 2 різних  
unit-тести і 2 різних звіти.  
93  
«Функція, яка повертає / обчислює / шукає ...» – має не виводити ці значення, а  
повернути їх у місце виклику як результат функції або як відповідний вихідний параметр.



# Структурна схема програми:



# Текст програми:

## main.cpp

// include header  
#include "../lib/function.h"  
  
// include libraries  
#include <iostream>  
#include <cstdlib>  
#include <ctime>  
  
using namespace std;  
  
int main() {  
 // script  
  
 srand((unsigned)time(NULL));  
  
 // variable declaration  
 int rowCount, colCount, Low, High;  
  
 cout << "Rows: "; cin >> rowCount;  
 cout << "Columbs: "; cin >> colCount;  
 cout << "Low: "; cin >> Low;  
 cout << "High: "; cin >> High;  
  
 if (rowCount <= 0 || colCount <= 0) {  
 cout << "Either rowCount or colCount is invalid" << endl;  
 return 0;  
 } else if (Low >= High) {  
 cout << "Low is not less than High" << endl;  
 return 0;  
 }   
  
 // creation of 2D array  
 int \*\*a = new int\*[rowCount];  
  
 for (int i=0; i<rowCount; i++)  
 a[i] = new int [colCount];  
  
 // main work  
 lib::Create(a, rowCount, colCount, Low, High);  
 lib::Print(a, rowCount, colCount);  
  
 cout << "Multiply: " << lib::Multiply(a, rowCount, colCount)  
 << endl;  
  
 lib::Max(a, rowCount, colCount);  
  
 // cleaning  
 for (int i=0; i<rowCount; i++)  
 delete [] a[i];  
  
 delete [] a;  
  
 return 0;  
}

## function.cpp

// include header  
#include "function.h"  
  
// include libraries  
#include <iostream>  
#include <iomanip>  
#include <cmath>  
  
using namespace std;  
  
namespace lib {  
 // functions script  
  
 void Create(int\*\* a, const int rowCount, const int colCount,  
 const int Low, const int High)  
 {  
 for (int i=0; i<rowCount; i++)  
 for (int j=0; j<colCount; j++)  
 a[i][j] = Low + rand() % (High-Low+1);  
 }  
  
 void Print(int\*\* a, const int rowCount, const int colCount)  
 {  
 cout << endl;  
 for (int i=0; i<rowCount; i++)  
 {  
 for (int j=0; j<colCount; j++)  
 cout << setw(4) << a[i][j];  
 cout << endl;  
 }  
 cout << endl;  
 }  
  
 /\*int Multiply(int\*\* a, const int rowCount, const int colCount)\*/  
 /\*{\*/  
 /\* int result = 1;\*/  
 /\* bool foundValidRow = false;\*/  
 /\*\*/  
 /\* for (int i = 0; i < rowCount; i++) {\*/  
 /\* int tmp = 1;\*/  
 /\* bool rowIsValid = true;\*/  
 /\*\*/  
 /\* for (int j = 0; j < colCount; j++) {\*/  
 /\* if (a[i][j] >= 0) {\*/  
 /\* tmp \*= a[i][j];\*/  
 /\* } else {\*/  
 /\* rowIsValid = false;\*/  
 /\* break;\*/  
 /\* }\*/  
 /\* }\*/  
 /\*\*/  
 /\* if (rowIsValid) {\*/  
 /\* result \*= tmp;\*/  
 /\* foundValidRow = true;\*/  
 /\* }\*/  
 /\* }\*/  
 /\*\*/  
 /\* if (!foundValidRow) {\*/  
 /\* return 0;\*/  
 /\* }\*/  
 /\*\*/  
 /\* return result;\*/  
 /\*}\*/  
 /\*\*/  
 /\*int Sum(int\*\* a, const int rowCount, const int colCount, int k)\*/  
 /\*{\*/  
 /\* int sum = 0;\*/  
 /\* int stopper = 0;\*/  
 /\*\*/  
 /\* if (rowCount < colCount)\*/  
 /\* stopper = rowCount;\*/  
 /\* else \*/  
 /\* stopper = colCount;\*/  
 /\*\*/  
 /\*\*/  
 /\* for (int i=0; i<stopper-1; i++)\*/  
 /\* {\*/  
 /\* cout << "| "<< sum << endl;\*/  
 /\* if (k > 0) {\*/  
 /\* cout << "$ " << a[i][i+static\_cast<int>(fabs(k))] << endl;\*/  
 /\* sum += a[i][i+static\_cast<int>(fabs(k))];\*/  
 /\* } else { \*/  
 /\* cout << "$ " << a[i+static\_cast<int>(fabs(k))][i] << endl;\*/  
 /\* sum += a[i+static\_cast<int>(fabs(k))][i];\*/  
 /\* }\*/  
 /\* }\*/  
 /\*\*/  
 /\* return sum;\*/  
 /\*}\*/  
 /\*\*/  
 /\*void Max(int\*\* a, const int rowCount, const int colCount)\*/  
 /\*{\*/  
 /\* int max = 0;\*/  
 /\* int k = 0;\*/  
 /\* if (rowCount%2 == 0)\*/  
 /\* k = rowCount / 2;\*/  
 /\* else \*/  
 /\* k = (rowCount-1) / 2;\*/  
 /\*\*/  
 /\* for (int k\_ind = -k; k\_ind < k; k\_ind++)\*/  
 /\* {\*/  
 /\* int r = Sum(a, rowCount, colCount, k\_ind);\*/  
 /\* if (r > max)\*/  
 /\* max = r;\*/  
 /\* }\*/  
 /\*\*/  
 /\* cout << "Max sum: " << max << endl;\*/  
 /\*}\*/  
  
  
 int Multiply(int\*\* a, const int rowCount, const int colCount) {  
 int result = 1;  
 bool foundValidRow = false;  
  
 for (int i = 0; i < rowCount; i++) {  
 int tmp = 1;  
 bool rowIsValid = true;  
  
 for (int j = 0; j < colCount; j++) {  
 if (a[i][j] >= 0) {  
 tmp \*= a[i][j]; // Multiply elements if they are non-negative  
 } else {  
 rowIsValid = false; // Mark row as invalid if there's a negative  
 break;  
 }  
 }  
  
 if (rowIsValid) {  
 result \*= tmp; // Multiply to the result if the row is valid  
 foundValidRow = true; // At least one valid row found  
 }  
 }  
  
 return foundValidRow ? result : 0; // Return 0 if no valid rows found  
 }  
  
   
 void Max(int\*\* a, const int rowCount, const int colCount) {  
 int maxSum = INT\_MIN; // Ініціалізуємо maxSum найменшим можливим значенням  
  
 // Перебираємо всі паралельні діагоналі  
 for (int k = -rowCount + 1; k < colCount; k++) {  
 int sum = 0;  
  
 // Обчислюємо суму для поточної діагоналі  
 for (int i = 0; i < rowCount; i++) {  
 int j = i + k; // Коригуємо індекс стовпця на основі зміщення  
  
 // Перевіряємо, чи знаходиться індекс стовпця в межах  
 if (j >= 0 && j < colCount) {  
 sum += a[i][j]; // Додаємо елемент до суми  
 }  
 }  
  
 // Оновлюємо maxSum, якщо поточна сума більша  
 if (sum > maxSum) {  
 maxSum = sum;  
 }  
 }  
  
 // Виводимо максимальну суму  
 cout << "Max sum of diagonals parallel to main diagonal: " << maxSum << endl;  
 }  
  
}

## Посилання на git-репозиторій з проктом

https://github.com/p-i-r-u-m/University-labs/tree/master/AP

# Результати unit-тесту:

## unit\_tests.cpp

// Include the gtest header  
# include <gtest/gtest.h>  
  
// Include lib headers  
#include "../lib/function.h"  
  
using namespace lib;  
  
// Helper function to allocate a 2D array  
int\*\* Create2DArray(int rowCount, int colCount) {  
 int\*\* array = new int\*[rowCount];  
 for (int i = 0; i < rowCount; ++i) {  
 array[i] = new int[colCount];  
 }  
 return array;  
}  
  
// Helper function to delete a 2D array  
void Delete2DArray(int\*\* array, int rowCount) {  
 for (int i = 0; i < rowCount; ++i) {  
 delete[] array[i];  
 }  
 delete[] array;  
}  
  
// Test cases  
TEST(MultiplyTest, ValidRows) {  
 // Test case where all rows have non-negative elements  
 int rowCount = 3;  
 int colCount = 3;  
 int\*\* matrix = Create2DArray(rowCount, colCount);  
   
 matrix[0][0] = 1; matrix[0][1] = 2; matrix[0][2] = 3; // Product: 6  
 matrix[1][0] = 4; matrix[1][1] = 5; matrix[1][2] = 6; // Product: 120  
 matrix[2][0] = 7; matrix[2][1] = 8; matrix[2][2] = 9; // Product: 504  
   
 EXPECT\_EQ(Multiply(matrix, rowCount, colCount), 6 \* 120 \* 504);  
   
 Delete2DArray(matrix, rowCount);  
}  
  
TEST(MultiplyTest, ContainsNegative) {  
 // Test case where one row contains a negative element  
 int rowCount = 3;  
 int colCount = 3;  
 int\*\* matrix = Create2DArray(rowCount, colCount);  
   
 matrix[0][0] = 1; matrix[0][1] = 2; matrix[0][2] = 3; // Product: 6  
 matrix[1][0] = 4; matrix[1][1] = -5; matrix[1][2] = 6; // Invalid row  
 matrix[2][0] = 7; matrix[2][1] = 8; matrix[2][2] = 9; // Product: 504  
   
 EXPECT\_EQ(Multiply(matrix, rowCount, colCount), 6 \* 504);  
   
 Delete2DArray(matrix, rowCount);  
}  
  
TEST(MultiplyTest, AllNegative) {  
 // Test case where all elements are negative  
 int rowCount = 3;  
 int colCount = 3;  
 int\*\* matrix = Create2DArray(rowCount, colCount);  
   
 matrix[0][0] = -1; matrix[0][1] = -2; matrix[0][2] = -3; // Invalid row  
 matrix[1][0] = -4; matrix[1][1] = -5; matrix[1][2] = -6; // Invalid row  
 matrix[2][0] = -7; matrix[2][1] = -8; matrix[2][2] = -9; // Invalid row  
   
 EXPECT\_EQ(Multiply(matrix, rowCount, colCount), 0);  
   
 Delete2DArray(matrix, rowCount);  
}  
  
TEST(MultiplyTest, MixedRows) {  
 // Test case with mixed valid and invalid rows  
 int rowCount = 4;  
 int colCount = 3;  
 int\*\* matrix = Create2DArray(rowCount, colCount);  
   
 matrix[0][0] = 1; matrix[0][1] = 2; matrix[0][2] = 3; // Product: 6  
 matrix[1][0] = 4; matrix[1][1] = -5; matrix[1][2] = 6; // Invalid row  
 matrix[2][0] = 7; matrix[2][1] = 8; matrix[2][2] = 9; // Product: 504  
 matrix[3][0] = 10; matrix[3][1] = 11; matrix[3][2] = 12; // Product: 1320  
   
 EXPECT\_EQ(Multiply(matrix, rowCount, colCount), 6 \* 504 \* 1320);  
   
 Delete2DArray(matrix, rowCount);  
}  
  
TEST(MultiplyTest, EmptyMatrix) {  
 // Test case with an empty matrix  
 int rowCount = 0;  
 int colCount = 0;  
 EXPECT\_EQ(Multiply(nullptr, rowCount, colCount), 0);  
}

## Вивід unit-тесту:

Internal ctest changing into directory: /home/pirum/University-labs/algorithmization\_and\_programming/7\_lab/3\_lab\_it/build  
Test project /home/pirum/University-labs/algorithmization\_and\_programming/7\_lab/3\_lab\_it/build  
 Start 1: MultiplyTest.ValidRows  
 Start 2: MultiplyTest.ContainsNegative  
 Start 3: MultiplyTest.AllNegative  
 Start 4: MultiplyTest.MixedRows  
 Start 5: MultiplyTest.EmptyMatrix  
1/5 Test #1: MultiplyTest.ValidRows ........... Passed 0.01 sec  
2/5 Test #2: MultiplyTest.ContainsNegative .... Passed 0.01 sec  
3/5 Test #3: MultiplyTest.AllNegative ......... Passed 0.01 sec  
4/5 Test #4: MultiplyTest.MixedRows ........... Passed 0.00 sec  
5/5 Test #5: MultiplyTest.EmptyMatrix ......... Passed 0.00 sec  
  
100% tests passed, 0 tests failed out of 5  
  
Total Test time (real) = 0.01 sec

# Висновки:

У результаті виконання лабораторної роботи я зміг Навчитися опрацьовувати динамічні двовимірні та багатовимірні масиви.