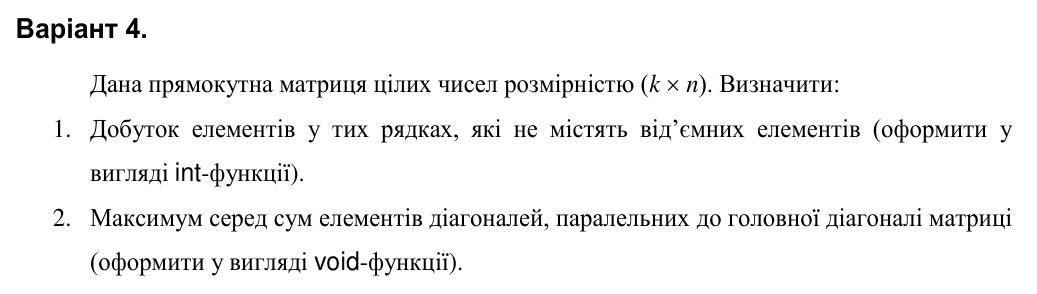
**ЗВІТ  
про виконання лабораторної роботи № 07.3.рек  
Опрацювання динамічних багатовимірних масивів  
з дисципліни  
"Алгоритмізація та програмування"  
студента групи РІ-12  
Грушевського Івана Олександровича**

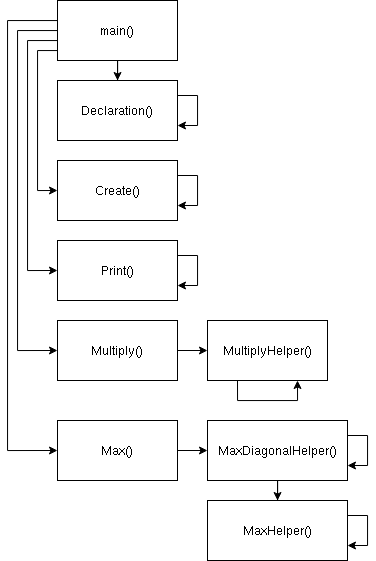
# Мета: Навчитися опрацьовувати динамічні двовимірні та багатовимірні масиви.

# 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() {  
 srand((unsigned)time(NULL));  
  
 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];  
  
 lib::Declaration(a, rowCount, colCount, 0);  
  
 lib::Create(a, rowCount, colCount, Low, High, 0, 0);  
 lib::Print(a, rowCount, colCount, 0, 0);  
  
 cout << "Multiply: " << lib::Multiply(a, rowCount, colCount) << endl;  
  
 lib::Max(a, rowCount, colCount);  
  
   
// cleaning  
 lib::Clean(a, rowCount, 0);  
  
 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 Declaration(int\*\* a, const int rowCount, const int colCount,  
 int i)  
 {  
 a[i] = new int [colCount];  
  
 if (i<rowCount-1)  
 Declaration(a, rowCount, colCount, i+1);  
 }  
  
 void Create(int\*\* a, const int rowCount, const int colCount,  
 const int Low, const int High, int i, int j)  
 {  
 a[i][j] = Low + rand() % (High-Low+1);  
  
 if (j<colCount-1) {  
 Create(a, rowCount, colCount, Low, High, i, j+1);  
 }  
 else {  
 if (i<rowCount-1) {  
 Create(a, rowCount, colCount, Low, High, i+1, 0);  
 }  
 }  
 }  
  
 void Print(int\*\* a, const int rowCount, const int colCount,   
 int i, int j)  
 {  
 cout << setw(4) << a[i][j];  
 if (j<colCount-1) {  
 Print(a, rowCount, colCount, i, j+1);  
 }  
 else {  
 if (i<rowCount-1) {  
 cout << endl;  
 Print(a, rowCount, colCount, i+1, 0);  
 }  
 else {  
 cout << endl << endl;  
 }  
 }  
 }  
  
   
   
// Recursive helper for Multiply function  
int MultiplyHelper(int\*\* a, const int rowCount, const int colCount, int i, int& result) {  
 if (i >= rowCount) {  
 return result;  
 }  
  
 int tmp = 1;  
 bool rowIsValid = true;  
  
 // Check the validity of the current row  
 for (int j = 0; j < colCount; j++) {  
 if (a[i][j] >= 0) {  
 tmp \*= a[i][j]; // Multiply non-negative elements  
 } else {  
 rowIsValid = false; // Mark row as invalid  
 break;  
 }  
 }  
  
 if (rowIsValid) {  
 result \*= tmp; // Multiply to the result if the row is valid  
 }  
  
 return MultiplyHelper(a, rowCount, colCount, i + 1, result); // Recur for the next row  
}  
  
// Wrapper function for Multiply  
int Multiply(int\*\* a, const int rowCount, const int colCount) {  
 int result = 1;  
 // Initialize result to 1 for multiplication  
 int foundValidRow = MultiplyHelper(a, rowCount, colCount, 0, result);  
   
 return (foundValidRow > 1) ? result : 0; // Return 0 if no valid rows found  
}  
  
  
void MaxHelper(int\*\* a, const int rowCount, const int colCount, int k, int i, int currentSum, int& maxSum) {  
 if (i >= rowCount) {  
 if (currentSum > maxSum) {  
 maxSum = currentSum;  
 }  
 return;  
 }  
  
 int j = i + k;   
 if (j >= 0 && j < colCount) {  
 currentSum += a[i][j];   
 }  
  
 MaxHelper(a, rowCount, colCount, k, i + 1, currentSum, maxSum);   
}  
  
void MaxDiagonalHelper(int\*\* a, const int rowCount, const int colCount, int k, int& maxSum) {  
 if (k > rowCount - 1) return;   
  
 int currentSum = 0;   
 MaxHelper(a, rowCount, colCount, k, 0, currentSum, maxSum);   
  
 MaxDiagonalHelper(a, rowCount, colCount, k + 1, maxSum);   
}  
  
void Max(int\*\* a, const int rowCount, const int colCount) {  
 int maxSum = 0;  
 MaxDiagonalHelper(a, rowCount, colCount, -rowCount + 1, maxSum);   
 cout << "Max sum: " << maxSum << endl;  
}  
  
  
 void Clean(int\*\* a, const int rowCount, int i)  
 {  
 delete [] a[i];  
 if (i<rowCount-2)  
 Clean(a, rowCount, i+1);  
 }  
}

## 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\_rec/build  
Test project /home/pirum/University-labs/algorithmization\_and\_programming/7\_lab/3\_lab\_rec/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 0.01 sec  
5/5 Test #5: MultiplyTest.EmptyMatrix ......... Passed 0.01 sec  
  
100% tests passed, 0 tests failed out of 5  
  
Total Test time (real) = 0.02 sec

# Висновки:

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