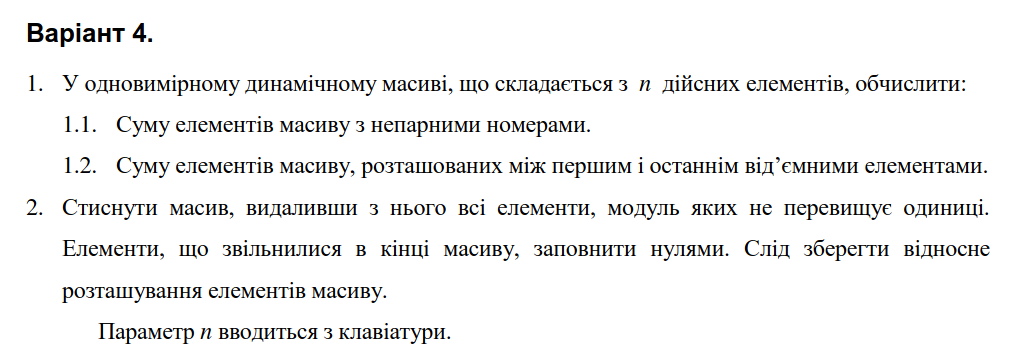
**ЗВІТ  
про виконання лабораторної роботи № 06.4.рек  
Опрацювання та впорядкування одновимірних динамічних масивів  
з дисципліни  
"Алгоритмізація та програмування"  
студента групи РІ-12  
Грушевського Івана Олександровича**

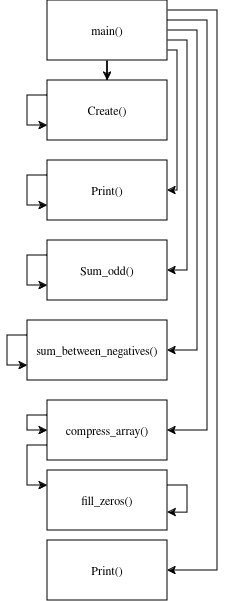
# Мета: навчитися опрацьовувати одновимірні динамічні масиви

# Умова завдання:

Необхідно написати програму для того, щоб виконати такі дії:  
- сформувати масив;  
- вивести його на екран у вигляді рядка, використовуючи форматне виведення;  
- виконати вказані у завдання дії;  
- вивести результати, причому модифікований масив вивести на екран у вигляді  
наступного рядка, використовуючи виведення з тими самими специфікаціями  
формату.  
Спосіб 1. Всі вказані дії необхідно реалізувати за допомогою окремих функцій, що  
використовують ітераційні алгоритми.  
Спосіб 2. Всі вказані дії необхідно реалізувати за допомогою окремих рекурсивних  
функцій.  
Інформацію у функції слід передавати лише за допомогою параметрів. Використання  
глобальних змінних – не допускається.  
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.  
«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 <time.h>  
  
using namespace std;  
  
int main() {  
 // script  
 srand((unsigned)time(NULL));  
  
 int n = 0;  
 cout << "n = "; cin >> n;  
  
 double \*a = new double[n];  
  
 double Low = -5.0;  
 double High = 5.0;  
  
 lib::Create(a, n, Low, High);  
 cout << "a = ["; lib::Print(a, n);  
  
 cout << "Sum of elements with odd indexes: " << lib::Sum\_odd(a, n) << endl;  
 int firstIndex = lib::first\_neg\_el(a, n);  
 int lastIndex = lib::last\_neg\_el(a, n);  
 cout << "Sum of elements between first and last negative: " << lib::sum\_between\_negatives(a, n, firstIndex, lastIndex, firstIndex + 1, 0) << endl;  
  
 lib::compress\_array(a, n, 0, 0);  
 cout << "a = ["; lib::Print(a, n);  
  
 delete[] a;  
 return 0;  
}

## function.cpp

// include header  
#include "function.h"  
  
// include libraries  
#include <iostream>  
  
using namespace std;  
  
namespace lib {  
 // functions script  
  
 void Create(double \*a, const int size, const double Low, const double High, int index) {  
 if (index < size) {  
 double random = Low + static\_cast<double>(rand()) / (static\_cast<double>(RAND\_MAX / (High - Low)));  
 a[index] = random;  
 Create(a, size, Low, High, index + 1);  
 }  
}  
  
 void Print(double \*a, const int size, int index) {  
 if (index < size) {  
 cout << " " << a[index];  
 Print(a, size, index + 1);  
 } else {  
 cout << " ]" << endl;  
 }  
}  
  
 double Sum\_odd(double \*a, const int size, int index) {  
 if (index >= size) {  
 return 0;  
 }  
 return (index % 2 != 0 ? a[index] : 0) + Sum\_odd(a, size, index + 1);  
}  
  
 int first\_neg\_el(double \*a, const int size, int index) {  
 if (index >= size) {  
 return -1;  
 }  
 if (a[index] < 0) {  
 return index;  
 }  
 return first\_neg\_el(a, size, index + 1);  
}  
  
 int last\_neg\_el(double \*a, const int size, int index) {  
 if (index < 0) {  
 index = size - 1;  
 }  
 if (index < 0) {  
 return -1;  
 }  
 if (a[index] < 0) {  
 return index;  
 }  
 return last\_neg\_el(a, size, index - 1);  
}  
  
 double sum\_between\_negatives(double \*a, const int size, int firstIndex, int lastIndex, int i, double sum) {  
 if (firstIndex == -1 || lastIndex == -1 || firstIndex >= lastIndex)   
 {  
 return 0.0;  
 }  
 else if (i < lastIndex) {  
 sum += a[i];  
 return sum\_between\_negatives(a, size, firstIndex, lastIndex, i+1, sum);  
 }  
 else {  
 return sum;  
 }  
 }  
  
 void fill\_zeros(double \*a, int k, int size) {  
 cout << "start: " << k << endl;  
 cout << "size: " << k << endl;  
 cout << "a[start]" << a[k] << endl;  
 if (k < size) {  
 a[k] = 0;  
 fill\_zeros(a, k + 1, size);  
 }  
 else   
 return;  
 }  
  
 void compress\_array(double \*a, int size, int i, int j) {  
 if (i >= size) {  
 fill\_zeros(a, j, size);  
 return;  
 }  
  
 if (abs(a[i]) > 1) {  
 a[j] = a[i];  
 compress\_array(a, size, i + 1, j + 1);  
 } else {  
 compress\_array(a, size, i + 1, j);  
 }  
 }  
}

## 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"  
  
// Test for the Sum\_odd function  
TEST(ArrayFunctionsTest, SumOdd) {  
 double arr1[] = {1.0, 2.0, 3.0, 4.0, 5.0};  
 EXPECT\_DOUBLE\_EQ(lib::Sum\_odd(arr1, 5), 6.0); // 2.0 + 4.0 = 6.0  
  
 double arr2[] = {0.5, -1.5, 2.5, -2.5, 1.5, 3.0};  
 EXPECT\_DOUBLE\_EQ(lib::Sum\_odd(arr2, 6), -1.0); // -1.5 + 3.0 = 1.5  
  
 double arr3[] = {0.0, 0.0, 0.0, 0.0};  
 EXPECT\_DOUBLE\_EQ(lib::Sum\_odd(arr3, 4), 0.0); // all zeros  
}

## Вивід unit-тесту:

Internal ctest changing into directory: /home/pirum/University-labs/algorithmization\_and\_programming/lab\_06/lab\_06\_4\_rec/build  
Test project /home/pirum/University-labs/algorithmization\_and\_programming/lab\_06/lab\_06\_4\_rec/build  
 Start 1: ArrayFunctionsTest.SumOdd  
1/1 Test #1: ArrayFunctionsTest.SumOdd ........ Passed 0.00 sec  
  
100% tests passed, 0 tests failed out of 1  
  
Total Test time (real) = 0.01 sec

# Висновки:

У результаті виконання лабораторної роботи я зміг навчитися опрацьовувати одновимірні динамічні масиви