ЗВІТ

про виконання лабораторної роботи № 6.01

«Обчислення сум та добутків за допомогою ітераційності» з дисципліни

«Алгоритмізація та програмування» студента групи РІ-12

Мілянця Мар`яна

# Мета роботи:

Навчитися програмувати пошук послідовним переглядом, обчислення кількості та суми заданих елементів одновимірного масиву.

# Умова завдання:

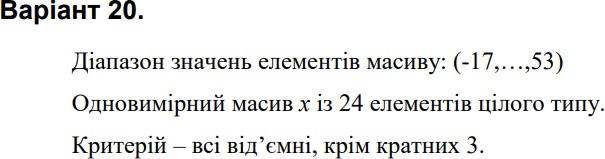
Написати програму, яка за допомогою генератора випадкових чисел формує вказаний масив – такий, що значення його елементів належать заданому діапазону. Обчислити кількість та суму тих елементів, які задовольняють вказаному критерію; а також замінити нулями ці елементи. Необхідно: - сформувати масив; - вивести його на екран у вигляді рядка, використовуючи форматне виведення; - виконати вказані у завдання дії (обчислити кількість та суму тих елементів, які задовольняють вказаному критерію; а також замінити нулями ці елементи); - вивести результат – обчислену кількість та суму, та вивести на екран модифікований масив у вигляді наступного рядка, використовуючи виведення з тими самими

специфікаціями формату. Всі вказані дії необхідно реалізувати за допомогою окремих функцій. Спосіб 1. Всі функції, у т.ч. ті, що реалізують обчислення кількості, суми, та заміну нулями вказаних

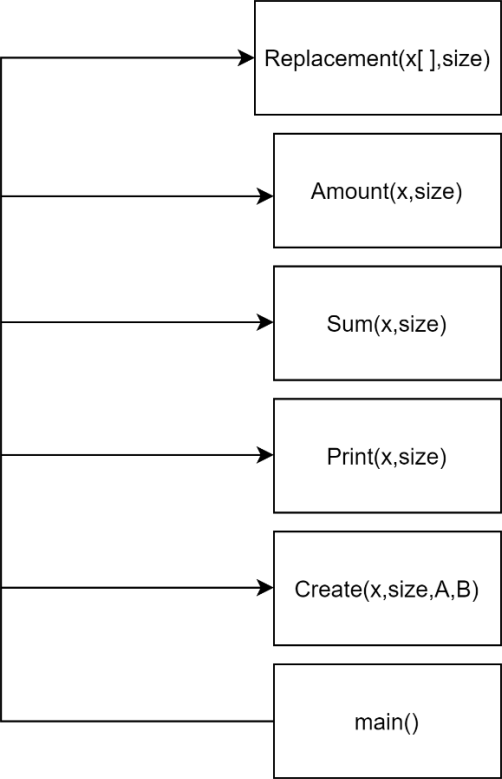
елементів, – використовують ітераційні алгоритми. Спосіб 2. Всі функції, у т.ч. ті, що реалізують обчислення кількості, суми, та заміну нулями

вказаних елементів, – мають бути рекурсивними. Інформація у функції має передаватися 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:



**Ітераційний спосіб:**

# Текст програми:

#include <iostream> #include <iomanip> #include <time.h>

using namespace std;

void Create(int\* x, const int size, const int A, const int B, int i)

{

x[i] = A + rand() % (B - A + 1);

if (i < size-1)

Create(x, size, A, B, i + 1);

}

void Print(int\* x, const int size, int i)

{

cout << setw(4) << x[i]; if (i < size - 1)

Print(x, size, i + 1); else

cout << endl;

}

int Sum(const int\* const x, const int size,const int i)

{

if (i < size)

{

if (x[i] < 0 && x[i] % 3 != 0)

return x[i] + Sum(x, size, i + 1); else

return Sum(x, size, i + 1);

}

else

return 0;

}

int Amount(const int\* const x, const int size, const int i)

{

if (i < size)

{

if (x[i] < 0 && x[i] % 3 != 0)

return Amount(x, size, i + 1) + 1; else

return Amount(x, size, i + 1);

}

else

return 0;

}

void Replacement(int x[], const int size, int i)

{

if (i < size)

{

if (x[i] < 0 && x[i] % 3 != 0) x[i] = 0;

Replacement(x, size, i + 1); return;

}

}

int main()

{

srand((unsigned)time(NULL)); const int n = 24;

int x[n]; int A = -17; int B = 53;

Create(x, n, A, B, 0);

Print(x, n, 0);

cout << "The sum of the elements that satisfy the condition = " << Sum(x, n, 0) << endl;

cout << "The number of elements that satisfy the condition = " << Amount(x, n, 0)

<< endl;

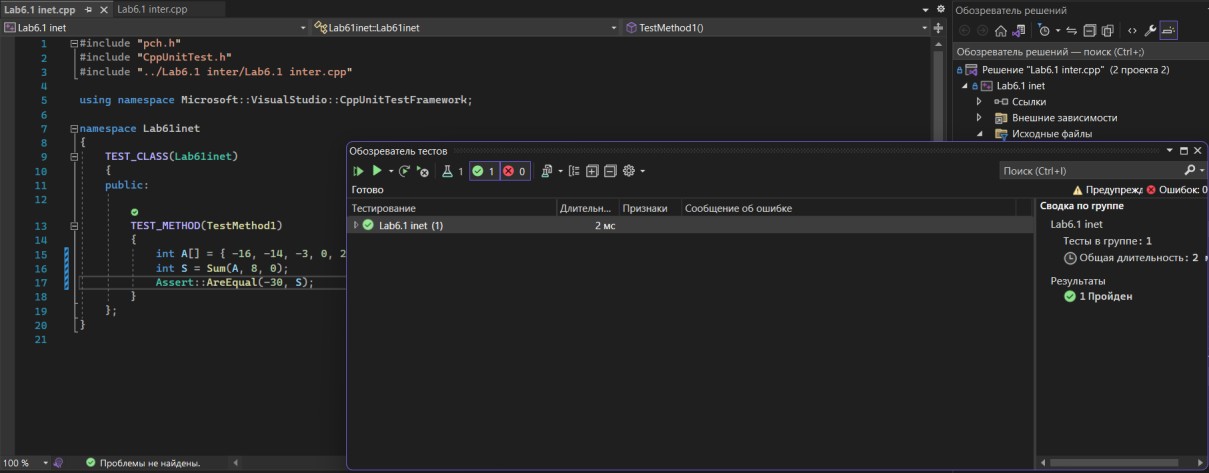
Replacement(x, n, 0);

Print(x, n, 0);

return 0;

}

# Результати unit-тесту:



**Текст unit-тесту:**

#include "pch.h" #include "CppUnitTest.h"

#include "../Lab6.1 inter/Lab6.1 inter.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework; namespace Lab61inet

{

TEST\_CLASS(Lab61inet)

{

public:

TEST\_METHOD(TestMethod1)

{

int A[] = { -16, -14, -3, 0, 2, 6, 10, 23 };

int S = Sum(A, 8, 0); Assert::AreEqual(-30, S);

}

};

}

**Посилання**: [https://github.com/milianets20/Lab-6.1-iter](https://github.com/milianets20/Lab-6.1-rec)

**Висновок**: виконуючи лабораторну роботу я навчився програмувати пошук послідовним переглядом, обчислення кількості та суми заданих елементів одновимірного масиву.