Додаток 1

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № ~~8~~6 з дисципліни «Алгоритми та структури даних-1. Основи алгоритмізації»

«Дослідження ~~лінійних~~ алгоритмів пошуку та сорування»

Варіант__30___

Виконав студент _____ ІП-15 Розін Олексій Іванович _____

(шифр, прізвище, ім'я, по батькові)

Перевірив _____Вечерковська Анастасія Сергіївна_____

(прізвище, ім'я, по батькові)

Київ 2021__

Лабораторна робота 8~~6~~

Дослідження ~~рекурсивних~~ алгоритмів пошуку та сортування

Мета – дослідити алгоритми пошуку та сортування, набути практичних навичок використання цих алгоритмів під час складання програмних специфікацій. ~~дослідити особливості роботи рекурсивних алгоритмів та набути практичних навичок їх використання під час складання програмних специфікацій підпрограм.~~

Індивідуальне завдання

Варіант 30

Постановка задачі

Розробити алгоритм та написати програму, яка складається з наступних дій:

1. Опису змінної індексованого типу (двовимірний масив) згідно з варіантом.
2. Ініціювання змінної, що описана в п.1 даного завдання.
3. Створення нової змінної індексованого типу (одновимірний масив) та її ініціювання значеннями, що обчислюються згідно з варіантом.

30	7 x 6	Дійсний	Із добутку від'ємних значень елементів рядків двовимірного масиву. Відсортувати методом Шела за спаданням.

~~Дано перший член і знаменник геометричної прогресії. Обчислити суму n перших членів прогресії та знайти n-й член прогресії.~~

Математична модель

Змінна	Тип	Ім'я	Призначення
~~Перший член прогресії~~Двомірний даний масив	Дійсний	arrA~~firstEl~~	Вхідні дані
~~Знаменник прогресії~~Масив добутків від'ємних елементів рядків даного масиву	Дійсний	~~denominator~~arrB	~~Вхідні дані~~Проміжні дані
Кількість рядків двомірного масиву та елментів одновимірного	Цілий	rows	Початкові дані
Кількість стовпців двомірного масиву	Цілий	columns	Початкові
~~Порядковий номер члена~~	Цілий	і~~n~~	~~Вхідні дані~~Проміжні дані

Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman
Formatted Table

~~прогресії~~Ліичильник для зовнішнього циклу			
~~Сума n членів прогресії~~Лічильник для внутрішнього циклу	~~Дійсний~~Цілий	j~~sum~~	Проміжні дані~~Результат~~
~~Значення n члена прогресії~~Тимчасова зміна для зміни місцями елементів масиву	Дійсний	count~~nVal~~	Проміжні дані~~Результат~~
Змінна шаг для сортування методом Шела~~Пошук суми n членів прогресії~~	Цілий~~Процедура~~	~~progressionSum~~d	Проміжні дані~~дані~~
Змінна для обчислення добутку від'ємних елементів рядків двовимірного масиву	Дійсний	mult	Проміжін дані
~~Пошук значення n члена прогресії~~Генерування двомірного даного масиву	Процедура	generateArrA()~~findProgressionElement~~	Проміжні дані
Генерування масива добутків	Процедура	generateArrB()	Проміжні дані
Сортування масива добутків	Процедура	sortArrB()	Проміжні дані Результат

Formatted: Font: (Default) Times New Roman

Formatted: English (United States)

Спочатку заповнюємо двомірний масив arrA[rows][columns] випадковими числами з діапазону від -10.5 до 10.5 за допомогою функції generateArrA(), яка має 3 параметра (перший – масив, елементи якого треба заповнити, другий та третій – кількість рядків та стовпців). Далі генеруємо одновимірний масив arrB[rows] за допомогою функції generateArrB(), яка має 4 параметра (перший – двомірний масив, другий – одновимірний масив, третій та четвертий – кількість рядків та стовпців двовимірного масиву). В тілі цієї функції за допомогою вкладеного арифметичного циклу(перший з лічильником і, другий - j), змінної mult = 1 та умовного оператору рахуємо добуток від'ємних елементів кожного рядка масиву arrA (mult *= arr[i][j]) та ініціалізуємо кожний елемент масиву arrB значенням mult. Далі викликаємо функцію sortArrB(), яка має 2 параметра (перший – масив, який треба відсортувати методом Шела, другий – вимірність масиву). Виводимо масив.~~Користувач задає значення firstEl, denominator та n. Перша рекурсивна функція progressionSum приймає 3 параметра: first, denom, n (first та denom дійсного типа, n - цілого). В тілі функції перевіряємо якщо n <= 0, то повертаємо 0 (return 0), інакше повертаємо first + progressionSum(first * denom, denom, n - 1). Друга рекурсивна функція findProgressionElement приймає такі ж параметри (first, denom, n). В тілі цієї функції перевіряємо, якщо n <= 1, то повертаємо перший аргумент first (return first), інакше повертаємо findProgressionElement(first, denom, n - 1) * denom. Ініціалізуємо змінну sum значенням, яке поверне визов функції progressionSum з параметрами (firstEl, denom, n). Ініціалізуємо змінну nVal значенням, яке поверне визов функції findProgressionElement з параметрами (firstEl, denom, n). Вивід sum та nVal.~~

Розв'язання

Програмні специфікації запишемо у псевдокоді та графічній формі у вигляді блок-схеми.

Крок 1. Визначимо основні дії.

Крок 2. ~~Введення firstEl, denominator, n~~Деталізуємо генерування масиву arrA[rows][columns].

Крок 3. ~~Ініціалізация sum~~Деталізуємо заповнення масиву arrB[rows].

Крок 4 ~~Ініціалізация nVal~~Деталізуємо сортування масиву arrB[rows] методом Шела.

Крок 5. Вивід arrB[rows]~~Вивід sum, nVal~~.

Псевдокод – основна программа

Крок 1

початок

Деталізуємо генерування масиву arrA[rows][columns]~~введення firstEl, denominator, n~~

Деталізуємо заповнення масиву arrB[rows]~~ініціалізація sum~~

Деталізуємо сортування масиву arrB[rows] методом Шела~~ініціалізация nVal~~

Вивід arrB[rows]~~вивід sum, nVal~~

кінець

Крок 2

початок

~~введення firstEl, denominator, n~~generateArrA(arrA, rows, columns)

Деталізуємо заповнення масиву arrB[rows]~~sum = progressionSum(firstEl, denominator, n)~~

Деталізуємо сортування масиву arrB[rows] методом Шела~~ініціалізация nVal~~

Вивід arrB[rows]~~вивід sum, nVal~~

кінець

Formatted: Font: (Default) Times New Roman

Formatted: Underline

Formatted: Underline

Formatted: No underline

Formatted: English (United States)

Formatted: Underline

Formatted: Underline

Formatted: Tab stops: 0.49", Left + 0.98", Left + 1.48", Left + 1.97", Left + 2.23", Left

Formatted: No underline

Крок 3

початок

generateArrA(arrA, rows, columns)

generateArrB(arrA, arrB, rows, columns)

Деталізуємо сортування масиву arrB[rows] методом Шела

Вивід arrB[rows]~~введення firstEl, denominator, n~~

~~sum = progressionSum(firstEl, denominator, n)~~

~~nVal = findProgressionElement(firstEl, denominator, n)~~

~~вивід sum, nVal~~

кінець

Крок 4

початок

generateArrA(arrA, rows, columns)

generateArrB(arrA, arrB, rows, columns)

sortArrB(arrB, rows)

Вивід arrB[rows]~~введення firstEl, denominator, n~~

~~sum = progressionSum(firstEl, denominator, n)~~

~~nVal = findProgressionElement(firstEl, denominator, n)~~

~~вивід sum, nVal~~

кінець

Псевдокод – підпрограми

generateArrA(arrray, rows, cols**)**

повторити для i **від** 0 **до** rows

повторити для j **від** 0 **до** cols

array[i][j] = rand() * 20.5 – 10.5

Formatted: Underline

Formatted: Underline

Formatted: No underline

Formatted: Underline

Formatted: Underline

все повторити

все повторити

кінець

generateArrB(arrayA, arrayB, rows, cols**)**

повторити для i **від** 0 **до** rows

mult = 1

повторити для j **від** 0 **до** cols

якщо array[i][j] < 0

mult *= array[i][j]

все якщо

все повторити

все повторити

кінець

sortArrB(array, n**)**

d = n / 2

поки d > 0

повторити

повторити для i **від** 0 **до** n – d

j = i

поки j >= 0 && array[j] < array[j + d]

повторити

count = array[j]

array[j] = array[j + d]

Formatted: Russian

Formatted: Font: Not Bold

Formatted: Font: Not Bold

Formatted: Font: Not Bold

array[j + d] = count

j -= 1

все повторити

все повторити

d /= 2

все повторити

кінець

~~**progressionSum**(first, denom, n)~~

~~**якщо** n <= 0~~

~~**то повернути** 0~~

~~**все якщо**~~

~~**повернути** first + **progressionSum**(first * denom, denom, n - 1)~~

~~**кінець**~~

~~**findProgressionElement**(firstEl, denominator, n)~~

~~**якщо** n <= 1~~

~~**то повернути** first~~

~~**все якщо**~~

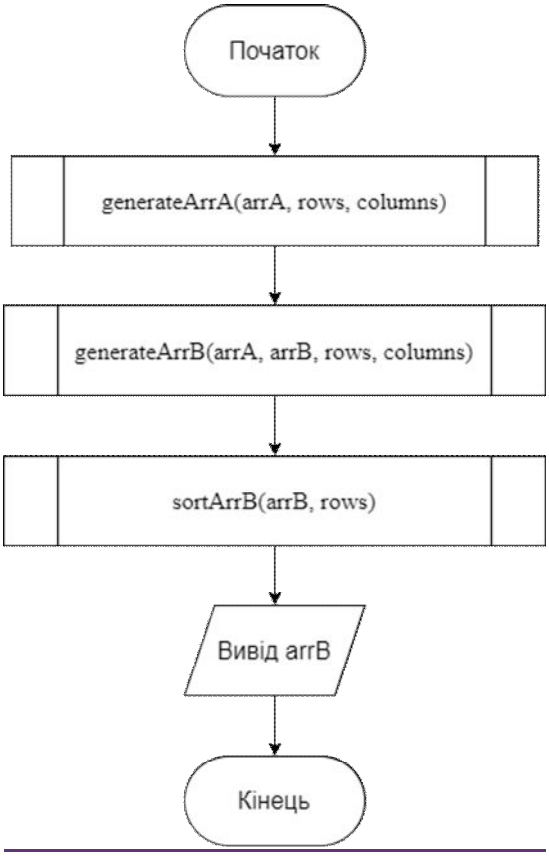
~~**повернути findProgressionElement**(first, denom, n - 1) * denom~~

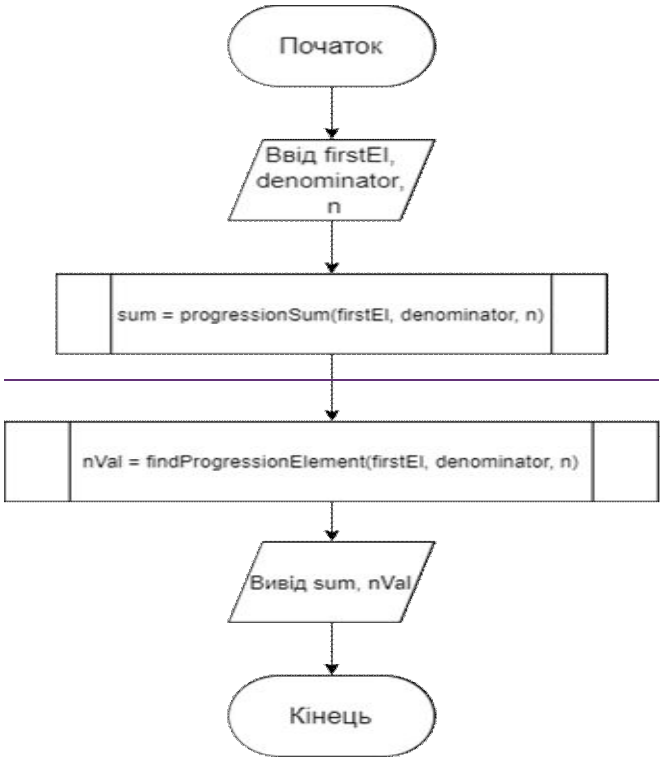
~~**кінець**~~

Блоксхема – основна програма

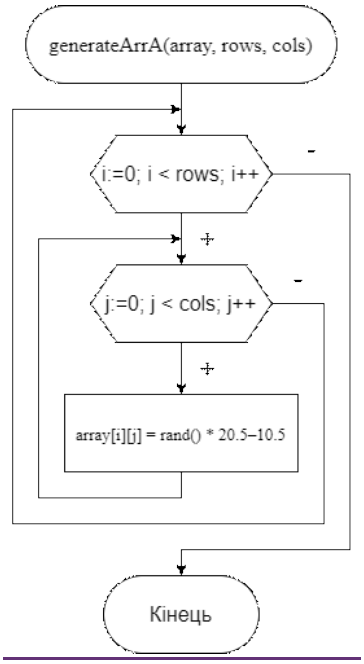
Formatted: Font: Bold

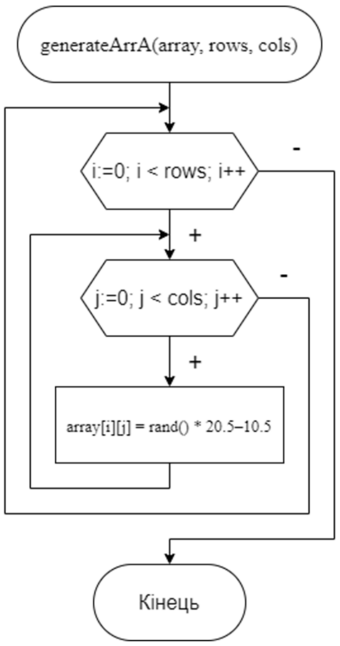
Formatted: Font: 12 pt, Not Bold

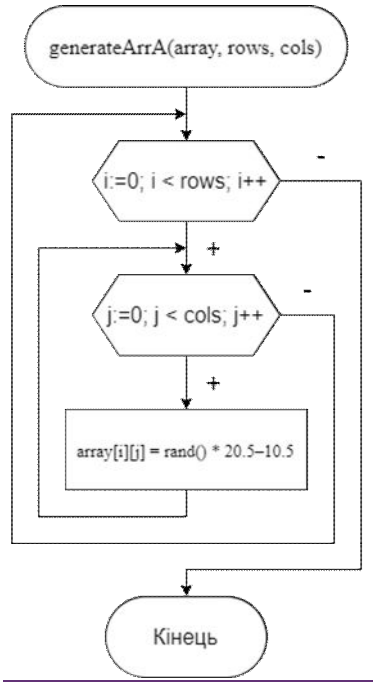

Початок
generateArrA(arrA, rows, columns)
generateArrB(arrA, arrB, rows, columns)
sortArrB(arrB, rows)
Вивід arrB
Кінець

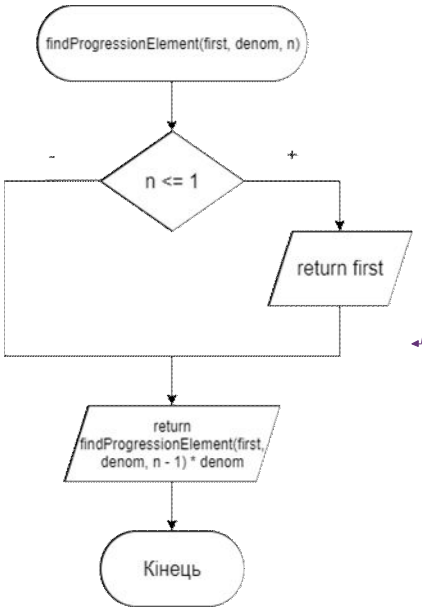


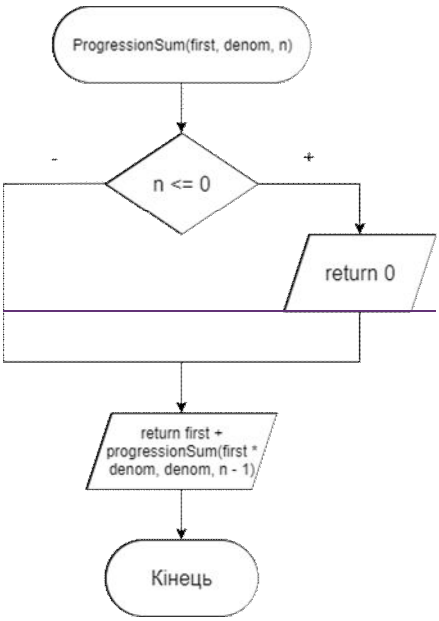
Блоксхема – підпрограми


generateArrA(array, rows, cols)
i:=0; i < rows; i++
-
+
j:=0; j < cols; j++
-
+
array[i][j] = rand() * 20.5–10.5
Кінець


generateArrA(array, rows, cols)
i:=0; i < rows; i++
-
+
j:=0; j < cols; j++
-
+
array[i][j] = rand() * 20.5–10.5
Кінець







Formatted: Centered

Formatted: Right

Код програми

lab6 (Глобальная область)

```cpp
#include <iostream>
using namespace std;

double progressionSum(double first, double denom, int n);
double findProgressionElement(double first, double denom, int n);

int main()
{
    setlocale(LC_ALL, "");
    double firstEl, denominator;
    int n;
    cout << "Введите первый член геометрической прогрессии: ";
    cin >> firstEl;
    cout << "Введите знаменатель прогрессии: ";
    cin >> denominator;
    cout << "Введите кол-во членов прогрессии: ";
    cin >> n;
    double sum = progressionSum(firstEl, denominator, n);
    double nVal = findProgressionElement(firstEl, denominator, n);
    cout << "Сумма " << n << " первых членов прогрессии: " << sum << endl;
    cout << n << "-ый член прогрессии: " << nVal << endl;
}

double progressionSum(double first, double denom, int n) {
    if (n <= 0) {
        return 0;
    }
    return first + progressionSum(first * denom, denom, n - 1);
}

double findProgressionElement(double first, double denom, int n) {
    if (n <= 1) {
        return first;
    }
    return findProgressionElement(first, denom, n - 1) * denom;
}
```

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

void generateArrA(double array[7][6], int rows, int cols);
void generateArrB(double arrayA[7][6], double arrayB[7], int rows, int cols);
void sortArrB(double array[7], int size);
void outputArrays(double arrayA[7][6], int rows, int cols, double arrayB[7]);


int main()
{
    const int rows = 7;
    const int columns = 6;
    double arrA[rows][columns];
    double arrB[rows];
    generateArrA(arrA, rows, columns);

    generateArrB(arrA, arrB, rows, columns);
    sortArrB(arrB, rows);

    outputArrays(arrA, rows, columns, arrB);
}

void generateArrA(double array[7][6], int rows, int cols) {
    srand(time(NULL));
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            array[i][j] = (double)(rand()) / RAND_MAX * 20.5 - 10.5;
        }
    }
}

void generateArrB(double arrayA[7][6], double arrayB[7], int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        double mult = 1;
        for (int j = 0; j < cols; j++) {
            if (arrayA[i][j] < 0) {
                mult *= arrayA[i][j];
            }
        }
        arrayB[i] = mult;
    }
}

void sortArrB(double array[7], int n) {
    double count;
    int d = n / 2;
    while (d > 0)
    {
        for (int i = 0; i < n - d; i++)
        {
            int j = i;
            while (j >= 0 && array[j] < array[j + d])
            {
                count = array[j];
                array[j] = array[j + d];
                array[j + d] = count;
                j--;
            }
        }
        d = d / 2;
    }
}

void outputArrays(double arrayA[7][6], int rows, int cols, double arrayB[7]) {
    cout << "   Random array A is:\n\n";
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << setprecision(6) << setw(13) << arrayA[i][j];
        }
        cout << "\n";
    }
    cout << "\n\n   Sorted array B is:\n\n";
    for (int i = 0; i < rows; i++) {
        cout << setprecision(6) << setw(12) << arrayB[i];
    }
    cout << "\n";
}
```

Консоль отладки Microsoft Visual Studio

```
Random array A is:

    7.50812     -2.22793     -3.62371     -7.05341      6.62098      3.59981
   -10.0614    -0.497436       5.0788     -6.23383      4.56015     -9.45082
   -2.60205     -8.13887    -0.690756      9.60836     -1.43025     -1.99207
   -7.71345     -1.77372      1.65848     -4.37822     -8.34658      5.43916
   -2.25108     -3.35218     -4.25935      5.51924      2.49057      5.47732
   -8.89839      8.86073      2.76897     -3.00121     -9.82057     -7.43817
   -6.73559     -6.01236     -3.54926      7.54566     -4.08542     -9.21871


Sorted array B is:

    1950.79      499.965      294.864     -32.1411     -41.6793     -56.9447     -5413.34
```

Консоль отладки Microsoft Visual Studio

```
Введите первый член геометрической прогрессии: 5
Введите знаменатель прогрессии: 3
Введите кол-во членов прогрессии: 3
Сумма 3 первых членов прогрессии: 65
3-ый член прогрессии: 45
```

~~Випробування~~

Formatted: Line spacing: 1.5 lines

~~Блок~~	~~Дія~~
	~~Початок~~
~~1~~	~~firstEl = 5; denominator = 3; n = 3~~
~~2~~	~~sum = progressionSum(firstEl, denominator, n)~~
~~2.1~~	~~Виклик progressionSum(firstEl, denominator, n)~~
~~2.2~~	~~n == 3~~ ~~n <= 0 == false~~ ~~return first + progressionSum(first * denom, denom, n - 1)~~
~~2.3~~	~~Виклик progressionSum(firstEl, denominator, n-1)~~
~~2.4~~	~~n == 2~~ ~~n <= 0 == false~~ ~~return first + progressionSum(first * denom, denom, n - 1)~~
~~2.5~~	~~Виклик progressionSum(firstEl, denominator, n-1)~~
~~2.6~~	~~n == 1~~ ~~n <= 0 == false~~ ~~return first + progressionSum(first * denom, denom, n - 1)~~
~~2.7~~	~~Виклик progressionSum(firstEl, denominator, n-1)~~
~~2.8~~	~~n == 0~~ ~~n <= 0 == true~~ ~~return 0~~

3	nVal = findProgressionElement(firstEl, denominator, n)
3.1	Виклик findProgressionElement(firstEl, denominator, n)
3.2	n == 3 n <= 1 == false return findProgressionElement(first, denom, n - 1) * denom
3.3	Виклик findProgressionElement(firstEl, denominator, n-1)
3.4	n == 2 n <= 1 == false return findProgressionElement(first, denom, n - 1) * denom
3.5	Виклик findProgressionElement(firstEl, denominator, n-1)
3.6	n == 1 n <= 1 == true return first
4	sum = 65 nVal = 45
5	Вивід sum, nVal
	Кінець

Висновки

Ми дослідили алгоритми пошуку та сортування, набули практичних навичок використання цих алгоритмів під час складання програмних специфікацій. Ми дослідили особливості роботи рекурсивних алгоритмів та набули практичних навичок їх використання під час складання програмних специфікацій підпрограм. У цій лабораторній роботі за допомогою двух рекурсивних функцій ми обчислили суму перших n членів та значення n члена даної нам геометричної прогресії. Склали програму, яка із даного нам двовимірного масиву дійсних чисел генерує новий масив, елементи якого дорівнюють добутку від'ємних елементів кожного рядка двовимірного масиву та сортирує цей масив за спаданням методом Шела.