МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп'ютерні інформаційні технології»

**Лабораторна робота №10**

**з дисципліни «Основи програмування»**

**на тему: «**Рекурсія та ітерація**»**

Виконав:

студент гр.ПЗ1911

Сафонов Д.Є.

Прийняла:

Нежуміра О. І.

Дніпро, 2020

**Тема.** Рекурсія та ітерація.

**Мета.** Навчитися описувати повторюванні обчислювальні процеси у рекурсивної та ітераційної формах.3) постановка завдання (завдання до лабораторної роботи, індивідуальне завдання);

# **Постановка завдання (завдання до лабораторної роботи, індивідуальне завдання).**

Розробити програму обчислення значення функції для заданого індивідуального завдання і обраного рівня складності.

Вимоги до програми:

− дані для обробки вводяться з консолі;

− валідація вхідних даних;

− результати роботи програми виводяться на консоль;

− обчислення функції рекурсивним методом;

− обчислення функції ітераційним методом;

− обчислення функції за допомогою функцій математичної бібліотеки, якщо це можливо.

Вимоги до тексту програми:

− специфікації програми і її функцій;

− самодокументованість коду: всі ідентифікатори повинні мати назви, що відповідають суті змінних.

Рівень III (90-100 балів): за допомогою звичайної і хвостової рекурсивних функцій та функції на основі ітерації для заданої точності обчислення (абсолютна різниця між двома сусідніми членами числового ряду не повинна перевищувати задане число).

17.

# **2. Зовнішні специфікації програми (вхідні та вихідні дані, функціональні вимоги до програми).**

2.1. Вхідні дані:

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування даних | Умовне позначення | Вимоги до данних | Приклад |
| X | x | Дійсне число більше -1, та меньше 1 | 0 |
| точність | epsilon | Дійсне число більше 0 | 0.01 |

2.2. Вихідні дані:

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування даних | Умовне позначення | Вимоги до данних | Приклад |
| Результат розрахунків | Назва функції(результат не зберігається в ОП) | Дійсне число більше 0, пораховане за формулою із потрібною точністю | 0.826 |

2.3. Функціональні вимоги:

- Перевірка вхідних даних

- Запобігання переповнення стеку

- Запобігання нескінченного циклу

# **3. Рекурсивний та ітераційний алгоритми розв’язання завдання.**

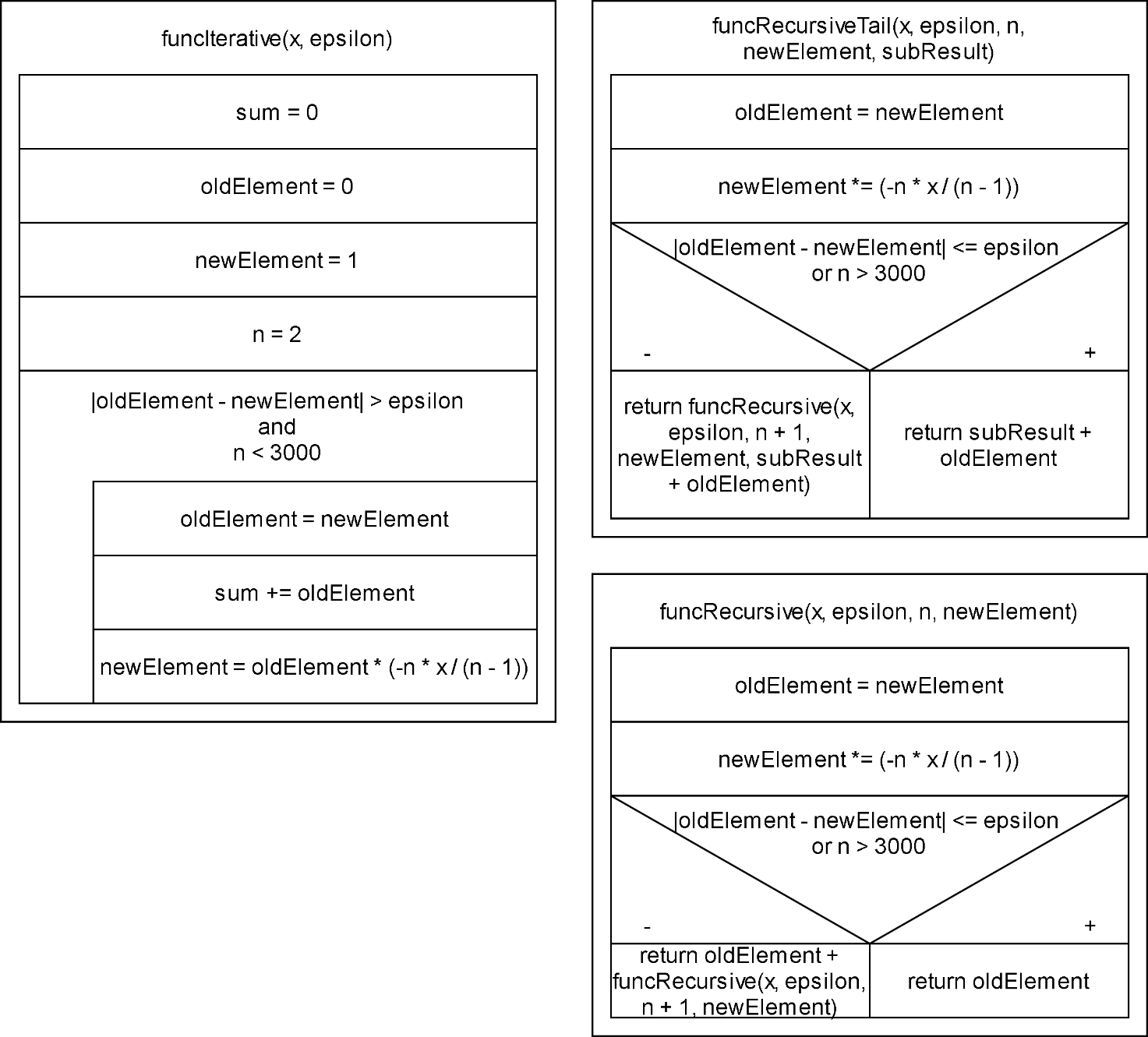


Рисунок 1

# **4. Схеми виконання рекурсивних функцій.**

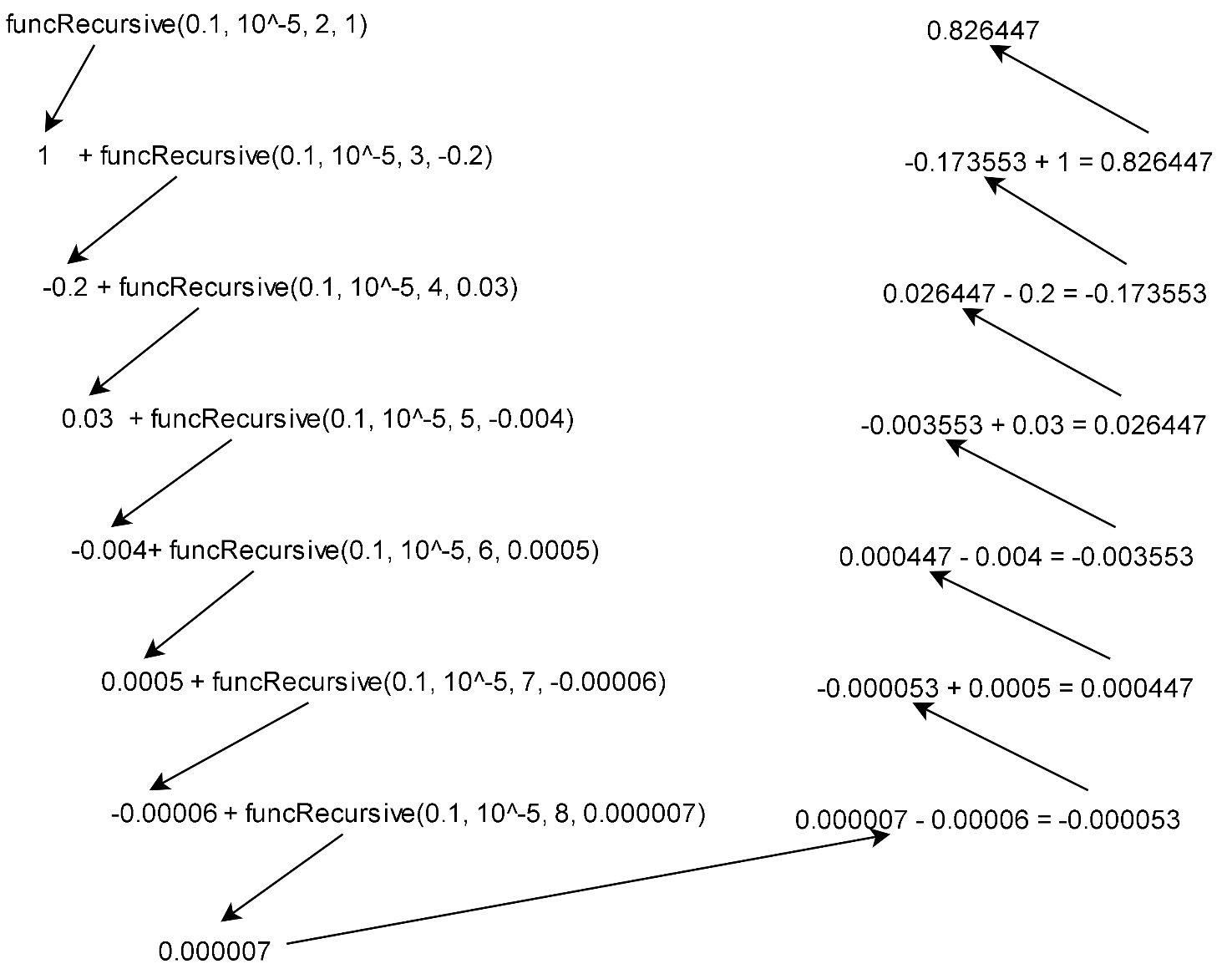


Рисунок 2(Хвостова рекурсія)

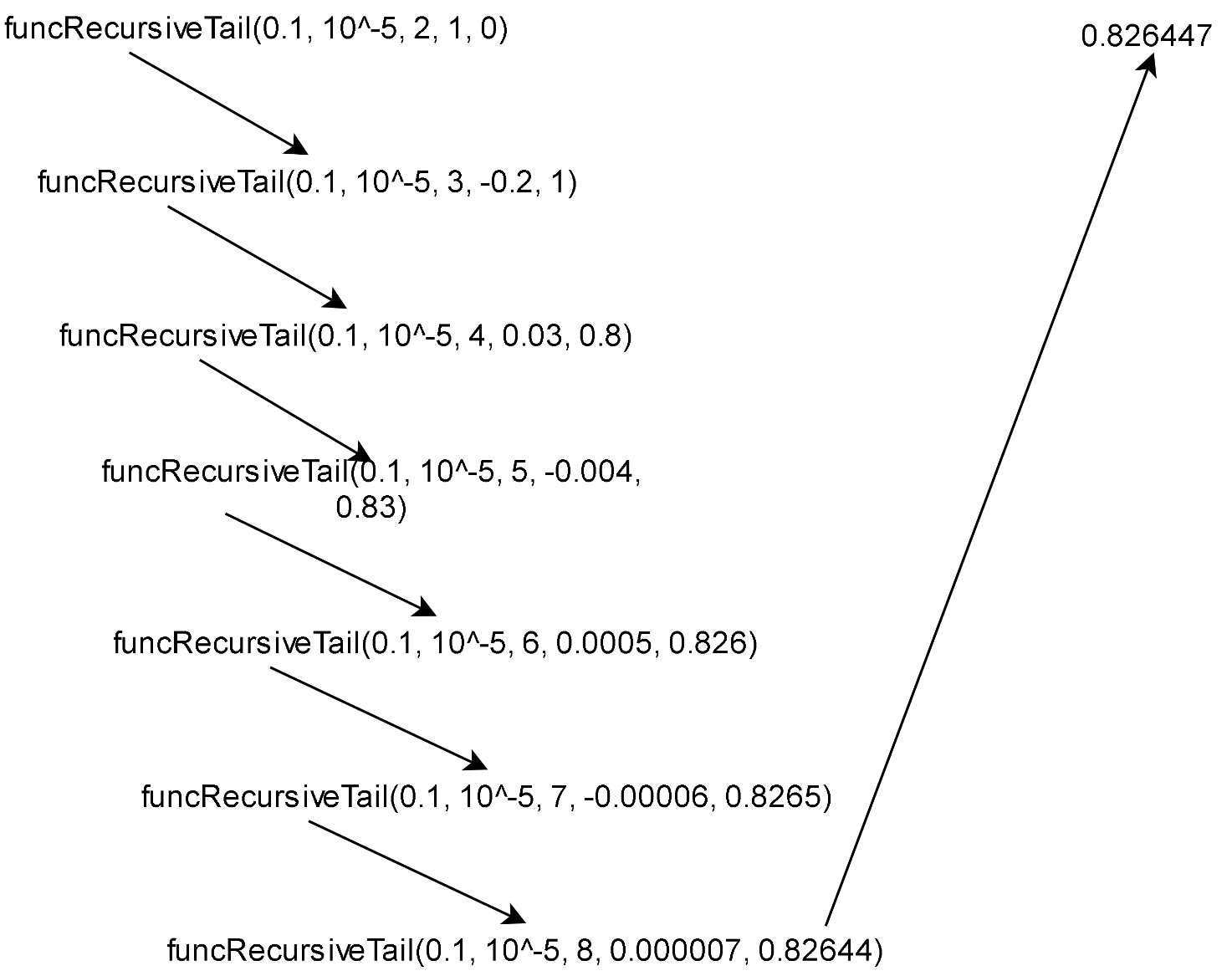


Рисунок 3(Звичайна рекурсія)

**5. Набори тестів для перевірки правильності виконання програми.**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва | Вхідні дані | Очікуваний результат |
| 1 | Граничні значення | x = 1 або -1 | Повідомлення про помилку |
| 2 | Некоректні значення x | |x|>1 | Повідомлення про помилку |
| 3 | Некоректні значення epsilon | epsilon <= 0, epsilon >=1 | Повідомлення про помилку |
| 4 | Коректні значення | |x|<1, усі значення у цьому проміжку з інтервалом 0.01  Та точністю 10^INT\_MIN | Коректний розрахунок |

# **6. Текст програми.**

“main.cpp”

#include <iostream>

#include <iomanip>

#include "computing.h"

#include "inp\_val7.h"

#include <cmath>

int main() {

std::cout << "this app calculates (x+1)^-2 for x in range (-1; 1) using cmath header, iterative function, recursive function and tail recursive function" << std::endl;

std::cout << std::fixed << std::setprecision(100);

bool exit;

do {

double x , epsilon;

x = cinum("input x in range (-1; 1): ", "wrong input, try again", 1, 0, std::nextafter(-1.0, 0.0), std::nextafter(1.0, 0.0));

epsilon = cinum("input epsilon in range (0; 1): ", "wrong input, try again", 1, 0, std::nextafter(0.0, 1.0), std::nextafter(1.0, 0.0));

std::cout << "1/(1+x)^2, x = \t" << x << std::endl;

std::cout << "epsilon = \t" << epsilon << std::endl << std::endl;

std::cout << "cmath \t\t" << funcNormal(x) << std::endl;

std::cout << "iterative \t" << funcIterative(x, epsilon) << std::endl;

std::cout << "recursive \t" << funcRecursive(x, epsilon) << std::endl;

std::cout << "recursive tail \t" << funcRecursiveTail(x, epsilon) << std::endl << std::endl;

char chExit;

std::cout << "Do you want to exit the programm?(Y/N): ";

std::cin >> chExit;

while ((chExit != 'Y' && chExit != 'y' && chExit != 'N' && chExit != 'n') || (std::cin.peek() != '\n')) {//until correct input

while (std::cin.get() != '\n');//clear istream

std::cout << "wrong input, try again." << std::endl;

std::cout << "Do you want to exit the programm?(Y/N): ";

std::cin >> chExit;

}

if (chExit == 'Y' || chExit == 'y')

exit = true;

else

exit = false;

std::cin.get();//enter

} while (!exit);

return 0;

}

“computing.h”

#ifndef \_\_COMPUTING\_H\_\_

#define \_\_COMPUTING\_H\_\_

// -1 < x < 1

double funcNormal(double x);

double funcIterative(double x, double epsilon);

double funcRecursive(double x, double epsilon, int n = 2, double newElement = 1);

double funcRecursiveTail(double x, double epsilon, int n = 2, double newElement = 1, double subResult = 0);

#endif

“computing.cpp”

#include "computing.h"

#include <cmath>

double funcNormal(double x) {

return (1 / ((1 + x) \* (1 + x)));

}

//n < 3000 to avoid stack overflow and/or infinite loop

double funcIterative(double x, double epsilon) {

double sum = 0, oldElement = 0, newElement = 1;

for (int n = 2; abs(oldElement - newElement) > epsilon && n < 3000; n++) {

oldElement = newElement;

sum += oldElement;

newElement = oldElement \* (-n \* x / (n - 1.0));

}

return sum;

}

double funcRecursive(double x, double epsilon, int n, double newElement) {

double oldElement = newElement;

newElement \*= (-n \* x / (n - 1.0));

if (abs(oldElement - newElement) <= epsilon || n > 3000)

return oldElement;

return oldElement + funcRecursive(x, epsilon, n + 1, newElement);

}

double funcRecursiveTail(double x, double epsilon, int n, double newElement, double subResult) {

double oldElement = newElement;

newElement \*= (-n \* x / (n - 1.0));

if (abs(oldElement - newElement) <= epsilon || n > 3000)

return subResult + oldElement;

return funcRecursiveTail(x, epsilon, n + 1, newElement, subResult + oldElement);

}

“inp\_val7.h”

//cinum - numerical only (R)

//char msg\_invite - your message for inviting user to input their number

//char msg\_error - if error input occures

//int multipleOf - check if input is multiple of this number

//bool intFlag - =1(only integer input); =0(any input)

//double left - leftmost point in diapasone for input

//double right - rightmost point in diapasone for input

double cinum(const char\* msg\_invite, const char\* msg\_error, int multipleOf, bool intFlag, double left, double right);

//input to char array with lenght validation

void val\_char\_lenght(char array[], int ary\_size);

“inp\_val7.cpp”

#include "inp\_val7.h"

#include <iostream>

#include <windows.h>

#include <cmath>

#include <climits>

#undef max

using namespace std;

double cinum(const char\* msg\_invite, const char\* msg\_error, int multipleOf = 1, bool intFlag = false, double left = -2147483648.0, double right = 2147483647.0) {

bool goodInput = false;

double doubleInput;

cout << msg\_invite << endl;

do {

cin >> doubleInput;

int integerInput;

integerInput = static\_cast<int>(doubleInput);

if (cin.fail() || (doubleInput <= left) || (doubleInput >= right) || ((intFlag == true) && (ceil(doubleInput) != floor(doubleInput))) || ((multipleOf != 1) && ((integerInput % multipleOf) != 0))) {

cout << msg\_error << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

else

goodInput = true;

} while (!goodInput);

return doubleInput;

}

void val\_char\_lenght(char array[], int ary\_size) {//input to char array with lenght validation

bool correct = false;

do {

std::cin.getline(array, ary\_size);

if ((strlen(array) > ary\_size - 1) || cin.fail()) {

std::cout << "wrong input, try again" << std::endl;

cin.clear();

std::cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

else

correct = true;

} while (!correct);

}

# **7. Результати тестування програми та їх аналіз.**

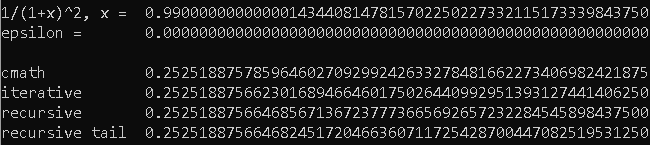


Рисунок 4

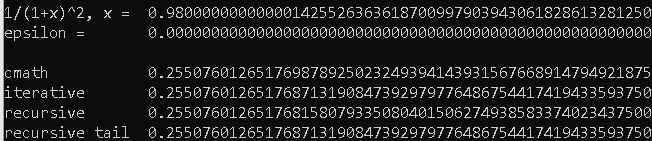


Рисунок 5

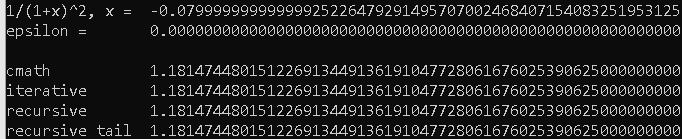


Рисунок 6

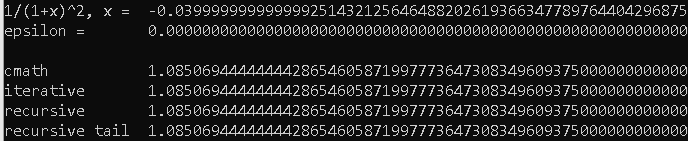


Рисунок 7

У більшості випадків результат повернений функцій достатньо точний як у випадках показаних на рисунках 4-7. Та навіть повністю співпадають із результатом функції розробленої з допомогою стандартної бібліотеки, як на рисунках 6, 7.

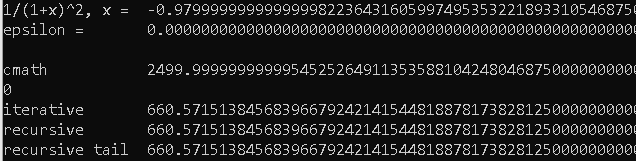


Рисунок 8

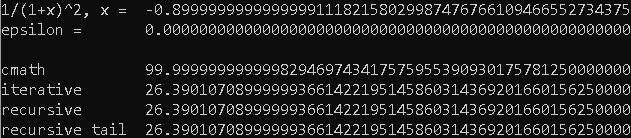


Рисунок 9

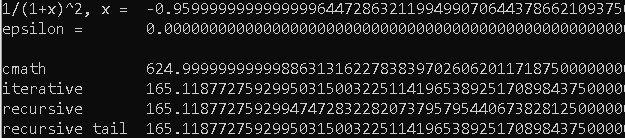


Рисунок 10

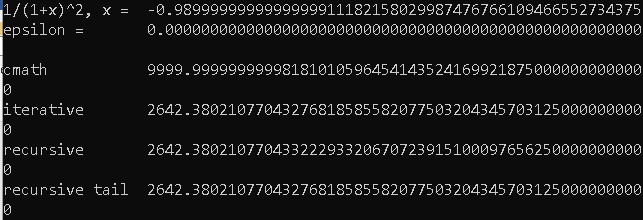


Рисунок 11

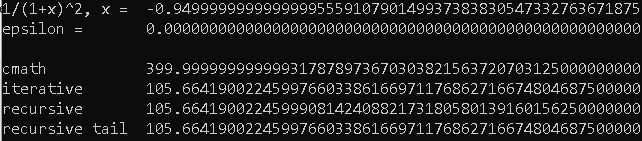


Рисунок 12

Але в деяких випадках результат достатньо далекий від наданого функцією розробленою з використанням стандартної бібліотеки.

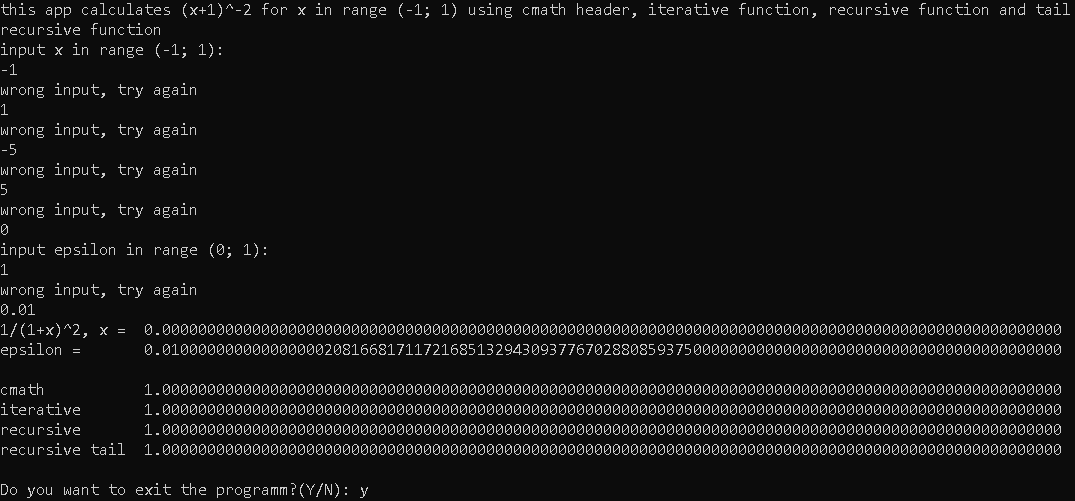


Рисунок 13(Валідація вхідних даних)

# **8. Висновки щодо використання рекурсії та ітерації при обчисленні функції.**

Код рекурсивних функцій дещо коротший ніж код таких самих ітеративних функцій. Але ітераційна функція викликається один раз, коли рекурсивні варіанти можуть викликатися неуявну кількість разів. Якщо підсумувати іноді рекурсивні варіанти функцій можуть забезпечити трохи простіший код, але ціною швидкості, іноді кількістю аргументів, лімітом розрахунків. Але на мою думку ітераційний варіант не набагато складніше читати і його плюси набагато більше ніж це єдиний мінус.