SCHOOL OF ENGINEERING

Integrated Engineering

Stenxxxxx

„Homework 1 – mathematic function. “

Homework

Supervisor Vladimir Viies

Tallinn 2023

„I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication. All works and major viewpoints of the other authors, data from other sources of literature and elsewhere used for writing this paper have been referenced. “

**The problem and the general structure of the program.**

The program works around the mathematical function (7x^(2)+18x-14) / sqrt(4-x^(2))

 

Due to sqrt(4-x^(2)) and that it is the enumerator, the domain of the function is ]-2;2[

This is because the value under a square root can not be negative, otherwise the result will be a complex number.

While making the program, I have made all the variables and function names as descriptive as possible, while not being too long. This helps with the code’s readability.

At the start of the main function, the following variables are declared:

* const int MAX\_ITERATIONS = 15;
* int total\_numbers = MAX\_ITERATIONS;
* float starting\_value; // A
* float steping\_value; // H
* float max\_value\_for\_y; // YM

Then the program gets the user inputs, calculates all of the solutions, outputs all x’s and y’s and finally exits the program with return 0;.

The algorithm can be located on the last page.

Incudes, prototypes and the main function:

#include<stdio.h>

#include<math.h>

// Prototypes.

float ask\_for\_value(char[]);

float ask\_for\_steping\_value(char[]);

int compute\_math\_function(float, float, float, float[], float[], int[], int);

float calculate\_y(float);

void print\_func\_results(float[], float[], int[], int);

int main(void){

    const int MAX\_ITERATIONS = 15;

    int total\_numbers = MAX\_ITERATIONS;

    float starting\_value;     // A

    float steping\_value;      // H

    float max\_value\_for\_y;    // YM

    // Input.

    starting\_value  = ask\_for\_value("Input a starting value: \0");

    steping\_value   = ask\_for\_steping\_value("Input a stepping value: \0");

    max\_value\_for\_y = ask\_for\_value("Input a maximum value: \0");

    // Proccess.

    float x\_array[MAX\_ITERATIONS];

    float y\_array[MAX\_ITERATIONS];

    int y\_is\_real[MAX\_ITERATIONS];

    total\_numbers = compute\_math\_function(starting\_value, steping\_value, max\_value\_for\_y, x\_array, y\_array, y\_is\_real, MAX\_ITERATIONS);

    // Output.

    print\_func\_results(x\_array, y\_array, y\_is\_real, total\_numbers);

    return 0;

}

**Input.**

The program has 3 input functions. These are the following:

* float ask\_for\_value (char question[])
* float ask\_for\_steping\_value(char question[])

The function ask\_for\_value takes in a question and asks the user for a number. Any floating-point number is accepted. If the user fails to provide a number, then the program asks for that number again. The function returns the number asked from the user.

The function ask\_for\_steping\_value works similar to the previous function, however the returning value can not be negative. If that criteria is not met, the program asks that value again. The function returns the stepping value, which was asked from the user.

Both functions check if the input value is correct, this is because if the variable does not match its variable type, the program fails. By default, the integer variable input\_success is set to 0. If the input is successful, then the input success is set to 1, otherwise it is set to 0. An if statement checks that the variable is not 1. If this criteria is met, the program will scan everything in the input until it reaches \n, it included, and discards it all. This prevents the program from failing. However, this safety check may not work on Linux operating systems.

The functions:

float ask\_for\_value(char question[]){

    int input\_success = 0;

    float value;

    do{

        printf("%s", question);

        input\_success = scanf("%f", &value);

        if(!input\_success){

            scanf("%\*[^\n]"); // "%\*[^\n]" -> scan everything, including "\n" and discards all of it.

            printf("Unknown value! ");

        }

    } while((!input\_success));

    return value;

}

float ask\_for\_steping\_value(char question[]){

    int input\_success = 0;

    float value;

    do{

        printf("%s", question);

        input\_success = scanf("%f", &value);

        if(!input\_success){

            scanf("%\*[^\n]"); // "%\*[^\n]" -> scan everything, including "\n" and discards all of it.

            printf("Unknown value! ");

        }

        else if (value <= 0)

        {

            printf("Must be bigger than 0! ");

        }

    } while((!input\_success) || (value <= 0));

    return value;

}

**Proccess.**

After the program has gathered all the necessary inputs, it will proceed with the calculations.

At first, the program will create 3 arrays with maximum length of 15.

Calculations consist of these functions:

* int compute\_math\_function( float starting\_value, float steping\_value, float max\_value\_for\_y, float x\_array[], float y\_array[], int y\_is\_real[], int max\_iterations)
* float calculate\_y(float x)

compute\_math\_function:

Said function has a lot of variables, all of them are used to calculate all the values for the mathematical function mentioned on the page The problem and the general structure of the program.

Inside the function, the program edits the arrays float x\_array, float y\_array, int y\_is\_real and returns how many solutions there are, default is 15 but it may be lower, if y becomes bigger than max\_value\_for\_y.

The array x\_array stores all the x values. The array y\_array stores all the results of the mathematical function, if the result is a complex number, then it is set to 0. The array y\_is\_real keeps track of which result is a real number, and which is a complex number.

calculate\_y:

Solves the mathematical function (7x^(2) + 18x - 14) / (sqrt(4 - x^(2))) and returns it.

This function will never fail because compute\_math\_function checks if x is in the range of the domain ]-2;2[.

The functions:

// Returns the total amout of values while considering the max\_value\_for\_y limit.

int compute\_math\_function( float starting\_value, float steping\_value, float max\_value\_for\_y,

                            float x\_array[], float y\_array[], int y\_is\_real[],

                            int max\_iterations){

    int i;

    float x;

    float y;

    for(i = 0; i < max\_iterations; i++){

        x = starting\_value + steping\_value \* i;

        if(-2 < x && x < 2){ // If realnumber from the formula.

            y = calculate\_y(x);

            if (max\_value\_for\_y < y){

                return i;

            }

            y\_array[i] = y;

            y\_is\_real[i] = 1;

        }

        else{

            y\_array[i] = 0;

            y\_is\_real[i] = 0;

        }

        x\_array[i] = x;

    }

    return max\_iterations;

}

float calculate\_y(float x){ // Real numbers only.

    float y;

    float numerator = 7 \* (pow(x, 2)) + (18 \* x) - 14;

    float denominator = sqrt(4 - pow(x, 2));

    y = numerator / denominator;

    return y;

}

**Output.**

Output consists of 1 function:

* void print\_func\_results(float x\_array[], float y\_array[], int y\_is\_real[], int total\_numbers)

The function uses all the 3 arrays float x\_array, float y\_array, int y\_is\_real to output all the values of x and the results of the mathematical function.

The function knows how many numbers there are with total\_numbers.

The function:

void print\_func\_results(float x\_array[], float y\_array[], int y\_is\_real[], int total\_numbers){

    int i;

    printf("\n\n");

    for(i = 0; i < total\_numbers; i++){

        printf("x: %f\t y: ", x\_array[i]);

        if(y\_is\_real[i]){

            printf("%g\n", y\_array[i]);

        }

        else{

            printf("Complex number\n");

        }

    }

}

**The algorithm.**

The functions are colored to assist with clarity.

