AIN SHAMS UNIVERSITY, FACULTY OF ENGINEERING

Mechatronics Engineering and Automation Program, i-Credit Hours Engineering Programs

Spring Semester 2020 | **Course Code: CSE131** | *Time Allowed:*

CSE131 Computer Programming– Final Exam

The Exam Consists of -- Questions in --Pages.

Ain Shams University
Faculty of Engineering

University of East London

Ain Shams University

Programme **Mechatronics Engineering and Automation Program**		
UEL Module Code **EG8313**	*UEL Module Name* **Circuit analysis and Programming**	
ASU Course Code **CSE131**	*ASU Course Name* **Computer Programming**	
Assessment weighting **40 Marks**	*Semester* **Spring 2020**	*Examination Date* ----------

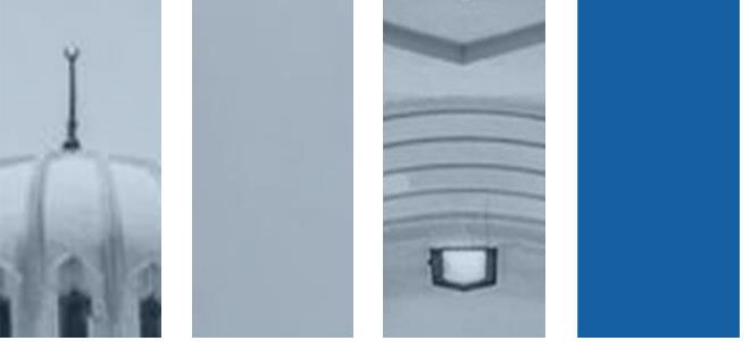
INSTRUCTIONS TO CANDIDATES

1. Organise your answer sheet so that you devote a separate zone/section for each question. Write clearly the question number on the top left corner of every section.
2. Write your ID clearly in the Student's IDs section on the paper sheet.
3. The coversheet must be stapled to the paper sheet.
4. Read all questions and instructions carefully

UEL Grading System	Agreed Mark Range	ASU Grading Scale	
% equivalent at UEL		% at ASU	Grade
95% and higher		97% and higher	A+
82% to less than 95%		93% to less than 97%	A
70% to less than 82%		89% to less than 93%	A-
66% to less than 70%		84% to less than 89%	B+
63% to less than 66%		80% to less than 84%	B
60% to less than 63%		76% to less than 80%	B-
56% to less than 60%		73% to less than 76%	C+
53% to less than 56%		70% to less than 73%	C
50% to less than 53%		67% to less than 70%	C-
45% to less than 50%		64% to less than 67%	D+
40% to less than 45%		60% to less than 64%	D
Less than 40%		Less than 60%	F

Question number(s)	*1st Marker*	*2nd Marker*	*Agreed Mark*
Total			
1st Marker Signature			
2nd Marker Signature			

Student ASU ID No.	*Student UEL ID No.*	*Student's Signature*
18P8912	***u2023587.***	***Thomas Medhat Mounir Botros***



Computer Programming – *CSE131* Report

Program: MCTA

UEL Module Code: ***EG8313***

Course Code: ***CSE131***

UEL Module Name:

Circuit analysis and Programming

Course Name:

Computer Programming

Examination Committee

XXXX

XXXX

XXXX

XXXX

Ain Shams University
Faculty of Engineering

Spring Semester – 2020

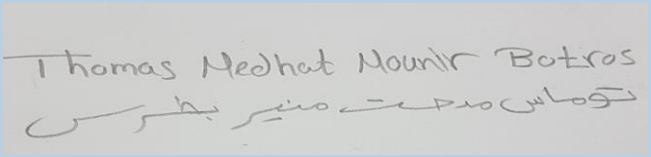
Student Personal Information

Student Name: Thomas Medhat Mounir Botros
Student ASU Code: 18P8912
Class/Year: Sophomore/Spring 2020
Student UEL Code: u2023587

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student Name: **Thomas Medhat Mounir Botros** **Date:** **11/6/2020**



Submission Contents

01

First Topic

Introduction to root finding

Root finding :

Root finding appear in a wide practical application like Engineering , Physics , chemistry and biosciences.

Root finding problem is a computational problems . as it appear in a wide viral practical applications.

In Engineering for example root finding is very important for several engineering specializations.

It is used in automatic control design and in solving optimization problems .also root finding is very crucial in machine learning.

As a matter of fact ,in engineering or scientific formulas determination of any unknow that appear in the formula is very important and determining the roots of the formula is also very important .

Root finding problem :

The root finding problem is having a problem to find the root of equation f(x)=0 where f(x) is a function of a single variable which is x .

As for some function it is not a problem to find its roots for example

f(x) = ax^2 + bx + c to get the root of this function there is an easy formula to get it

x=(-b $\pm \sqrt{b^2 - 4 * a * c}$) /(2*a) where x if the value that make f(x)=0 i.e. the root of f(x) .

For some other function there is no such formula exists or there is a formula but it is very difficult.

Root finding problem dates back to the 18th century .

To overcome this problem different methods were created but they don't give the exact value it give an approximate value of the root .

02

Second Topic

Some Methods for root finding

It was important to find the root of any function that we know of to avoid that problem therefore the is a lot of numerical methods to find the root of the function f(x) they do not give the exact value they give an approximate value of the root .

They are easier than any other method that give the exact value of the root and much faster.

Most of those method are numerical methods as it is easier to work with numbers than variable.

Most of the these methods uses iteration and producing a sequence of numbers that converge to toward the root as a limit. Some of them may require one or more initial guesses for example to guess the first root then using iteration it produces a more accurate approximation of the root.

Numerical methods divide to :

One point method : one point method means that you need to estimate one value for the method to work .

Two point method : .Two point method means that you need to estimate two value for the method to work .

Some methods to find the root of a certain equation :

Bisection method :

The method is based on repeated bisection of an interval containing the root as it bisect the interval and then make iteration to get the nearest value of the root.

In this method you need to estimate two starting values which will make the interval it is a two point method .

This method is slow but gives a very near approximation of the exact value of the root.

Newton Raphson method :

In this method you need to estimate the first root x_n and then using iteration step n to get x_{n+1} , x_{n+2} and so on . also we need to know the derivative of the function

As $x_{n+1} = x_n - (f(x) / f'(x))$

This method is probably the most well known method for root finding as it give fast result also there is no need to assume any value except the starting value also in many cases there is no conditions to satisfy this is a one point method.

Regula Falsi Method : aka False Position Method

It is also a two point method as Bisection method yet it is much faster than bisection as it use some information about the function to faster solution like newton Raphson method .
This method is like a combination between Bisection Method and Newton Raphson Method.
As it uses Bisection Method but instead of bisect the interval without knowing any information about the function Regula Falsi Method divide the interval closer to the true value of the root using information about the function as Newton Raphson Method .
Therefore it needed less iterations than Bisection Method.
It still slower that Newton Raphson Method.

Secant Method:

This method can be explained as its Newton Raphson Method without derivative at each iteration .
This method is faster than Regula Falsi Method but it may not converge at all

03

Third Topic

Bisection Method

In the code to find the root of any polynomial function I used **Bisection Method** .

Bisection Method :

In this method you need to estimate two starting values which will make the interval it is a two point method .

The method is based on repeated bisection of an interval containing the root as it bisect the interval and then make iteration to get the nearest value of the root.

Therefore you estimate an interval that will contain the real root of the function for example You need to estimate two initial guesses x_a and x_b your estimation need to satisfy one condition which is that f(x_a)* f(x_b)<=0 .

If the condition was satisfied then there is a root between the two initial guesses .

If the condition was not satisfied then there is NO root between the two initial guesses .

The Bisection Method is over when the value reach the required tolerance for example 0.0001. This method is slow but gives a very near approximation of the exact value of the root.

Bisection Method Implementation :

1) Obtain two initial guesses x_a and x_b and the required tolerance
2) Check the Condition f(x_a)* f(x_b)<=0 if the condition is satisfied continue if not stop there is no root between the two initial guesses
3) Create the next iteration x_c=(x_a+ x_b)/2
4) Find f(x_c)
5) If f(x_c) = 0 or the required tolerance is reached stop if not continue
6) Repeat using [x_a, x_c] or [x_c, x_b] it depend on the sign of x_c and which of $x_a$and x_b has the same sign as x_c
7) Move to next iteration tell f(x_c) = 0 or the required tolerance is reached

04

Fourth Topic

Conclusion

Root finding appear in a wide Practical application like Engineering , Physics , chemistry and biosciences.
In Engineering for example root finding is very important for several engineering specializations.

Root finding problem also appear in a wide viral applications like automatic control design.

Having a problem in finding the value of the variable that make the function equal zero is a real problem to deal with. But not all the function have root finding problem like f(x) = ax^2 + bx + c

To get the root there is an easy formula for that.

Not all functions have an easy formula those function are where the root finding problem appear .

Since root finding is a problem that we need to overcome .

Different methods were created for example

1) Bisection Method
2) Newton Raphson Method
3) Regula Falsi Method (False Position Method)
4) Secant Method

They all are Numerical Methods they don't give the exact value of the root but gives a very near approximation of the exact value of the root. They all use iteration and , producing a sequence of numbers that converge to toward the root as a limit.

In the code to find the root of any polynomial function I used **Bisection Method** .

Bisection Method :

The method is based on repeated bisection of an interval containing the root as it bisect the interval and then make iteration to get the nearest value of the root.

This method always converges to the root .

It is a two point method .

it is simple and easy to implement .

it is a slow method .

it need two guesses that falls under a certain condition .

it is simple and easy to implement .

you need two initial guesses and the required tolerance then Check the Condition if the condition is satisfied create the next iterations till the required tolerance is reached

05

Fifth Topic

Code Algorithms Clarification

The code starts with the header files needed .

#include<iostream> which is basic in any code.

#pragma warning (disable : 4996) because I am using strtok and the that must be included for it to work.

using namespace std; which allow me to use cout ,cin ,endl and couple of other things also basic in any code .

double EP = 0.00000001; then define EP as double type and EP =0.00000001 where EP is the tolerance required and it was defined for all functions in the code.

// float horner(int* coefficient, int n, float x)
Then define a function to get the value of the function entered at any x function name is horner
Where Horner is a method used to find the value of the polynomial function
float horner(int* coefficient, int n, float x) float horner function is defined as float because the value of the function will be in float type
int* coefficient where * coefficient is the dynamic array where the coefficient is going to be sorted.
int n where n is the degree of the polynomial equation .
float x where x is the variable of the polynomial equations the type of float as it can be any number.
Then function code float result = 0; where result is f(x)
Then for loop with counter I that give the value of result f(x) without the constant part
So result = result + coefficient[n]; the constant part is added to the value of f(x) and then return result so when the function is called it give the value of the function at the x entered
Horner function is used to check that the value of the roots output is right as they must give zero

// inline float f(float x, int eq[], int n)
This function is used to decrease debugging time as the method used to find roots is Bisection method which takes a lot of time.

This function take three input which are float x the value of x , int eq[] which is the array containing the coefficients of the polynomial equation and int n which is the degree of the polynomial function

In this function double y=0 where y = f(x) the value of the function at a certain x
Int i=0 counter to reach each element in the array that contains the coefficient

// while(n>=0)
While loop that find the value of the polynomial equation

This function return y the value of the polynomial function

// double bisection(float a, float b, int eq[], int n)
This function represents the Bisection method

It take 4 inputs which are
Float a and float b which are the two guesses needed
Int eq[] is the array that contain the coefficient
Int n is the degree of the polynomial equation

// function code

Double c = (a + b)/2 where c is the root of the polynomial
C_before is the value of the root in the iteration
Y which is f(x)

// do while loop
y = f(c, eq, n) using function inline f to get the vale of the function when x=c

// if (y>0)
Need more iteration
In this case a=c and c_before =c and move in the interval [c,b]

// else if (y<0)
Need more iteration
b=c and c_before =c therefore work on the interval [a,c]

// else if(y==0)
In this case you have reached the root of the equation

// while (fabs(c-c_before)>=pow(10,-6))
This is the condition of the loop so that the loop stop also when the required tolerance is reached.

Then return the value of the root c

// void print_roots(int eq[], int n, int a, int b)

This function is used to print the roots on the screen and also to check the intervals it takes four input
int eq[] the array where the coefficients are stored
int n the degree of the polynomial
int a and int b the two guesses they are entered in int type as there will counter j that will equal a so there is no counter count in float type

int i=0 is a counter to reach each element in the array roots
float roots [10] array where the roots are stored

// for (int j = a; j <= b; j++)
This for loop is for reach the interval [a , b]
j++ counter : if input intervals are for example [-5,5],the program find roots between each unit

(roots between -5 and -4 , -4 and -3 and so on till 5), to avoid finding only one root in the same interval and if two roots are found in the same unit we can increase precision by DECREASING increment value , but will slow debugging time in an obvious effect

Float y1 and y2 are the value of the polynomial we will get it using inline f function
For example if j=2
y1 = f(j, eq, n); y1 = f(2)
y2 = f(j + 1, eq, n); y2=f(3)

// if(y1==0)
In this case root is j

// else if (y1 > 0 && y2 < 0)
In this case its interval so bisection method
root[i] = bisection(j, j + 1, eq, n); bisection function is called to get the root out of this interval

// else if (y1 < 0 && y2 >0)
Same as the one before but with different arranging

root[i] = bisection(j + 1, j, eq, n); bisection function is called to get the root out of this interval

in all cases counter I increase

// if(i>0)
Then there is roots of the polynomial equations within the interval entered

// for (int count = 0; count < i; count++)
To print root of the function

// for (int count = 0; count < i; count++)
To check the roots
(float)(int)horner(eq, n, root[count]) to change the function to integer horner function is called

// else
There is no roots of the polynomial equations within the interval entered

// main function //

// some declaration needed
Like size of the string const int stringsize = 1000;
char st[stringsize]; the string where the polynomial equation is stored

char* s1; for dynamic array which will be a copy of st polynomial that will help in tokenization of the string

char* ptr; will be used for tokenization and extract coefficient
char* search; will be used for tokenization and extract coefficient
char* eqptr; pointer of st polynomial equation to reach every element in st
int* coefficient = NULL; dymanic array where coefficient will be stored

int counter = 0; counter for printing coefficient
int count = 0; counter to extract coefficient
int errors = 0; counter of errors
int n = 0; power of polynomial equation
float a = 0.0; start point of domain
float b = 0.0; end point of domain

// introduction to the user
Abstract of the program
Shape to enter the polynomial equation in it
Some precautions the user need to avoid so there is no errors

// letting the user input the polynomial equation
Using cin.getline() so the user input the polynomial as string

// copy st in s1
Create a dynamic array and copy in it the polynomial equation using strcpy()

// check for errors codes //
Let pointer eqptr point to st string to access the elements in it

// to check that if the user did not enter polynomial equation on the form (space)4x^4 - 3x^3 + 2x^2 -1x^1 -10

// space at the beginning check as mentioned in introduction

// if(eqptr[0]!= ' ')
Print the problem and increase the counter of errors

// no space between -ve and coefficient as mentioned in introduction

// for loop for (int i = 1; i < strlen(eqptr); i++) strlen to get length of the array st
To access every element in the string except element 0 which must be space

// if(eqptr[i] == '-')
To only apply when -ve sign appear

// if (eqptr[i + 1] == ' ')
If there is a space that is an error so print that error and increase counter error

// check space before and after +ve

// for loop for (int i = 1; i < strlen(eqptr); i++)
To access every element in the string except element 0 which must be space

// if(eqptr[i] == '+')
To only apply when +ve sign appear

// if (eqptr[i - 1] == ' ' && eqptr[i + 1] == ' ')
There is space before and after +ve sign break from the loop

// else
There is no space before and after the +ve sign print the problem

// to check that user did not input a sign and then nothing more

// for loop for (int i = 1; i < strlen(eqptr); i++)
To access every element in the string except element 0 which must be space

// if (eqptr[i] == '+' || eqptr[i] == '-')
To only apply when any sign appear

// if (eqptr[i + 1] == '\0')
There is an error print the error and increase the error counter

// to check that user did not enter for example (1x1) insted of (1x^1)

// for loop for (int i = 1; i < strlen(eqptr); i++)
To access every element in the string except element 0 which must be space

// if(eqptr[i] == 'x')
To only apply when +ve sign appear

// eqptr[i] == 'x' && eqptr[i + 1] == '^'
There is no problem

//else
There is a problem and print it out and increase the error counter

// to check that user did not enter for example (1x^+) or (=)

// for loop for (int i = 1; i < strlen(eqptr); i++)
To access every element in the string except element 0 which must be space

// if (eqptr[i] == '^' && eqptr[i + 1] == ' ')
There is a problem print it out and increase the error counter

// else if (eqptr[i] == '=')
There is a problem print it out and increase the error counter

// to check that user did not enter for example (x^2) instead of (1x^2)

// for loop for (int i = 1; i < strlen(eqptr); i++)
To access every element in the string except element 0 which must be space

// if ((eqptr[i] == 'x' && eqptr[i - 1] == ' ') || (eqptr[i] == 'x' && eqptr[i - 1] == '-'))

There is a problem print it out and increase the error counter

// to check that user did not enter any letter except (x)

// for loop for (int i = 1; i < strlen(eqptr); i++)
To access every element in the string except element 0 which must be space

// if (eqptr[i] >= 'a' && eqptr[i] <= 'z')
To find any letter entered in the polynomial equation

// if (eqptr[i] - 'x' != 0)

There is a problem print it out and increase the error counter

//NO Error code

// if(errors == 0)
Error counter errors is zero this mean that there is no error in the polynomial input by the user
If error not equal zero it will print that there is a problem

// extract coefficients
ptr = strtok(st, " +"); to token the polynomial equation in st

// while loop while (ptr)

// search = strstr(ptr, "x^"); return the pointer to the first occurrence of "x^"

// ptr = strtok(NULL, " +"); another tokenization

// counter ++ counter count the number of terms of the polynomial equation

// to identify the degree of the polynomial equation
Counter count number of terms of the polynomial equation

// n=counter-1
Degree of the polynomial equation = number of terms of the polynomial equation-1
Then print the degree of the polynomial

// to find coefficient

// coefficient = new int[counter];
Dynamic array where coefficients are stored

// ptr = strtok(s1, " +");
To token the string s1 which contains the copy of st where polynomial equation is stored

// while loop while (ptr)

// long coeff; will be needed in strtol

// if (*ptr == 'x') Then coeff =1

// else if (*ptr == NULL) then coeff =0 ;

// else

char* endptr; pointer needed for strtol

// coeff = strtol(ptr, &endptr, 10)

strtol interprets the contents of a string as an integral number of the specified base and return its value as a long int therefore we can print the coefficient

// coefficient[count] = coeff;
To store coefficient in the dynamic array coefficient

// ptr = strtok(NULL, " +"); tokenization count increase

// to print the coefficient

// for loop for (int i = 0; i < counter; i++)
To have access to every element in the array

// to let user enter domain of function
Print some precautions to avoid errors
Read the the domain entered by the user

// print_roots(coefficient, n, a, b)
Calling the function that takes for inputs to solve the polynomial ,check interval and print roots

delete[]coefficient; close dynamic array
delete[]s1; close dynamic array

// return 0 basic in any code to pause the program

06

Sixth Topic

Appendix

```
// this program calculate roots of any polynomial equation
// header files needed
#include<iostream>
#pragma warning (disable : 4996 )  // for strtok
using namespace std;

double EP = 0.00000001;   // epsilon is percision

float horner(int* coefficient, int n, float x)        // horner method to find the value of F(x) at
any x
{
        float result = 0;  // Initialize result
        //  to Evaluate value of polynomial using Horner's method
        // n is power of polynomial
        for (int i = 0; i < n; i++)
        {
                result = (result + coefficient[i]) * x;
        }
        result = result + coefficient[n];
        return result;     // F(x)
}//end of horner function


inline float f(float x, int eq[], int n)    //inline function as to decrease debugging time.
{
        double y = 0;   // y=f(x)
        int i = 0;     // counter
                while (n >= 0)  // n is power
                {
                        y += eq[i] * pow(x, n); // to get the value of f(X) ex : y=f(x)= 3(2)^1+2(2)^0
                        i++;
                        n--;
                }
                return y;
} // end of f function
```

```
double bisection(float a, float b, int eq[], int n)    //BISECTION METHOD OF ROOT FINDING
{
        double c = (a + b) / 2, c_before = 0, y;
        do
        {
                y = f(c, eq, n);
                if (y > 0)
                {
                        a = c;    c_before = c;
                }
                else if (y < 0)
                {
                        b = c;    c_before = c;
                }
                else if (y == 0)
                        break;
                c = (a + b) / 2;
        }
        while (fabs(c - c_before) >= pow(10, -6)); // condition to exit while loop when the
percentage of difference between the output & exact value less than 0.00000001%

        return c;
}// end of bisection function


void print_roots(int eq[], int n, int a, int b)  // to print the roots of polynomial function
{
        int i = 0;
        float root[10];      // can calculate and get output to roots for same equation
        for (int j = a; j <= b; j++)

                /*j++ counter : if input intervals are for example [-5,5],the program find roots
between each unit
                ( roots between -5 and -4 , -4 and -3 and so on till 5), to avoid finding only one
root in
                the same interval and  if two roots are found in the same unit we can increase
precision by
                DECREASING increment value , but will slow debugging time in an obvious effect
*/
        {
                float y1, y2;
                y1 = f(j, eq, n);
                y2 = f(j + 1, eq, n);
```

```
                if (y1 == 0)

                {
                        root[i] = j; i++;
                }


                else if (y1 > 0 && y2 < 0)

                {
                        root[i] = bisection(j, j + 1, eq, n);    i++;
                }

                else if (y1 < 0 && y2 >0)

                {
                        root[i] = bisection(j + 1, j, eq, n);   i++;
                }

        } // end for
        if (i > 0)
        {
                cout << "\nRoot(s) of equation :-\n\n";
                for (int count = 0; count < i; count++)
                {
                        cout << "Root " << count + 1 << " = " << root[count] << endl << endl;
                }
                // to check roots
                for (int count = 0; count < i; count++)
                {
                        cout << "Check of Bisection method \n value of F( " << root[count] << " ) =
" << (float)(int)horner(eq, n, root[count]) << endl << endl;
                }
        } // end if
        else
        {
                cout << "No Roots for this polynomial Equation." << endl << endl;
        }
} // end of print_roots function


// main function
```

```
int main() {

        // some decleration
        const int stringsize = 1000;      //  size for string input
        char st[stringsize];           // the string which will store the polynomial equation input by
user
        char* s1;                  // for dynamic array which will be a copy of st polynomial
equation
        char* ptr;                 // for tokenization and extract coefficient
        char* search;               // for tokenization and extract coefficient
        char* eqptr;               // pointer of st polynomial equation
        int* coefficient = NULL;    //  dymanic array where coefficient will be stored
        int counter = 0;          //  counter for printing coefficient
        int count = 0;           // counter  to extract coefficient
        int errors = 0;          // counter of errors
        int n = 0;             // power of polynomial equation
        float a = 0.0;          // start point of domain
        float b = 0.0;         // end point of domain


        // introdunction to user
        cout << "\n\t\t\tThis Program for calculating the root of any Polynomial equation .
\n\t\t\t F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0 \n";
        cout << "\nPRECAUTIONS TO AVOID ERRORS :- \n 1)start your function with space \n
2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7\n3) All Letters must be
lowercase\n 4) ALL terms of the Polynomial must be entered even zero coefficient one \n
EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0 \n\n";




        // to let the user input the polynomial equation
        cout << "Enter the Polynomial Equation  F(X) = ";

        cin.getline(st, stringsize, '\n');  // to get the polynomial as string
        cout << endl << endl;

        // copy of st in s1
        s1 = new char[stringsize];  // dynamic array

        strcpy(s1, st);    // to copy polynomial in st in s1


                                    // check for errors codes
```

```
        eqptr = st;   // pointer of st

        // to check that if  the user did not enter polynomial equation on the form   (space)4x^4
-3x^3 + 2x^2 -1x^1 -10


                // to check that there is space at start of the polynomial equation
        if (eqptr[0] != ' ')
        {
                cout << "ERROR : No Space entered at start of the polynomial equation \n \n";
                errors++;
        }

        // to check that user did not enter spaces bewteen negative sign and coefficient
        for (int i = 1; i < strlen(eqptr); i++)
        {
                if (eqptr[i] == '-')
                {
                        if (eqptr[i + 1] == ' ')
                        {
                                errors++;
                                cout << "ERROR: there is Space between negative sign and
coefficient\n";
                                cout << "No Space should be between negative sign and
coefficient \n\n";
                                break;
                        }
                }
        }

        // to check that there is space before and after positive sign
        for (int i = 1; i < strlen(eqptr); i++)
        {
                if (eqptr[i] == '+')
                {
                        if (eqptr[i - 1] == ' ' && eqptr[i + 1] == ' ')
                        {
                                break;
                        }
                        else
                        {
                                errors++;
                                cout << "ERROR : There is no space before and after the + sign
\n\n";
```

```
                }
            }
        }

        // to check that user did not input a sign and then nothing more
        for (int i = 1; i < strlen(eqptr); i++)
        {
            if (eqptr[i] == '+' || eqptr[i] == '-')
            {
                if (eqptr[i + 1] == '\0')
                {
                    errors++;
                    cout << "ERROR : There is a sign with nothing after it as example
4x^4 -3x^3 + 2x^2 -1x^1 +  \n\n";
                    break;
                }
            }
        }

        // to check that user did not enter for example (1x1) insted of (1x^1)
        for (int i = 1; i < strlen(eqptr); i++)
        {
            if (eqptr[i] == 'x')
            {
                if (eqptr[i] == 'x' && eqptr[i + 1] == '^')
                {

                }
                else
                {
                    cout << "ERROR: Polynomial equation contain x without power
sign example x1\n\n";
                    errors++;

                }
            }
        }

        // to check that user did not  enter for example (1x^+) or (=)
        for (int i = 1; i < strlen(eqptr); i++)
        {
            if (eqptr[i] == '^' && eqptr[i + 1] == ' ')
            {
                errors++;
```

```
                cout << "ERROR : Polynomial equation entered contain for example x^+
\n\n";
            }
            else if (eqptr[i] == '=')
            {
                errors++;
                cout << "ERROR : Polynomial equation entered contain = \n\n";
            }

        }

        // to check that user did not enter for example (x^2) instead of (1x^2)
        for (int i = 1; i < strlen(eqptr); i++)
        {
            if ((eqptr[i] == 'x' && eqptr[i - 1] == ' ') || (eqptr[i] == 'x' && eqptr[i - 1] == '-'))
            {
                errors++;
                cout<< "ERROR : Polynomial equation entered contain for example + x^2
no coefficient is entered  \n\n";
            }
        }


        // to check that user did not enter any letter except (x)
        for (int i = 1; i < strlen(eqptr); i++)
        {
            if (eqptr[i] >= 'a' && eqptr[i] <= 'z')
            {
                if (eqptr[i] - 'x' != 0)
                {
                    errors++;
                    cout << "ERROR: Polynomial equation should be in one variable
which is x \n\n";
                    break;
                }
            }
        }


            //  NO ERROR CODE
            if(errors == 0)
            {
                // extract the coeficients
                ptr = strtok(st, " +");    // to token the polynomial equation in st
```

```
while (ptr)
{
        search = strstr(ptr, "x^");        // Returns a pointer to the first occurrence of x^ in pt
        ptr = strtok(NULL, " +");
        counter++;              // counter count number of terms of the polynomial equations
}

// to identify the degree of function

n = counter - 1;            // degree = number of terms - 1
cout << "The Polynomial equation degree = " << n << endl << endl;   // to print the degree of polynomial equation


// find coefficient

coefficient = new int[counter];    // dynamic array to store coefficient

ptr = strtok(s1, " +");        // to token the polynomial equation in st

while (ptr)
{

        long coeff;
        if (*ptr == 'x')
        {
                coeff = 1;
        }
        else if (*ptr == NULL)
        {
                coeff = 0;
        }
        else
        {
                char* endptr;
                coeff = strtol(ptr, &endptr, 10);      // strtol interprets the contents of a string as an integral number of the specified base and return its value as a long int.
        }
        coefficient[count] = coeff;
        ptr = strtok(NULL, " +");
```

```
                count++;
            }
            cout << "Cofficient are : \t" << endl << endl;

            // to print the coefficient
            for (int i = 0; i < counter; i++)
            {
                cout << coefficient[i] << " \t ";
            }
            cout << endl << endl;

            // to let user enter domain of function

            cout << "Enter the domain : \n ";

            cout << "\nPRECAUTIONS TO AVOID ERRORS :- \n 1)Enter the start point
then end point (smaller number then larger number)\n\n\n";

            cout << "Start Point (a) = ";
            cin >> a;

            cout << endl;

            cout << "End Point (b) = ";
            cin >> b;

            cout << endl << endl;

            print_roots(coefficient, n, a, b); // void function to print

            delete[]coefficient;  // close dynamic array
            delete[]s1;        //  close dynamic array

        }//end if
        else
        {
            cout << "ERROR ERROR THERE IS AN ERROR THAT NEED TO BE FIXED
\n\n";
        }
        cout << "\n\t\t\t\t\tEND OF PROGRAM\n\n\n\n";
        return 0;
}// end main
```

07

Seventh Topic

Testing cases

Case 1 :

```
Microsoft Visual Studio Debug Console

                        This Program for calculating the root of any Polynomial equation .
                         F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  1x^3 -2x^2 -1x^1 + 2


The Polynomial equation degree = 3

Cofficient are :

1        -2      -1      2

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -3

End Point (b) = 3



Root(s) of equation :-

Root 1 = -1

Root 2 = 1

Root 3 = 2

Check of Bisection method
 value of F( -1 ) = 0

Check of Bisection method
 value of F( 1 ) = 0

Check of Bisection method
 value of F( 2 ) = 0


                                        END OF PROGRAM
```

Case 2 :

```
Select Microsoft Visual Studio Debug Console

                        This Program for calculating the root of any Polynomial equation .
                         F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  1x^4 + 0x^3 -5x^2 + 0x^1 + 4


The Polynomial equation degree = 4

Cofficient are :

1        0       -5      0       4

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -3

End Point (b) = 3


Root(s) of equation :-

Root 1 = -2

Root 2 = -1

Root 3 = 1

Root 4 = 2
```

```
Check of Bisection method
 value of F( -2 ) = 0

Check of Bisection method
 value of F( -1 ) = 0

Check of Bisection method
 value of F( 1 ) = 0

Check of Bisection method
 value of F( 2 ) = 0


                                        END OF PROGRAM
```

Case 3 :

```
Microsoft Visual Studio Debug Console

                        This Program for calculating the root of any Polynomial equation .
                         F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  1x^5 -3x^4 -5x^3 + 15x^2 + 4x^1 -12


The Polynomial equation degree = 5

Cofficient are :

1        -3      -5      15      4       -12

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -2

End Point (b) = 2



Root(s) of equation :-

Root 1 = -2

Root 2 = -1

Root 3 = 1

Root 4 = 2
```

```
Check of Bisection method
 value of F( -2 ) = 0

Check of Bisection method
 value of F( -1 ) = 0

Check of Bisection method
 value of F( 1 ) = 0

Check of Bisection method
 value of F( 2 ) = 0


                                                END OF PROGRAM
```

Case 4 :

```
Microsoft Visual Studio Debug Console

                    This Program for calculating the root of any Polynomial equation .
                     F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  1x^6 + 0x^5 -14x^4 + 0x^3 + 49x^2 + 0x^1 -36


The Polynomial equation degree = 6

Cofficient are :

1       0       -14     0       49      0       -36

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -2

End Point (b) = 2



Root(s) of equation :-

Root 1 = -2

Root 2 = -1

Root 3 = 1

Root 4 = 2
```

```
Check of Bisection method
 value of F( -2 ) = 0

Check of Bisection method
 value of F( -1 ) = 0

Check of Bisection method
 value of F( 1 ) = 0

Check of Bisection method
 value of F( 2 ) = 0


                                        END OF PROGRAM
```

Case 5 :

```
                        This Program for calculating the root of any Polynomial equation .
                         F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  1x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 -5x^2 + 0x^1 + 2


The Polynomial equation degree = 7

Cofficient are :

1       0       0       0       0       -5      0       2

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -5

End Point (b) = 5


Root(s) of equation :-

Root 1 = -0.626441

Root 2 = 0.639321

Root 3 = 1.30819
```

```
Check of Bisection method
 value of F( -0.626441 ) = 0

Check of Bisection method
 value of F( 0.639321 ) = 0

Check of Bisection method
 value of F( 1.30819 ) = 0


                                        END OF PROGRAM
```

Case 6 :

Microsoft Visual Studio Debug Console

```
                        This Program for calculating the root of any Polynomial equation .
                         F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  -1x^8 + 0x^7 + 2x^6 + 6x^5 + 0x^4 + 0x^3 -4x^2 + 6x^1 + 10


The Polynomial equation degree = 8

Cofficient are :

-1       0       2       6       0       0       -4      6       10

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -5

End Point (b) = 5



Root(s) of equation :-

Root 1 = -0.843307

Root 2 = 2.18648

Check of Bisection method
 value of F( -0.843307 ) = 0

Check of Bisection method
 value of F( 2.18648 ) = 0


                                    END OF PROGRAM
```

Case 7 :

```
Microsoft Visual Studio Debug Console

                    This Program for calculating the root of any Polynomial equation .
                     F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  -1x^9 + 0x^8 + 0x^7 + 4x^6 + 0x^5 + 2x^4 + 0x^3 -2x^2 + 0x^1 -


The Polynomial equation degree = 9

Cofficient are :

-1      0       0       4       0       2       0       -2      0       -2

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -5

End Point (b) = 5



Root(s) of equation :-

Root 1 = -0.889529

Root 2 = 0.941278

Root 3 = 1.63409
```

```
Check of Bisection method
 value of F( -0.889529 ) = 0

Check of Bisection method
 value of F( 0.941278 ) = 0

Check of Bisection method
 value of F( 1.63409 ) = 0


                                        END OF PROGRAM
```

Case 8 :

Microsoft Visual Studio Debug Console

```
                    This Program for calculating the root of any Polynomial equation .
                     F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  -1x^10 + 0x^9 + 0x^8 + 0x^7 + 4x^6 + 12x^5 + 2x^4 + 7x^3 -2x^2 + 0x^1 -10


The Polynomial equation degree = 10

Cofficient are :

-1      0       0       0       4       12      2       7       -2      0       -10

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -5

End Point (b) = 5



Root(s) of equation :-

Root 1 = 0.840327

Root 2 = 1.85142

Check of Bisection method
 value of F( 0.840327 ) = 0

Check of Bisection method
 value of F( 1.85142 ) = 0


                                    END OF PROGRAM
```

Case 9 :

```
Microsoft Visual Studio Debug Console

                    This Program for calculating the root of any Polynomial equation .
                     F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  -1x^11 + 0x^10 + 0x^9 + 6x^8 + 0x^7 + 4x^6 + 0x^5 -2x^4 + 0x^3 + 2x^2 + 0x^1 + 36


The Polynomial equation degree = 11

Cofficient are :

-1      0      0      6      0      4      0      -2      0      2      0      36

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -5

End Point (b) = 5



Root(s) of equation :-

Root 1 = 1.92739

Check of Bisection method
 value of F( 1.92739 ) = 0


                                    END OF PROGRAM
```

Case 10 :

```
Microsoft Visual Studio Debug Console

                    This Program for calculating the root of any Polynomial equation .
                     F( x ) = anx^n + an-1x^n-1 + an-2x^n-2 + ... + a1x + a0 = 0

PRECAUTIONS TO AVOID ERRORS :-
 1)start your function with space
 2)Minus Sign must not have a blank with coefficient i.e. -7 not - 7
3) All Letters must be lowercase
 4) ALL terms of the Polynomial must be entered even zero coefficient one
 EXAMPLE :  (space)4x^4 -3x^3 + 0x^2 -1x^1 + 0

Enter the Polynomial Equation  F(X) =  -1x^12 + 0x^11 + 8x^10 + 0x^9 + 6x^8 + 0x^7 -4x^6 + 0x^5 + 4x^4 + 0x^3 + 1x^2 + 0x^1 + 48


The Polynomial equation degree = 12

Cofficient are :

-1      0      8      0      6      0      -4      0      4      0      1      0      48

Enter the domain :

PRECAUTIONS TO AVOID ERRORS :-
 1)Enter the start point then end point (smaller number then larger number)


Start Point (a) = -5

End Point (b) = 5



Root(s) of equation :-

Root 1 = -2.9407

Root 2 = 2.9407

Check of Bisection method
 value of F( -2.9407 ) = 0

Check of Bisection method
 value of F( 2.9407 ) = 0


                                    END OF PROGRAM
```