**CSC 150 Program 2 Name: Brandon Amundson**

**Input & Validation (total 35 points) Scores**

*Notes: If a prompt is unclear but program inputs work correctly, award half credit.*

Displays clear opening program message \_\_\_4\_\_ (4)

Displays appropriate prompt for degree of equation \_\_\_4\_\_ (4)

Displays appropriate prompts for equation coefficients \_\_\_11\_\_ (11)

Displays appropriate prompt for equation constant \_\_\_\_4\_ (4)

For error test case, displays error message and stops program \_\_\_\_4\_ (8)

Results displayed clearly \_\_\_\_4\_ (4)

**Program Correctness (total 40 points)**

*Notes: Roots are correct if they are within ±0.02 of the sample output.*

Displays all correct roots 40 points

**Documentation and Coding Standards (total 25 points)**

Points from Doxygen scoresheet \* 0.25 \_\_\_21\_\_ (25)

TOTAL PROGRAM GRADE: \_\_\_92\_\_ (100)

Documentation grade: (Points are all or nothing unless otherwise stated)

Program Header

Main Page \_\_5\_\_\_ (5)

Course Information

Author \_\_5\_\_\_ (5)

Date Due \_\_5\_\_\_ (5)

Professor \_\_5\_\_\_ (5)

Course \_\_5\_\_\_ (5)

Location \_\_\_5\_\_ (5)

Program Information (0, 5, 10, 15 will be assigned)

Details \_\_\_5\_\_ 15

Compiling

Compile & usage \_\_5\_\_\_ 5

Todo, Bugs, and Modifications

Bugs

Todo

Timeline \_\_10\_\_\_ 10

**Total \_\_50\_\_\_ (60)**

Function Header (all functions must have the following, if one function is missing, no points will be assigned.

Author \_\_5\_\_\_ 5

Description \_\_5\_\_\_ 5

Params (when needed) \_\_5\_\_\_ 5

Returns (when needed) \_\_5\_\_\_ 5

**Total \_\_\_20\_\_ (20)**

Others

Meaningful variable names \_\_5\_\_\_ (5)

Whitespace around functions,

blocks of code and operators \_\_5\_\_\_ (5)

Global constants document

(if they have a variable do not give credit) \_\_5\_\_\_ (5)

Comments, must be adequate to cover the algorithm

If the code was removed \_\_0\_\_\_ (5)

**Total \_\_15\_\_\_ (20)**

Other items not mentioned above \_\_\_\_\_ ( )

Files named correctly ( -5 )

**Documentation Total \_\_\_85\_\_ ( 100 )**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

\* @file

\* @brief Demonstrates the use of arrays and passing through functions

\*

\* @mainpage Program 2 - Newtons Method Expanded

\*

\* @section course\_section Course Information

\*

\* @authors Brandon Amundson

\*

\* @date November 07, 2014

\*

\* @par Instructor:

\* Dr. Logar

\*

\* @par Course:

\* CSC 150 - Section 1 - 10:00 am

\*

\* @par Location:

\* McLaury - Room 306

\*

\* @section program\_section Program Information

\*

\* @details This program will compute the zero

\* of any function as input by user using Newton's

\* Method

\*

\*

\* @section compile\_section Compiling and Usage

\*

\* @par Compiling Instructions:

\* None

\*

\* @par Usage:

@verbatim

c:\> prog2.exe

d:\> c:\bin\prog2.exe

@endverbatim

\*

\* @section todo\_bugs\_modification\_section Todo, Bugs, and Modifications

\*

\* @bug

\*

\* @todo Write the code to compute the root

\*

\* @par Modifications and Development Timeline:

@verbatim

Date Modification

------------ ---------------------------------------------------------------

Oct 24, 2014 Started writing the program

Oct 29, 2014 Wrote get\_function\_coefficients, main, compute\_derivative, and f

Nov 05, 2014 added doxygen coding and wrote f\_prime, and started to write

find\_root

Nov 06, 2014 added error check into get\_function\_coefficients, finished writing

main funtion

Nov 07, 2014 finished rest of program

@endverbatim

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <cmath>

#include <iomanip>

#include "constants.h"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

\* @author Brandon Amundson

\*

\* @par Description:

\* Function that gets the input from the user, and returns

\* a status value to indicate whether there has been

\* an input error.

\*

\* @param[in] coeff[] - the array that holds the coefficients.

\* @param[in] degree - the degree of the polynomial.

\*

\* @returns 0 - error

\* @returns 1 - okay coeff

\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int get\_function\_coefficients(double coeff[], int &degree)

{

int i;

cout << "Enter the degree of the polynomial: ";

cin >>degree;

if (degree <= 0)

{

cout << "Error!";

return (0);

}

cout << "Enter the coefficients for the equation." << endl;

for (i=degree; i >=0; i--)

{

if (i == 0)

{

cout << "Enter the constant term: ";

cin >> coeff[i];

}

else

{

cout << "Enter coefficient for term x^" << i << ": ";

cin >>coeff[i];

}

}

return (1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

\* @author Brandon Amundson

\*

\* @par Description:

\* This is the main function of the program. It will call the other functions

\* and output and will then exit with code 0

\*

\*

\* @returns 0 - error

\* @returns 1 - okay coeff

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main ()

{

double coeff[MAX\_DEGREE] = {0};

int degree;

double derivative[MAX\_DEGREE] = {0};

double x1 = 0;

double x2 = 0;

int status;

double temp2;

cout << "This program finds the roots of polynomial functions." << endl;

status = get\_function\_coefficients(coeff, degree);

if (status == 0)

{

return (1);

}

compute\_derivative(coeff, derivative, degree);

temp2 = find\_root(x1, x2, coeff, derivative, degree);

return (0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

\* @author Brandon Amundson

\*

\* @par Description:

\* Function that takes the original equation

\* and computes the equation's derivative

\*

\* @param[in] coeff[] - the coefficients of polynomial

\* @param[in] derivative[] - coefficients of derivative of polynomial.

\* @param[in] degree - degree of function

\*

\* @returns derivative[] - coefficients of derivative

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void compute\_derivative(double coeff[], double derivative[], int degree)

{

int i;

for (i = degree; i > 0; i--)

{

derivative[i-1] = coeff[i] \* i;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

\* @author Brandon Amundson

\*

\* @par Description:

\* Function that computes and returns the value of f(x)

\*

\* @param[in] x - the number being input into function.

\* @param[in] coeff[] - the coefficients of polynomial

\* @param[in] degree - degree of function

\*

\* @returns result - of computed derivative

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double f( double x, double coeff[], int degree)

{

int i;

double result = 0;

for (i = degree; i >= 0; i--)

result += (coeff[i] \* pow(x, i));

return (result);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

\* @author Brandon Amundson

\*

\* @par Description:

\* Function that computes and returns the value of f'(x)

\*

\* @param[in] x - the number being input into function.

\* @param[in] derivative[] - the coefficients of polynomial

\* @param[in] degree - degree of function

\*

\* @returns result - of computed derivative

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double f\_prime (double x, double derivative[], int degree)

{

int i = 0;

double result = 0;

for (i = (degree - 1); i >= 0; i--)

{

result += (derivative[i] \* pow(x, i));

}

return (result);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

\* @author Brandon Amundson

\*

\* @par Description:

\* Function that takes the original equation, the derivative, the

\* start and end values, and uses Newton's Method to find

\* the root within the interval.

\*

\* @param[in] x1 -

\* @param[in] x2 -

\* @param[in] coeff[] - the coefficients of polynomial.

\* @param[in] derivative[] - coefficients of derivative of polynomial.

\* @param[in] degree - degree of polynomial.

\*

\* @returns 1

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double find\_root(double x1, double x2, double coeff[], double derivative[], int degree)

{

double x = 0;

int iterations = 0;

int iter = 0;

double root[MAX\_DEGREE];

double temp1;

double temp2;

double temp3 = 1;

double temp4;

double temp5;

x1 = START;

x2 = x1 + STEP;

do

{

temp1 = f(x1, coeff, degree);

temp2 = f(x2, coeff, degree);

if (temp1 \* temp2 <= 0)

{

x = (x1 + x2) / 2.0;

temp1 = f(x, coeff, degree);

while (abs(temp3) > TOLERANCE && iterations <= N\_ITER)

{

temp3 = f(x, coeff, degree);

temp4 = f\_prime(x, derivative, degree);

temp5 = temp3 - (x \* temp4);

x = (-1 \* temp5)/temp4;

iterations++;

}

root[iter] = x;

iter++;

}

else if ((temp1 \* temp2) == 0)

{

root[iterations] = f(x2, coeff, degree);

iterations++;

}

x1 = x2;

x2 = x1 + STEP;

}

while (x1 <= FINISH);

if (iter == 1)

cout << "there is " << iter << " root in the interval (-3, 4): ";

else

cout << "There are " << iter << " roots in the interval (-3, 4): ";

for ( int i = 0; i <iter; i++)

cout << root[i] << " ";

cout << endl;

return (0);

}

//x1 = START;

//x2 = x1 + STEP;

//do

//{

//if (x1 \* x2 >=0)

//{

//do

//{

//x2 = x1 + STEP;

//x = (x1 + x2)/2.0;

//x1 = f (x, coeff, degree); //x1 becomes 1 here because i return 1, how do i return arrays?

//x2 = f\_prime (x, derivative, degree); //x2 also becomes 1 here see above?

//result = x1 - (x \* x2);

//b = -result/x2;

//error = x1; //error is always going to be one at this point causes infinite loop

//}

//while (error >= TOLERANCE);

//}

//if (x = 0)

//{

//cout << "The root of the function exists at " << x;

//iter++;

//}

//}

//while (x1 <= FINISH);

//return iter++;

//for (x1 = START; x1 < FINISH; x1 += STEP)

//{

// x2 = x1 + STEP;

// if (x1 \* x2 <= 0)

// {

// x = (x1 + x2)/2.0;

// f(x, coeff, degree);

// f\_prime(x, derivative, degree);