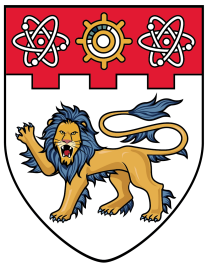

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

# MA4830: Realtime Software for Mechatronic Systems

## Minor Programming Assignment

**Dihan Jannatan Mutaalim (U2020091J)**
**Soh Zheng Da (U2022972E)**
**Steven Edbert Winata (U2020152E)**
**Yang Xunsheng (U2021913E)**

Lecturer: Prof. Gerald Seet Gim Lee
Class: MA3

**SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING**
**NANYANG TECHNOLOGICAL UNIVERSITY**

**AY SEM 2 2022/2023**

# Table of Contents

# Chapter 1 Introduction

## 1.1 Background

A quadratic equation is a type of polynomial equation of degree two which can be written in the standard form:

$$ax^2 + bx + c = 0$$

where a, b, and c are constants, and x is the variable. The coefficient 'a' is the leading coefficient, and must not be zero.

The quadratic equation can be solved to find the values of x that satisfy the equation using methods such as factoring, completing the square, or using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where $\sqrt{}$ denotes the square root symbol, and ± indicates that there are two possible solutions to the equation, one with the plus sign and one with the minus sign.

The solutions of a quadratic equation can be real or complex numbers, depending on the values of the coefficients a, b, and c.

## 1.2 Objective

The goal is to develop a software program that can solve quadratic equations. The program will be written in the C programming language and designed to run on both QNX and Windows operating systems. This means that the program will need to be compatible with the specific requirements and constraints of both operating systems.

The program will take three coefficients (a, b, and c) as input and use them to solve the quadratic equation in the form $ax^2 + bx + c = 0$. The program will then output the solutions (real, complex, or imaginary) of the equation.

To ensure that the program is user-friendly, it will provide clear prompts for the user to input the coefficients and display the solutions in a clear and concise manner. Additionally, the program will handle any potential errors or exceptions that may occur during input or computation, such as invalid inputs or division by zero.

Overall, the objective is to create a reliable and efficient quadratic equation solver that can be used by individuals or businesses operating on both QNX and Windows systems.

# Chapter 2 Problem Analysis

## 2.1 Problem-Solving Approach

Before writing the C program, the group needs to understand how to solve the quadratic equation.

There are several methods for solving a quadratic equation, including factoring, completing the square, and using the quadratic formula[1]. Here's a brief overview of each method:

**Factoring**: If the quadratic equation can be factored into two binomials, then the roots can be found by setting each factor equal to zero and solving for x. For example, if we have the equation $x^2 + 5x + 6 = 0$, we can factor it as (x + 2)(x + 3) = 0. Setting each factor equal to zero, we get x = -2 and x = -3, which are the roots of the equation.

**Completing the square**: This method involves manipulating the quadratic equation to express it in a form that can be easily solved by taking the square root of both sides. For example, if we have the equation $x^2 + 6x + 5 = 0$, we can complete the square as follows:

$$\begin{gathered} x^2+6x+5=0 \\ (x+3)^2-4=0 \\ (x+3)^2=4 \\ x+3=\pm 2 \\ x=-3\pm 2 \end{gathered}$$

So the roots of the equation are x = -1 and x = -5.

**Quadratic formula**: This technique, which covers almost all cases, was first obtained by Simon Stevin in 1594. In 1637 René Descartes published La Géométrie containing special cases of the quadratic formula in the form we know today[2]. The quadratic formula is a formula that gives the roots of a quadratic equation directly in terms of the coefficients a, b, and c. The formula is:

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

Using the formula, we can find the roots of any quadratic equation, even if it cannot be factored or completed by the square. For example, if we have the equation $2x^2 - 5x + 3 = 0$, we can use the quadratic formula to find the roots:

$$x = \frac{-(-5) \pm \sqrt{(-5)^2 - 4(2)(3)}}{2 \cdot 2} \qquad \longrightarrow x = \frac{5 \pm \sqrt{25-24}}{4}$$

So the roots of the equation are x = 1 and x = $\frac{3}{2}$.

These are the most common methods for solving a quadratic equation, and one can use any method that is most convenient for the given problem.

The group research on the required C libraries that will be needed for the program. Next, the function and variable needed to solve for each condition then the printing and looping of the program.

## 2.2 Program Highlight

Here are some keypoints on highlights and novelty of our program on solving quadratic equations.

1. The program is able to solve the quadratic equation and give all 3 conditions which are one real root, two real roots and two complex roots. This was achieved by evaluating the discriminant condition.

2. One highlight of the program will be the prompt of continuous solving of quadratic roots if the user wanted. This was achieved by asking the user at the end of the program.

3. The program also has a custom square roots function using the Newton-Raphson method, which can calculate the approximate square root of a number. Hence the program need not import the math.h library.

4. The function sqrt() in math.h library is not able to handle the negative number, however, our program can produce the imaginary output by evaluating the discriminant conditions and calculating the real and imaginary parts.

## 2.3 Program Limitation

However, there are some limitations to the program which were highlighted below:

1. Not able to handle complex coefficients: The program assumes that the user inputs real-valued coefficients, and cannot handle equations with complex coefficients.

2. Limited error checking: While the program does perform some error checking, it is limited to checking that the user inputs a number for each coefficient and that coefficient a is not equal to zero. It does not perform more advanced error checking, such as checking for NaN or infinite values.

3. Limited precision: The program uses double precision floating point numbers, which have limited precision. This can result in inaccurate solutions for some equations.

4. No support for symbolic solutions: The program only provides numerical solutions to the quadratic equation and does not support symbolic solutions. This means that the program cannot show the steps taken to arrive at the solution, which can be helpful in some cases.

5. Input such as a combination of numeric and alphabets may result in errors in the output. For instance, input such as ‘1a’ will be read as input of coefficient b = 1 and c = invalid input. However, the program will still run and read the equation as b = 1 and c = 0, resulting in an output that the user may not expect it to be, i.e. wrong output

6. If the program is to be run on the QNX operating system, the program may fail due to semantics of variable declarations. In modern C-language compilers, variables need not be declared first before using them, while in the QNX operating system, the variable declaration needs to be written first.

7. Input in case where the value is larger than the double type of variables will result in overflow case. Hence the input should be within the range of double type of input which is 1.7E-308 to 1.7E+308.

# Chapter 3 Solution

## 3.1 Flowchart of Program

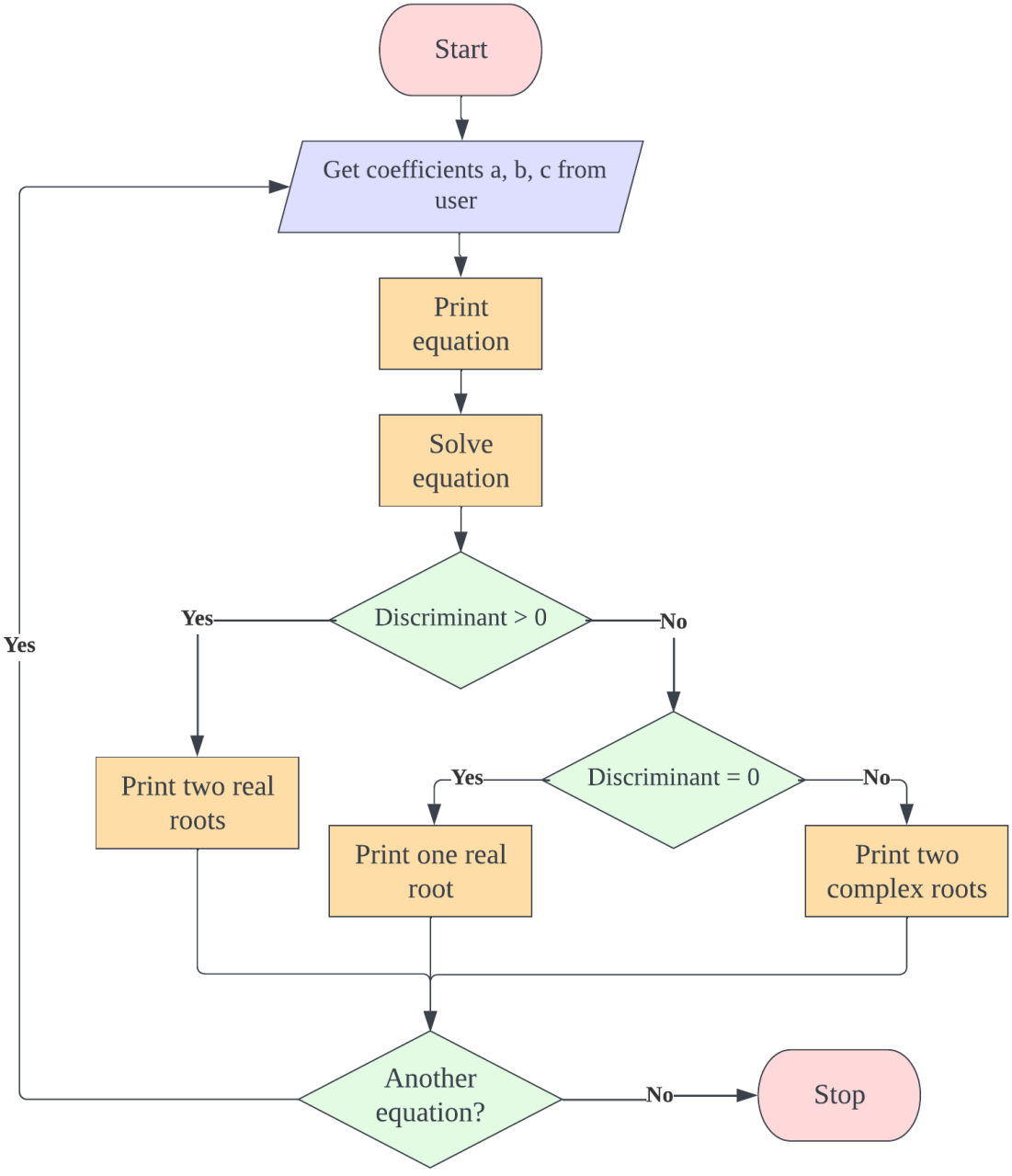


Figure 3-1: Flowchart of program

## 3.2 Program Listing

To see the whole program, see Appendix B. Please note that section Appendix A is the original function which contains only the main task without any miscellaneous print function.

```
// Main function to execute the program and prompt the user input
int main() {
    char Flag = 'Y';
    while(Flag=='Y' || Flag=='y'){
    system("cls");
    print_equal_signs(total_print_space);
    printf("|"); print_with_indent_centered(total_print_space-2, "Quadratic Equation Solver"); printf("|\n"); // account for the two | at the start
       and end
    printf("|"); print_with_indent_centered(total_print_space-2, "ax^2 + bx + c = 0"); printf("|\n"); // account for the two | at the start and end
    print_equal_signs(total_print_space);
    double a = get_coefficient('a');
    double b = get_coefficient('b');
    double c = get_coefficient('c');
    char concatentated_string[30] = "";
    snprintf(concatentated_string, 30, "%.2lfx^2 + %.2lfx + %.2lf = 0", a, b, c);
    print_equal_signs(total_print_space);
    printf("|"); print_with_indent_centered(total_print_space-2, "The equation of"); printf("|\n");
    printf("|"); print_with_indent_centered(total_print_space-2, concatentated_string); printf("|\n");
    print_equal_signs(total_print_space);
    printf("|");
    print_coefficient_indent(32, 'a', a, 2);
    print_coefficient_indent(34, 'b', b, 2);
    print_coefficient_indent(32, 'c', c, 2);
    printf("|\n");
    print_equal_signs(total_print_space);
    solve_quadratic(a, b, c);
    printf("Do you want to solve another equation?\n(Input 'Y' or 'y' to continue or type any key to end the program): ");
    scanf(" %c",&Flag);
    }
    printf("Program End");
    return 0;
}

```

## 3.3 Program Output

The following output display and the error handling are presented below for each case of user input.

1. Coefficient ‘a’, ‘b’, and ‘c’ inputs are in numerical value (real number)

```
================================================================================================================
||                                      Quadratic Equation Solver                                               ||
||                                          ax^2 + bx + c = 0                                                   ||
================================================================================================================
Enter coefficient a: 1
Enter coefficient b: 2
Enter coefficient c: -1
================================================================================================================
||                                            The equation of                                                   ||
||                                       1.00x^2 + 2.00x + -1.00 = 0                                            ||
================================================================================================================
||          a = 1.00            ||           b = 2.00            ||          c = -1.0            ||
================================================================================================================
||                                          TWO REAL roots                                                      ||
================================================================================================================
||                                         Root 1 : 0.414214                                                    ||
||                                        Root 2 : -2.414214                                                    ||
================================================================================================================
Do you want to solve another equation?
(Input 'Y' or 'y' to continue or type any key to end the program): x
Program End
```

Figure 3-2: Output is two real roots

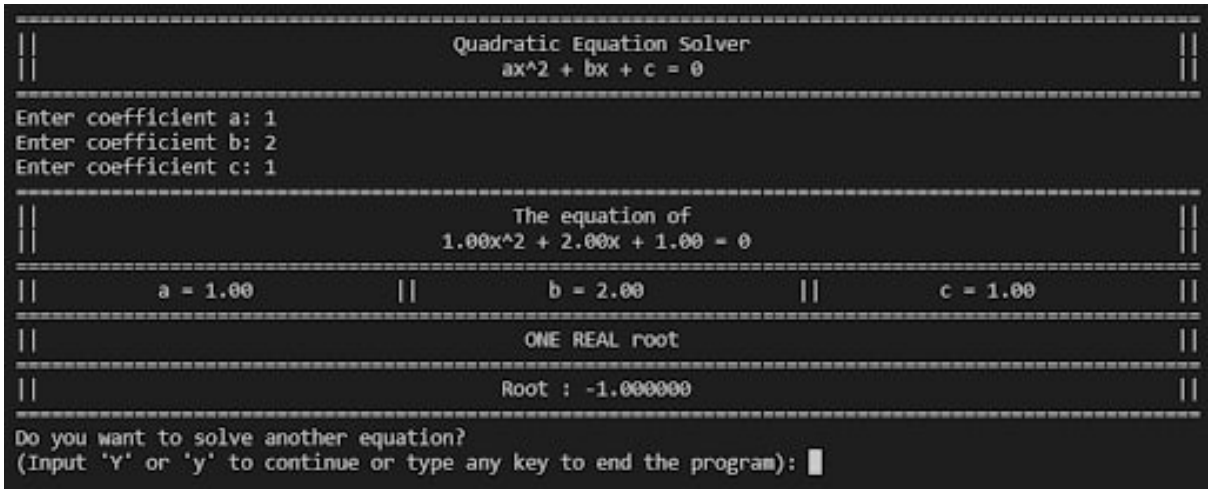


Figure 3-3: Output is one real root

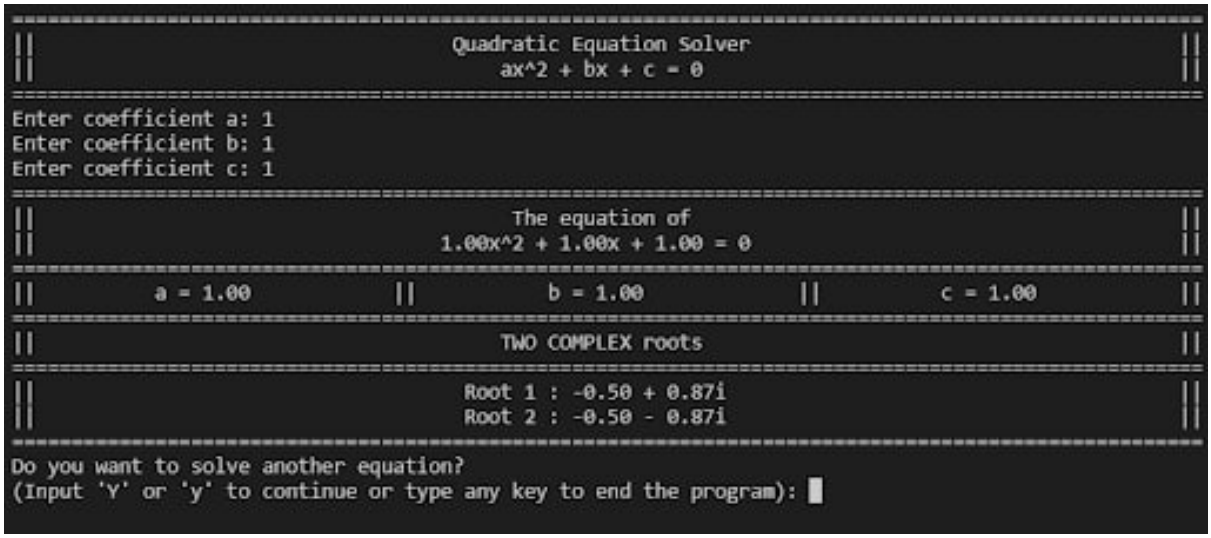


Figure 3-4: Output is two complex roots

2. One of the coefficient input is not in numerical value

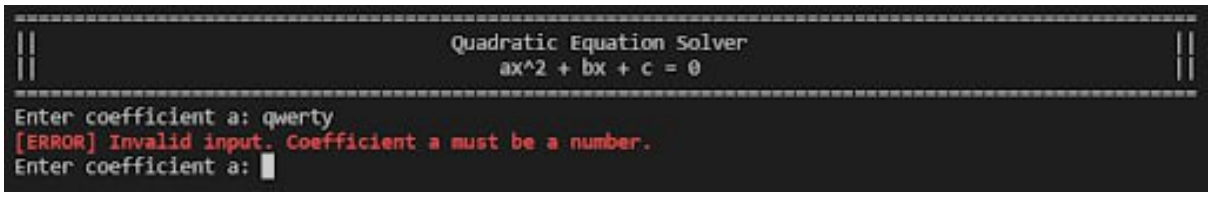


Figure 3-5: One of the coefficient input is not in numerical value

3. Coefficient ‘a’ is 0

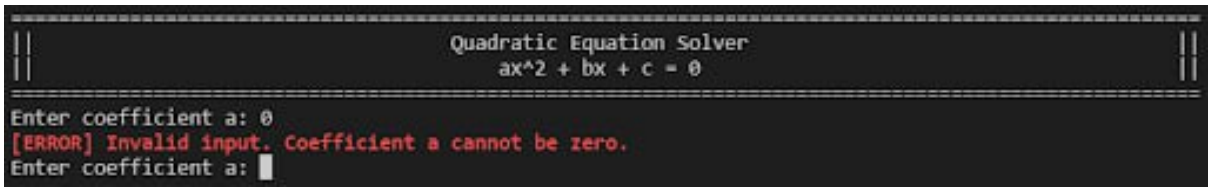


Figure 3-6: Coefficient ‘a’ is 0

# Chapter 4 Conclusion

In conclusion, the program developed can solve quadratic equations using the following methods: factoring, completing the square, and the quadratic formula. It is capable of handling different scenarios such as equations with one real root, two real roots, or two complex roots. The program also has a custom square root function that can handle negative numbers and produce imaginary output.

However, the program has some limitations, such as the inability to handle equations with complex coefficients, limited error checking, limited precision, and no support for symbolic solutions. Despite these limitations, the program can still be useful for solving quadratic equations in various applications.

Further improvements can be made to the program, such as adding more advanced error checking and improving the precision of the numerical solutions. Nonetheless, the current program provides a solid foundation for solving quadratic equations and can be a useful tool for everyone.

# References


[1] C. P. McKeague, *Intermediate algebra with trigonometry*. Orlando, FL: Academic Press., 1983, pp. 232–235.

[2] L. Rogers and S. Pope, "A brief history of quadratic equations for mathematics educators," *Proceedings of the British Society for Research into Learning Mathematics*, vol. 35, no. 3, pp. 90–95, 2015.

# Appendix A Function Code

## A.1 coefficient

```
// Function to get a coefficient from the user with error checking
double coefficient;
    while (1) {
        printf("Enter coefficient %c: ", name);
        if (scanf("%lf", &coefficient) != 1) {
            red_bolded();
            printf("[ERROR] Invalid input. Coefficient %c must be a number.\n", name);
            reset();
            while (getchar() != '\n'); // clear input buffer
        } else if (coefficient == 0 && name == 'a') {
            red_bolded();
            printf("[ERROR] Invalid input. Coefficient a cannot be zero.\n");
            reset();
        } else {
            return coefficient;
        }
    }
}
```

## A.2 get_discriminant

```
// Function to calculate the discriminant and return its value
double get_discriminant(double a, double b, double c) {
    return b * b - 4 * a * c;
}
```

## A.3 my_sqrt

```
// Custom square root function using the Newton-Raphson method
double my_sqrt(double x) {
    if (x == 0.0) {
        return 0.0;
    }
    double prev = 0.0;
    double next = 1.0;
    while (prev != next) {
        prev = next;
        next = (prev + x / prev) / 2;
    }
    return next;
}
```

## A.4 solve_quadratic

```
// Function to solve the quadratic equation and print the results
void solve_quadratic(double a, double b, double c) {
    double discriminant = get_discriminant(a, b, c);
    if (discriminant > 0) {
        double root1 = (-b + my_sqrt(discriminant)) / (2 * a);
        double root2 = (-b - my_sqrt(discriminant)) / (2 * a);
        printf("has two real roots: %.2lf and %.2lf\n", root1, root2);
    } else if (discriminant == 0) {
        double root1 = -b / (2 * a);
        printf("has one real root: %.2lf\n", root1);
    } else {
        double realPart = -b / (2 * a);
        double imagPart = my_sqrt(-discriminant) / (2 * a);
        printf("has two complex roots: %.2lf + %.2lfi and %.2lf - %.2lfi\n", realPart,
    imagPart, realPart, imagPart);
    }
}
```

# Appendix B Full Code

```c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int total_print_space = 100;

void print_with_indent_centered(int total_print_space, char * string);
void print_equal_signs(int total_print_space);
void my_strcat(char * destination, char * source);

void red_bolded () {
  printf("\033[1;31m");
}

void reset () {
  printf("\033[0m");
}

// Function to get a coefficient from the user with error checking
double get_coefficient(char name) {
    double coefficient;
    while (1) {
        printf("Enter coefficient %c: ", name);
        if (scanf("%lf", &coefficient) != 1) {
            red_bolded();
            printf("[ERROR] Invalid input. Coefficient %c must be a number.\n", name);
            reset();
            while (getchar() != '\n'); // clear input buffer
        } else if (coefficient == 0 && name == 'a') {
            red_bolded();
            printf("[ERROR] Invalid input. Coefficient a cannot be zero.\n");
            reset();
        } else {
            return coefficient;
        }
    }
}

// Function to calculate the discriminant and return its value
double get_discriminant(double a, double b, double c) {
    return b * b - 4 * a * c;
}

// Custom square root function using the Newton-Raphson method
double my_sqrt(double x) {
    if (x == 0.0) {
        return 0.0;
    }
    double prev = 0.0;
    double next = 1.0;
    while (prev != next) {
        prev = next;
        next = (prev + x / prev) / 2;
    }
```

```
55      return next;
56  }
57
58  // Function to solve the quadratic equation and print the results
59  void solve_quadratic(double a, double b, double c) {
60      double discriminant = get_discriminant(a, b, c);
61      char buffer_1[46]= "Root 1 : ";
62      char buffer_2[46]= "Root 2 : ";
63      char buffer_1root[46]= "Root : ";
64      char buffer[38];
65      char buffer2[38];
66      char buffer_1root_value[39];
67
68      if (discriminant > 0) {
69          double root1 = (-b + my_sqrt(discriminant)) / (2 * a);
70          double root2 = (-b - my_sqrt(discriminant)) / (2 * a);
71          snprintf(buffer, 38, "%f", root1);
72          my_strcat(buffer_1, buffer);
73          snprintf(buffer, 38, "%f", root2);
74          my_strcat(buffer_2, buffer);
75          printf("|"); print_with_indent_centered(total_print_space-2, "TWO REAL roots");
      printf("|\n");
76          print_equal_signs(total_print_space);
77
78          printf("|"); print_with_indent_centered(total_print_space-2, buffer_1); printf("|\
      n");
79          printf("|"); print_with_indent_centered(total_print_space-2, buffer_2); printf("|\
      n");
80
81          print_equal_signs(total_print_space);
82      } else if (discriminant == 0) {
83          double root1 = -b / (2 * a);
84          snprintf(buffer_1root_value, 39, "%f", root1);
85          my_strcat(buffer_1root, buffer_1root_value);
86
87          printf("|"); print_with_indent_centered(total_print_space-2, "ONE REAL root");
      printf("|\n");
88          print_equal_signs(total_print_space);
89
90          printf("|"); print_with_indent_centered(total_print_space-2, buffer_1root); printf
      ("|\n");
91          print_equal_signs(total_print_space);
92      } else {
93          double realPart = -b / (2 * a);
94          double imagPart = my_sqrt(-discriminant) / (2 * a);
95          snprintf(buffer, 38, "%.2lf + %.2lfi", realPart, imagPart);
96          my_strcat(buffer_1, buffer);
97          snprintf(buffer2, 38, "%.2lf - %.2lfi", realPart, imagPart);
98          my_strcat(buffer_2, buffer2);
99
100         printf("|"); print_with_indent_centered(total_print_space-2, "TWO COMPLEX roots");
       printf("|\n");
101         print_equal_signs(total_print_space);
102
103         printf("|"); print_with_indent_centered(total_print_space-2, buffer_1); printf("|\
      n");
104         printf("|"); print_with_indent_centered(total_print_space-2, buffer_2); printf("|\
      n");
105         print_equal_signs(total_print_space);
106     }
```

```
107 }
108
109 int my_strlen(char * string) {
110     int length = 0;
111     while (string[length] != '\0') {
112         length++;
113     }
114     return length;
115 }
116
117 void print_equal_signs(int n) {
118     for (int i = 0; i < n; i++) {
119         printf("=");
120     }
121     printf("\n");
122 }
123
124 void my_strcat(char * dest, char * src) {
125     int dest_length = my_strlen(dest);
126     int src_length = my_strlen(src);
127     for (int i = 0; i < src_length; i++) {
128         dest[dest_length + i] = src[i];
129     }
130     dest[dest_length + src_length] = '\0';
131 }
132
133 int digit_counter(int number) {
134     int count = 0;
135     while (number != 0) {
136         number /= 10;
137         count++;
138     }
139     return count;
140 }
141
142 void print_with_indent_centered(int total_print_space, char * string)
143 {
144     int string_length = 0;
145     int indent1 = 0, indent2 = 0;
146     string_length = my_strlen(string);
147     total_print_space -= 2; // account for the two | at the start and end
148     if (string_length%2 == 0) {
149         indent1 = (total_print_space - string_length) / 2 - 1;
150         indent2 = (total_print_space - string_length) / 2 + 1;
151     }
152     else {
153         indent1 = (total_print_space - string_length) / 2;
154         indent2 = indent1 + 1;
155     }
156     printf("|%*s%s%*s|", indent1, "", string, indent2, "");
157 }
158 void print_coefficient_indent(int total_print_space, char coefficient_name, double
    coefficient, int decimal_places){
159     float coefficient_float = (float)coefficient;
160     char buffer[26];
161     int string_length = 0;
162     int indent1 = 0, indent2 = 0;
163     snprintf(buffer, digit_counter((int)coefficient) + decimal_places + 2, "%lf",
    coefficient);
```

```
164     string_length = my_strlen(buffer) + 4 + 2; // 4 for the coefficient name, 2 for | at
        start and end
165     if (string_length%2 == 0) {
166         indent1 = (total_print_space - string_length) / 2 - 1;
167         indent2 = (total_print_space - string_length) / 2 + 1;
168     }
169     else {
170         indent1 = (total_print_space - string_length) / 2;
171         indent2 = indent1 + 1;
172     }
173     printf("|%*s%c = %s%*s|", indent1, "", coefficient_name, buffer, indent2, "");
174 }
175
176 int main() {
177     char Flag = 'Y';
178     while(Flag=='Y' || Flag=='y'){
179     system("cls");
180     // printf("============================\n");
181     print_equal_signs(total_print_space);
182     // printf(*print_with_indent_centered(total_print_space, "Quadratic Equation Solver"))
        ;
183     printf("|"); print_with_indent_centered(total_print_space-2, "Quadratic Equation
        Solver"); printf("|\n"); // account for the two | at the start and end
184     printf("|"); print_with_indent_centered(total_print_space-2, "ax^2 + bx + c = 0");
        printf("|\n"); // account for the two | at the start and end
185     print_equal_signs(total_print_space);
186     // printf("============================\n");
187
188
189     double a = get_coefficient('a');
190     double b = get_coefficient('b');
191     double c = get_coefficient('c');
192     char concatentated_string[30] = "";
193     snprintf(concatentated_string, 30, "%.2lfx^2 + %.2lfx + %.2lf = 0", a, b, c);
194
195     // printf("The equation of %.2lfx^2 + %.2lfx + %.2lf = 0 \n", a, b, c);
196     print_equal_signs(total_print_space);
197     printf("|"); print_with_indent_centered(total_print_space-2, "The equation of");
        printf("|\n");
198     printf("|"); print_with_indent_centered(total_print_space-2, concatentated_string);
        printf("|\n");
199     print_equal_signs(total_print_space);
200
201     printf("|");
202     print_coefficient_indent(32, 'a', a, 2);
203     print_coefficient_indent(34, 'b', b, 2);
204     print_coefficient_indent(32, 'c', c, 2);
205     printf("|\n");
206     print_equal_signs(total_print_space);
207
208     solve_quadratic(a, b, c);
209     printf("Do you want to solve another equation?\n(Input 'Y' or 'y' to continue or type
        any key to end the program): ");
210     scanf(" %c",&Flag);
211     }
212
213     printf("Program End");
214     return 0;
215 }
```