ECE 175 Computer Programming for Engineering Applications

Homework Assignment 3

Due Date: Tuesday February 6, 2018, 11:59 PM, via D2L Drop-box

Conventions: Name your C programs $hw\mathbf{x}p\mathbf{y}.c$ where x corresponds to the homework number, and y corresponds to the problem number. For example, the C program for homework 3, problem 1 should be named as $hw\mathbf{3}p\mathbf{1}.c$.

Write comments to your programs. Programs with no comments will receive PARTIAL credit. For each program that you turn in, at least the following information should be included at the beginning of your file as comments

- Author:
- Date created:
- Brief description of the program:
  - input(s):
  - output(s):
  - brief description or relationship between inputs and outputs

Submission Instructions: Use **Zylab** and the **"Assignments"** drop-box on D2L to submit your homework. Submit

- Pseudo code for each of the problems
- The corresponding .c files

1 League of Legends: Problem Description

In *League of Legends*, a player's Effective Health when defending against physical damage is given by $E = \frac{H\left(100+A^2\right)}{100}$ where H is health, A is armor and E is the effective health. Health costs 2 gold per unit, and armor costs 18 gold per unit. In order for the player to survive as long as possible it is important to maximize the player's Effective Health E.

1.1 Program Description

For a given amount of gold that the user wishes to spend, we can use a computer program to calculate the optimal amount of health H and armor A that will maximize the Effective Heath E. The two formulae that we will use are

$$E = \frac{H\left(100+A^2\right)}{100}$$

$$GoldToSpend = 2 \cdot H + 18 \cdot A$$

Note that A and H can only be purchased in whole units, i.e. it doesn't make sense to buy 1.5 health or 6.7 armor.

Write a C program that asks the player how much gold they wish to spend, then calculates the optimal amount of H and A that the user should purchase in order to maximize E.

Note: Loop structures must be used to solve the problem, otherwise 0 point for this problem.

Tips to solving this problem:

1. Loop across all possible values of A; The minimum amount of armor that can be purchased is $A = 0$ and the maximum amount is $A = \frac{GoldToSpend}{18}$
2. For a given amount of A, solve the $GoldToSpend$ equation for H
3. Calculate E using the current values of A and H
4. Keep track of the maximum amount of E seen during the loop. Also keep track of the corresponding A and H used to calculate the maximum E.
5. Report the results

1.2 Test your program

Sample Code Execution: Red text indicates information entered by the user

```
How much gold do you wish to spend? -45
You have no gold to spend.
Please enter in a valid amount of gold to spend.

How much gold do you wish to spend? 0
You have no gold to spend.
Please enter in a valid amount of gold to spend.

How much gold do you wish to spend? 120
For 120 gold you should buy 60 Health, and 0 Armor for an effective health score of 60.000
```

Sample Code Execution: Red text indicates information entered by the user

```
How much gold do you wish to spend? 151
For 150 gold you should buy 75 Health, and 0 Armor for an effective health score of 75.000
You should save 1 gold for the next round
```

Sample Code Execution: Red text indicates information entered by the user

```
How much gold do you wish to spend? 400
For 400 gold you should buy 92 Health, and 12 Armor for an effective health score of 224.480
```

Sample Code Execution: Red text indicates information entered by the user

```
How much gold do you wish to spend? 1000
For 1000 gold you should buy 176 Health, and 36 Armor for an effective health score of 2456.960
```

Sample Code Execution: Red text indicates information entered by the user

```
How much gold do you wish to spend? 10023
For 10022 gold you should buy 1672 Health, and 371 Armor for an effective health score of 2303029.520
You should save 1 gold for the next round
```

2 Polynomial Root Calculation: Problem Description

Engineers often need to calculate the roots of a given polynomial. For low order polynomials these computations can be done by hand. For higher order polynomials these calculations can be cumbersome, if not impossible, without the use of a computer. One computational method of finding real roots of a polynomial is the Newton-Raphson algorithm.

We wish to write a *C* program that uses the Newton-Raphson method for calculating the roots of a polynomial. Although this program will not be able to solve for complex roots, it will be able to calculate real roots; maybe later we can adjust the routine to include complex roots. Since it is possible that the polynomial roots are all complex, i.e. no real roots at all, it may be that the Newton-Raphson routine fails to converge.

2.1 Newton-Raphson Algorithm:

Write a program that prompts the user to input the coefficients $c_5, c_4, c_3, c_2, c_1, c_0$ of a 5_{th}-order polynomial.

The 5_{th} order polynomial has the form

$$y = c_5 \cdot x^5 + c_4 \cdot x^4 + c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x + c_0$$

We know that the first derivative of y with respect to x is

$$\frac{dy}{dx} = y' = 5 \cdot c_5 \cdot x^4 + 4 \cdot c_4 \cdot x^3 + 3 \cdot c_3 \cdot x^2 + 2 \cdot c_2 \cdot x + c_1$$

We can use this information to find the roots of the polynomial. The basic idea, in the Newton-Raphson method, is as follows:

(a) Given an initial guess x, and polynomial coefficients c, calculate y

(b) Check to see if y is close enough to zero, i.e. within some small tolerance close to zero.

  (i) If so then terminate. Algorithm has converged!

  (ii) If not then continue

(c) Use the current value of x to calculate y'

(d) Create a new guess for x using the update formula $x = x - \frac{y}{y'}$

(e) Increment a counter to count the number of algorithm iterations

(f) Check to see if the number of iterations has exceeded a predetermined count limit (say 500)

  (i) If so then terminate. Algorithm has failed!

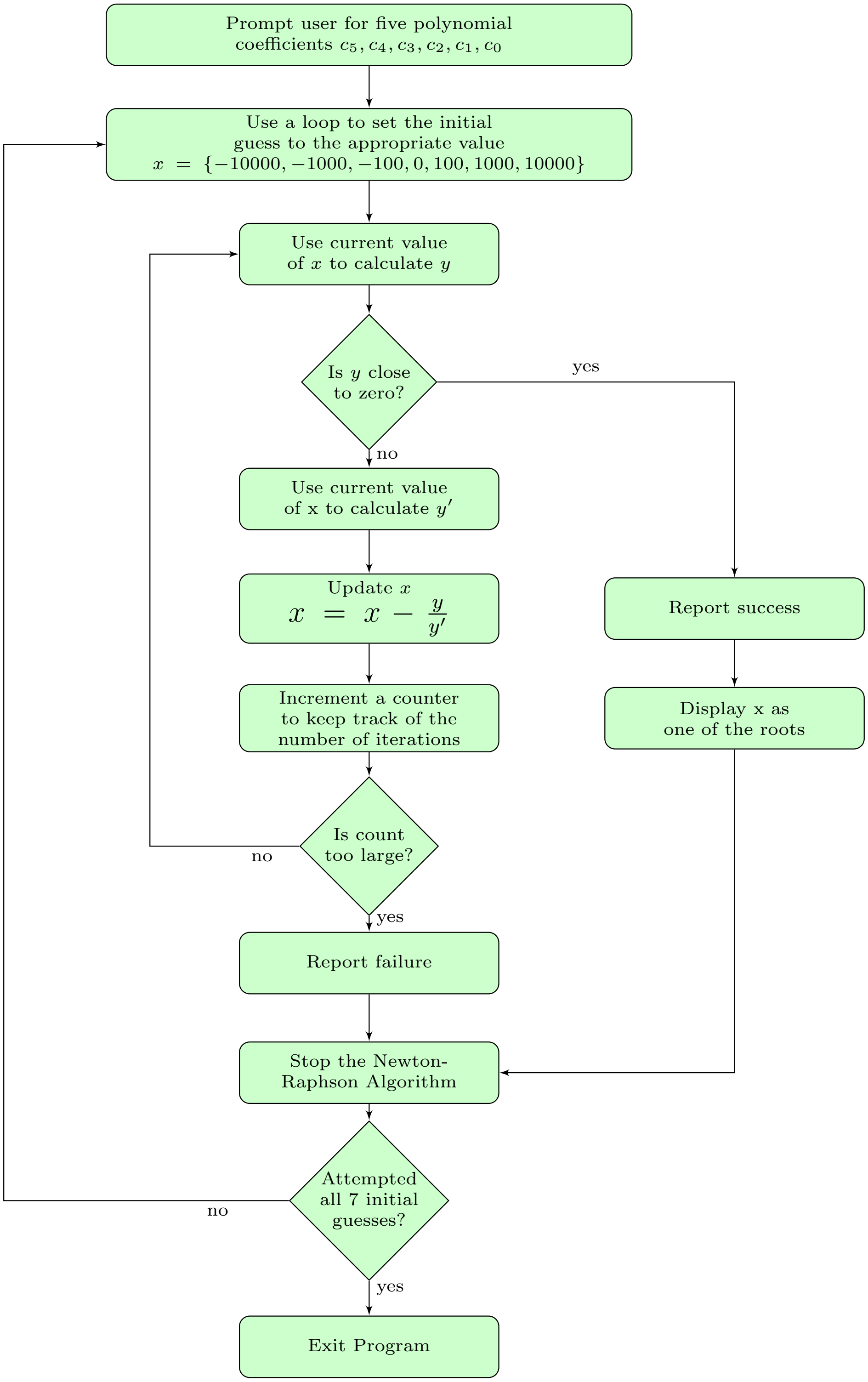
  (ii) If not then return to step a

We would like to use 7 different initial guesses to test the Newton-Raphson algorithm;

$$X_{InitialGuess} = \{-10000, -1000, -100, 0, 100, 1000, 10000\}$$

Thus your code will need to loop around the Newton-Raphson algorithm 7 times, once for each initial guess.

This same information is described in the flow chart below:


Prompt user for five polynomial coefficients $c_5, c_4, c_3, c_2, c_1, c_0$
Use a loop to set the initial guess to the appropriate value $x = \{-10000, -1000, -100, 0, 100, 1000, 10000\}$
Use current value of x to calculate y
Is y close to zero?
yes
no
Use current value of x to calculate y'
Update x $x = x - \frac{y}{y'}$
Increment a counter to keep track of the number of iterations
Is count too large?
no
yes
Report failure
Report success
Display x as one of the roots
Stop the Newton-Raphson Algorithm
Attempted all 7 initial guesses?
no
yes
Exit Program

If a solution is not found within 500 iterations then terminate the Newton-Raphson loop, report a failure, and move to the next initial guess. Failure to converge will occur when the polynomial roots are all complex; for example, if the user enters $c_5 = 0, c_4 = 0, c_3 = 0, c_2 = 1, c_1 = 2, c_0 = 3$. These coefficients equate to a second order polynomial $y = x^2 + 2x + 3$, with roots $x = -1 \pm i\sqrt{2}$.

The search algorithm has converged when $y(x) = 0$. Since y is a floating point variable it will most likely never actually equal zero. If $|y| \leq 1E-5$ then this is close enough. $1E-5$ is a tolerance value. It is the amount of error in the solution that we are willing to tolerate. Use an *if* statement to determine if the solution has converged within a tolerance of $1E-5$. Use the *fabs* (from *math.h*) statement to calculate the absolute value of y.

Make sure your program reports out the following:

- The polynomial that the user has entered
- Whether or not the Newton-Raphson algorithm converged to a solution for each initial guess attempt
- If successful then the root of the polynomial that the algorithm found
- The number of iterations that the Newton-Raphson algorithm required before converging on a solution

Also, add any numerical protections to the algorithm that you find necessary *HINT: Divide by zero protection?*

2.2 Test your program

Fill out the following table to test your program

Test Case	c_5	c_4	c_3	c_2	c_1	c_0	Converged?	First Root Found
1	0	0	1	-3	-760	-1500	Yes	-25
2	0	0	1	-3	-760	-1500	Yes	?
3	0	0	1	-3	-760	-1500	Yes	?
4	0	1	-4	1	36	200	?	?
5	1	2	3	4	5	6	?	?
6	0	1	9.7	-23.8	-179.5	285	?	?
7	0	1	9.7	-23.8	-179.5	285	?	?
8	0	0	0	1	-20	75	?	?
9	0	0	0	1	-20	75	?	?
10	1	15	0	0	4	60	?	?

2.3 Sample Code Execution

Sample Code Execution: Red text indicates information entered by the user

```
This program is to find one root of 5th-order polynomial using Newton-Rhapson method.
        c5x^5 + c4x^4 + c3x^3 + c2x^2 + c1x + c0

Enter polynomial coefficients: c5 c4 c3 c2 c1 c0 in this order:
0 0 1 -3 -760 -1500
Your polynomial is 1.0x^3 + -3.0x^2 + -760.0x + -1500.0

One of the roots of this polynomial is -25.00000
Starting from an initial guess of x = -10000.0, this answer was obtained in 19 iterations.

One of the roots of this polynomial is -25.00000
Starting from an initial guess of x = -1000.0, this answer was obtained in 14 iterations.

One of the roots of this polynomial is -25.00000
Starting from an initial guess of x = -100.0, this answer was obtained in 8 iterations.

One of the roots of this polynomial is -2.00000
Starting from an initial guess of x = 0.0, this answer was obtained in 3 iterations.

One of the roots of this polynomial is 30.00000
Starting from an initial guess of x = 100.0, this answer was obtained in 8 iterations.

One of the roots of this polynomial is 30.00000
Starting from an initial guess of x = 1000.0, this answer was obtained in 13 iterations.

One of the roots of this polynomial is 30.00000
Starting from an initial guess of x = 10000.0, this answer was obtained in 19 iterations.
```

Sample Code Execution: **Red text indicates information entered by the user**

```
This program is to find one root of 5th-order polynomial using Newton-Rhapson method.
        c5x^5 + c4x^4 + c3x^3 + c2x^2 + c1x + c0

Enter polynomial coefficients: c5 c4 c3 c2 c1 c0 in this order:
0 1 -4 1 36 200
Your polynomial is 1.0x^4 + -4.0x^3 + 1.0x^2 + 36.0x + 200.0

The initial guess of x = -10000.0 failed to converge to a solution; roots may be complex

The initial guess of x = -1000.0 failed to converge to a solution; roots may be complex

The initial guess of x = -100.0 failed to converge to a solution; roots may be complex

The initial guess of x = 0.0 failed to converge to a solution; roots may be complex

The initial guess of x = 100.0 failed to converge to a solution; roots may be complex

The initial guess of x = 1000.0 failed to converge to a solution; roots may be complex

The initial guess of x = 10000.0 failed to converge to a solution; roots may be complex
```

Sample Code Execution: **Red text indicates information entered by the user**

```
This program is to find one root of 5th-order polynomial using Newton-Rhapson method.
        c5x^5 + c4x^4 + c3x^3 + c2x^2 + c1x + c0

Enter polynomial coefficients: c5 c4 c3 c2 c1 c0 in this order:
1 17 30 -4 -64 -60
Your polynomial is 1.0x^5 + 17.0x^4 + 30.0x^3 + -4.0x^2 + -64.0x + -60.0

One of the roots of this polynomial is -15.00000
Starting from an initial guess of x = -10000.0, this answer was obtained in 36 iterations.

One of the roots of this polynomial is -15.00000
Starting from an initial guess of x = -1000.0, this answer was obtained in 25 iterations.

One of the roots of this polynomial is -15.00000
Starting from an initial guess of x = -100.0, this answer was obtained in 15 iterations.

One of the roots of this polynomial is -1.41421
Starting from an initial guess of x = 0.0, this answer was obtained in 5 iterations.

One of the roots of this polynomial is 1.41421
Starting from an initial guess of x = 100.0, this answer was obtained in 20 iterations.

One of the roots of this polynomial is 1.41421
Starting from an initial guess of x = 1000.0, this answer was obtained in 30 iterations.

One of the roots of this polynomial is 1.41421
Starting from an initial guess of x = 10000.0, this answer was obtained in 41 iterations.
```