ECE15: Homework #4

Notes:

- In the sample runs below, computer output is shown in black, and user inputs in red.
- To be accepted by the automated homework checker, your output should match the output shown exactly, including spelling, capitalization, punctuation, and spacing.
- To clarify the number of spaces, we show them as ␣. In your program, use regular spaces.
- Comment your programs clearly. We may deduct points for under-commented programs.
- You may assume that the user will always enter the input correctly.

Problem 1

Write a program `polyroot.c` that uses the Newton-Raphson algorithm to find a root of a degree-3 polynomial. The program prompts the user for integer coefficients a_0, a_1, a_2, a_3 and a double-precision initial point x_0, and approximates a solution to the equation

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 = 0.$$

For example, if the coefficients are 1, 2, 2, 3, then the program approximates a solution to the equation $1 + 2x + 2x^2 + 3x^3 = 0$. The program stops after 50 iterations or when $|f(x)| < 10^{-5}$, whichever comes first, and prints the result with a 5-decimal point precision.

Note: Store the coefficients a_0, a_1, a_2, a_3 in an integer array.

Sample runs:

```
(~)$ a.out
Coefficients: 1 2 2 3
Initial point: 0
-0.55232
(~)$ a.out
Coefficients: 2 -2 0 1
Initial point: 1
1.00000
(~)$
```

Problem 2

Write a program `sentence.c` that prompts the user for a sentence (ending in a period) and prints it, possibly in multiple lines, so that:

- each line has at most 15 characters,
- lines can be broken only at a space, and this space does not get printed.

You can assume that all entered words are less than 15 characters long, and that the length of the entered sentence is at most 500 characters.

Sample runs:

```
(~)$ a.out
Sentence: C programming is very cool.
C programming
is very cool.
(~)$ a.out
Sentence: Thanksgiving is always on a Thursday.
Thanksgiving is
always on a
Thursday.
(~)$ a.out
Sentence: The rain in Spain stays mainly in the plain.
The rain in
Spain stays
mainly in the
plain.
(~)$
```

Hint: One way to implement this program is to read each word using "%s" into a character array of length 15, check the word's length and whether it ends with a ".", and then print it on the same or a new line depending on how many characters have already been printed on that line.

Problem 3

Write a program `monotone.c` that prompts the user for length of a sequence and then a sequence of integers and converts it to an increasing sequence of numbers via a sequence of iteration. In each iteration, integers that are strictly less than their preceding integer are eliminated and the resulted sequence is printed. This process continues until a non-decreasing sequence of numbers remains.

Sample runs:

```
(~)$ a.out
Length of sequence: 8
Sequence: 0 4 2 -5 6 12 23 9
0 4 6 12 23
(~)$ a.out
Length of sequence: 5
Sequence: 1 1 2 3 5
1 1 2 3 5
(~)$ a.out
Length of sequence: 7
Sequence: 10 20 8 15 40 7 33
10 20 15 40 33
10 20 40
(~)$ a.out
Length of sequence: 4
Sequence: 50 10 20 30
50 20 30
50 30
50
(~)$ a.out
Length of sequence: 1
Sequence: 88
88
(~)$
```

Problem 4

Write a program `zombie.c` that prompts the user for a 5×5 array of zeros and ones (entered consecutively row by row) and run the following game for three times:

- Each element of the array is either a *zombie* (if it is 0) or a *human* (if it is 1).
- The neighbors of each elements are those horizontally, vertically, or diagonally adjacent to it, hence corner elements have three neighbors, elements on the side have five neighbors, and elements in the middle have eight neighbors.
- At each iteration of the game the following events occur:
  - Any human with less than two (namely 0 or 1) human neighbors is attacked by the zombies and becomes a zombie.
  - Any zombie with more than two (namely 3 up to 8) human neighbors is cured by the humans and becomes a human.

Denoting zombies by - and humans by $*$, the program prints the initial state of the game (provided by the user), and then runs the above iterations three times, printing the new state each time.

Sample run:

```
(~)$ a.out
Enter 5*5 array: 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
State 0:
- - - - -
- - - - -
- * * * -
- - - - -
- - - - -
State 1:
- - - - -
- - * - -
- - * - -
- - * - -
- - - - -
State 2:
- - - - -
- - - - -
- * * * -
- - - - -
- - - - -
State 3:
- - - - -
- - * - -
- - * - -
- - * - -
- - - - -
(~)$
```

```
(~)$ a.out
Enter 5*5 array: 1 1 0 1 1 1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1
State 0:
* * - * *
* - - - *
- - * - -
* - - - *
* * - * *
State 1:
* * - * *
* * - * *
- - - - -
* * - * *
* * - * *
State 2:
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
State 3:
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
(~)$
```