Before you begin your work, please create a new file folder on your computer. The name of the folder should be YourLastName_YourFirstName For example, if your name is John Smith your folder should be named Smith_John. Please store all your C files in that folder. Name your C files simply 1.c, 2.c, 3.c, 4.c, and 5.c.

At the end of the exam you must copy that folder onto a USB stick. Also, you must e-mail your files to the instructor.

Please attach all of your *.c files to one e-mail.
Use this subject line: "CprE185: Midterm 2, Section X"
Where X is your lab section (ask your TA if you don't know it).

Please DO NOT leave the room until you have done these 2 things:

- **copied your folder onto the USB memory stick**
- **and e-mailed your *.c files!!!**

1. Palindromes (10 points)

Write a complete C program that checks if a number entered by the user is a palindrome. A palindrome is defined as a number that can be written in the same way left-to-right and right-to-left. For example, 121, 3553, 531135, 2795972, are all palindromes, but 453,1232, 123123 are not palindromes.

The user must enter the number on one input line as a single integer, i.e., it is not OK to ask the user to enter the individual digits separately. You can assume that the user will always enter a positive integer number that fits within the size limits of the int data type.

```
============= START OF SAMPLE RUN ======================
Enter an integer number: 121

The number is a palindrome.
=============  END OF SAMPLE RUN  ======================
```

```
============= START OF SAMPLE RUN ======================
Enter an integer number: 453

The number is NOT a palindrome.
=============  END OF SAMPLE RUN  ======================
```

```
============= START OF SAMPLE RUN ======================
Enter an integer number: 5

The number is a palindrome.
=============  END OF SAMPLE RUN  ======================
```

2. Linearly Independent Vectors (15 points)

Two vectors are linearly independent if and only if there is no scalar value by which one can multiply the first vector to get the second vector. If two vectors are not linearly independent, then they are linearly dependent.

To illustrate linear dependence, let v1 be the vector (1, 2, -1) and let v2 be the vector (2, 4, -2). The two vectors v1 and v2 are linearly dependent because we can multiply v1 by a scalar to and get a result that is equivalent to v2. In this case that scalar is equal to 2:

```
      2 * v1 = 2 * (1, 2, -1) = (2*1, 2*2, 2*(-1)) = (2, 4, -2) = v2
```

To illustrate linear independence, let v1 be the vector (1, 2, -1), let v2 be the vector (1, 4, -1). The two vectors are linearly independent because there is no scalar by which we can multiply v1 to get v2.

Write a C program that prints whether two vectors with the same number of elements are either linearly independent or linearly dependent. The program must first read the size, N, of the vectors. Next, the program must read the elements of the two vectors as doubles. Finally, the program must print the result. See the sample runs shown below.

Hint: The vectors are linearly dependent if either vector is all zeros. Watch out for other cases containing zeros.

```
============= START OF SAMPLE RUN =======================
3
1 2 -1
1 4 -1
The two vectors are linearly independent.
=============  END OF SAMPLE RUN  =======================

============= START OF SAMPLE RUN =======================
3
1 0 0
0 1 0
The two vectors are linearly independent.
=============  END OF SAMPLE RUN  =======================

============= START OF SAMPLE RUN =======================
5
0  2.2  3.0 -0.1  2.1
0 -2.2 -3.0  0.1 -2.1
The two vectors are linearly dependent.
=============  END OF SAMPLE RUN  =======================

============= START OF SAMPLE RUN =======================
8
1 2 2 3.5 4 -1 -1 0
2 4 4   7 8 -2 -2 0
The two vectors are linearly dependent.
=============  END OF SAMPLE RUN  =======================
```

3. Closest Point (15 points)

Write a C program that asks the user to enter two integers: N and K. The program then ask the user to enter a list of N Cartesian points with integer coordinates from the keyboard (one 2D point per line). The program must then find the closest point to the K-th point in that list and print its index. It must also print the distances to the k-th point from all other points. Several sample runs are provided below. Please match the output formatting.

HINT: A point is always at distance 0 from itself. Your program must handle this case appropriately and print the closest point that is different from the k-th point.

```
============= START OF SAMPLE RUN =======================
Enter the number of points (N): 2
Enter the base point (K): 1
Enter Point #1: 1 1
Enter Point #2: 2 2

Distances from point #1
#1: 0
#2: 1.41421

Point #2 is closest to point #1
=============  END OF SAMPLE RUN  =======================
```

```
============= START OF SAMPLE RUN =======================
Enter the number of points (N): 4
Enter the base point (K): 2
Enter Point #1: 5 5
Enter Point #2: 3 3
Enter Point #3: 2 2
Enter Point #4: 3 3

Distances from point #2
#1: 2.82843
#2: 0
#3: 1.41421
#4: 0

Point #4 is closest to point #2
=============  END OF SAMPLE RUN  =======================
```

4. Border Problem (15 points)

Your task is to print a number of single-digit integers inside a border of asterisks. The input will consist of a list of integers that need to be printed within the border. There will also be a number indicating how many integers should be placed per line inside the border.

The input is formatted as follows. The first line contains a number N indicating the number of single-digit integers that will follow.
The second line contains N single-digit integers that will need to be printed. The third line contains only one number L that indicates how many integers can be printed per line. If L < N then the integers must be placed on multiple lines within the border. If L does not divide N evenly, then the last line of the output will have less than L numbers.

You can assume that N and L will always be greater than or equal to 1.

Make sure that a single space (either horizontal or vertical) separates the border from the numbers inside. Also, place a space between digits on the same line.

Here are two sample test runs:

```
---------------- START OF INPUT -------------------------
1
1
1
---------------- END OF INPUT ---------------------------
---------------- START OF OUTPUT ------------------------
*****
*   *
* 1 *
*   *
*****
----------------- END OF OUTPUT ------------------------
```

```
---------------- START OF INPUT -------------------------
7
7 6 5 4 3 2 1
3
---------------- END OF INPUT ---------------------------
---------------- START OF OUTPUT ------------------------
*********
*       *
* 7 6 5 *
* 4 3 2 *
* 1     *
*       *
*********
----------------- END OF OUTPUT ------------------------
```

5. Run-length Encoding (15 points)

“Run-length encoding (RLE) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and a count, rather than as the original run. This is most useful on data that contains many such runs; for example, simple graphic images such as icons and line drawings. For example, consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text.” (Source: Wikipedia)

As an example, consider this scan line, where 0 represents a black pixel and 1 represents a white pixel: 00111000110111100001

If apply the run-length encoding algorithm to that scan line, then we will get the following encoded result: 2B3W3B2W1B4W4B1W

Interpret this as 2 Black pixels, 3 White pixels, 3 Black pixels, etc.

Write a complete C program that reads a line of 20 pixels from the user (represented with 0's and 1's), encodes them using run-length encoding, and prints the encoding on the screen. Sample runs are shown below.

```
============= START OF SAMPLE RUN =======================
Original:  00001110111111100001
Encoded:   4B3W1B7W4B1W
=============  END OF SAMPLE RUN  =======================
```

```
============= START OF SAMPLE RUN =======================
Original:  11110011000011100000
Encoded:   4W2B2W4B3W5B
=============  END OF SAMPLE RUN  =======================
```

```
============= START OF SAMPLE RUN =======================
Original:  01001100011100001111
Encoded:   1B1W2B2W3B3W4B4W
=============  END OF SAMPLE RUN  =======================
```

```
============= START OF SAMPLE RUN =======================
Original:  11111100101001100000
Encoded:   6W2B1W1B1W2B2W5B
=============  END OF SAMPLE RUN  =======================
```

```
============= START OF SAMPLE RUN =======================
Original:  10101010101010101010
Encoded:   1W1B1W1B1W1B1W1B1W1B1W1B1W1B1W1B1W1B1W1B
=============  END OF SAMPLE RUN  =======================
```