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. Reversing an Array (10 points)

Write a C program that reverses the order of the elements of an integer array, i.e., the first element becomes the last and vice versa, the second element becomes the last but one and vice versa, etc.

Your program must do the following 4 things in order:

 * initialize an array of size 10 with **random** integers in the range 1-10
 * print the array by calling a function that you defined
 * reverse the entries of the array in the main function
 * print the reversed array by calling the same print function

```
============= START OF SAMPLE RUN =======================
4,   1,   8,   9,   5,  10,   2,   5,   4,   1,
1,   4,   5,   2,  10,   5,   9,   8,   1,   4,
=============  END OF SAMPLE RUN  =======================
```

```
============= START OF SAMPLE RUN =======================
4,   5,   9,   7,   7,   3,   7,   2,   9,   1,
1,   9,   2,   7,   3,   7,   7,   9,   5,   4,
=============  END OF SAMPLE RUN  =======================
```

```
============= START OF SAMPLE RUN =======================
6,  10,   4,   4,  10,   4,  10,   1,   1,   6,
6,   1,   1,  10,   4,  10,   4,   4,  10,   6,
=============  END OF SAMPLE RUN  =======================
```

2. The Best Mentor (15 points)

Write a complete C program that helps a local school identify the best student mentor. The school has many mentors and each mentor has one or more students. Different mentors may have different number of students. The performance metric that the school has chosen to use is the average GPA of the students in each mentor’s group, i.e., the higher the average GPA in the group the better the mentor.

In other words, this program must do the following things:

* read from the keyboard how many mentors there are
* read from the keyboard how many students each mentor mentors
* read from the keyboard the GPAs for the students in each group
* calculate the average GPA of the students for each mentor
* determine the best of the mentors.

The average GPA should be rounded to two decimal places, and the best GPA should be displayed using %.2lf.

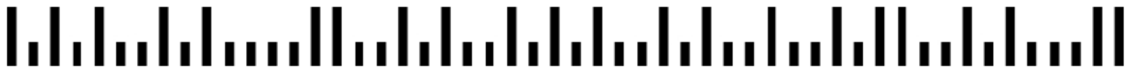
To make this problem easier, you can assume that no mentor has the same average GPA for his students as another mentor (i.e., no ties).

A sample run is provided below.

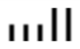
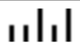
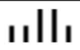
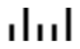
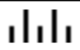
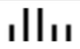
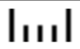
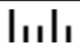
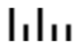
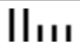
```
============= START OF SAMPLE RUN =======================
Enter the number of Mentors: 3

Enter the number of students for Mentor 1: 2
Enter the GPAs of his/her students:
4.00 3.56

Enter the number of students for Mentor 2: 3
Enter the GPAs of his/her students:
3.15 3.82 3.56

Enter the number of students for Mentor 3: 1
Enter the GPAs of his/her students:
3.75

Mentor 1 is the best mentor with an average GPA of 3.78.
=============  END OF SAMPLE RUN  =======================
```

3. POSTNET Barcodes (15 points)

Have you ever wondered why mail envelopes have barcodes printed on them? The United States Postal Service uses the POSTNET barcode to automate mail routing. The barcode encodes the zip code of the destination address as a sequence of long and short vertical bars. Here is an example:



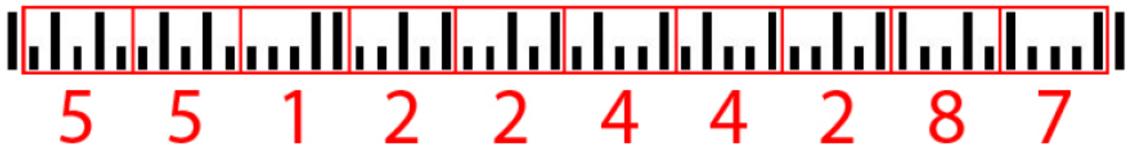
There are several POSTNET versions. This problem only focuses on the "C" version, which uses 52 bars. Here is how it works.

Value	Encoding
1	
2	
3	
4	
5	
6	
7	
8	
9	
0	

Every POSTNET barcode starts and ends with a full bar. This is commonly referred to as the guard rail. The remaining bars are grouped into sections of 5, each representing a single digit in the code. The table shows the encoding for each individual digit.

The first 9 digits represent the ZIP code (5 zip + 4 zip extension). The 10-th digit, often called the checksum digit, is not part of the zip code, but it is used for error detection in case the barcode is unreadable or damaged. The checksum digit is calculated based on the previous nine digits such that when all ten digits are added the result should be a multiple of 10. If the sum is already a multiple of 10 the checksum digit is 0.

Example: The barcode shown above encodes the zip code 55122-4428. Adding these nine digits we get 5+5+1+2+2+4+4+2+8=33. The tenth digit of the barcode must be 7, as 33+7=40 and 40 is a multiple of 10. In other words, 40%10=0. The figure below groups the individual digits.



Source: http://en.wikipedia.org/wiki/POSTNET

Write a complete C program that reads two ints from the keyboard. The first one represents the 5-digit zip code. The second one represents the 4-digit zip code extension. The program must print the POSTNET barcode for this 9-digit zip code using ASCII characters. You can use the '|' character to print a short vertical bar. Printing two of these on two consecutive lines will give you a long vertical bar. Here is an example of a sample run of the program:

```
Enter 5-digit zip code:  50014
Enter 4-digit extension: 1234
50014-1234 in POSTNET is

| | | ||   ||      || |  |   ||   | |  ||  |  |||  |
||||||||||||||||||||||||||||||||||||||||||||||||||||
```

4. Calendar (15 points)

Write a complete C program that prints a monthly calendar for any month in 2012. Your program can assume (and not calculate) that Jan 1, 2012 is a Sunday and also that 2012 is a leap year (i.e., February has 29 days). It is OK to hardcode an integer array in your program that has the number of days in each month, e.g., int days[12]={31, 29, 31,....}.

The month is specified by the user as a number from 1 to 12. The output must be in the format specified the three sample runs given below.

Important: Your program must calculate the calendar for each month. You can't simply hardcode a set of printf statements for each month. Also, you can't call the calendar utility of the operating system do the job. (The 'cal' program that is available in Unix inspired this problem.)

```
> Please enter a month [1-12]: 1

     January 2012
 Su Mo Tu We Th Fr Sa
  1  2  3  4  5  6  7
  8  9 10 11 12 13 14
 15 16 17 18 19 20 21
 22 23 24 25 26 27 28
 29 30 31
```

```
>Please enter a month [1-12]: 3
     March 2012
 Su Mo Tu We Th Fr Sa
              1  2  3
  4  5  6  7  8  9 10
 11 12 13 14 15 16 17
 18 19 20 21 22 23 24
 25 26 27 28 29 30 31
```

```
 >Please enter a month [1-12]: 10

     October 2012
 Su Mo Tu We Th Fr Sa
     1  2  3  4  5  6
  7  8  9 10 11 12 13
 14 15 16 17 18 19 20
 21 22 23 24 25 26 27
 28 29 30 31
```

5. Guessing Game (15 points)

Write a complete C program that asks the user to enter an integer number between 1 and 100. The program then tries to guess that number. After every guess the user must answer Yes (if the guess was correct), Higher (if the next guess should be higher), or Lower (if the next guess should be lower). The letters Y,H,L are used as shortcuts.

If the user's answers are always truthful, then the program will eventually guess the number. If the user's answers are not truthful, then the program should detect that and print "You lied to me!"

Try to optimize your program so that it does not exhaustively try to guess all possible numbers. Instead, it should use the higher/lower answers to restrict the range of possible remaining numbers.

Note: even though the program "knows" what number the user entered (it is stored in a variable after all) it **should not** use the value of that variable in the guessing algorithm. The only reason the program reads this number is so that the user can better keep track of his answers.

```
============= START OF SAMPLE RUN =======================
Enter a number between 1 and 100: 70
Is your number equal to 50?
Please answer Higher, Lower, or Yes [H|L|Y]: h
Is your number equal to 75?
Please answer Higher, Lower, or Yes [H|L|Y]: l
Is your number equal to 62?
Please answer Higher, Lower, or Yes [H|L|Y]: h
Is your number equal to 68?
Please answer Higher, Lower, or Yes [H|L|Y]: h
Is your number equal to 71?
Please answer Higher, Lower, or Yes [H|L|Y]: l
Is your number equal to 69?
Please answer Higher, Lower, or Yes [H|L|Y]: h
Is your number equal to 70?
Please answer Higher, Lower, or Yes [H|L|Y]: y
I guessed it correctly!
=============  END OF SAMPLE RUN  =======================
```