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, 5a.c and 5b.c. At the end of the exam you will have to copy that folder onto a USB memory stick. Also, you will have to upload your solutions to WebCT (just like a homework but this one is named "Midterm2 Lab").

Please do not leave the room until you have copied your files onto the USB memory stick and also uploaded them on WebCT!!!

1. Random Phone Numbers (10 points)

Write a complete C program that generates 10 random phone numbers and prints them on the screen. The numbers must be formatted as shown in the example given below. The 10 numbers must be different every time your program is called. The program is not required to check if these are valid phone numbers (e.g., some of them may start with a zero or the area code may be invalid).

SAMPLE RUN:

```
(142)-677-8454
(445)-443-5049
(280)-766-7669
(776)-046-4139
(432)-903-8187
(901)-893-4787
(067)-439-5665
(107)-037-6924
(775)-917-8436
(237)-906-3133
```

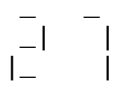
2. Seven Segment Indicators (15 points)

Seven segment indicators are commonly used to display numbers 0-9. Using an ASCII approximation the numbers can be represented as:

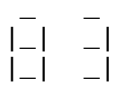
```
 _       _   _       _   _   _   _   _
| |   |  _|  _| |_| |_  |_    | |_| |_|
|_|   | |_   _|   |  _| |_|   | |_|  _|
```

Write a complete C program that generates two random numbers in the range [0 .. 9] and prints them on the screen side by side using an ASCII encoding of their seven-segment indicator representation (as shown in the sample runs given below). The program must come up with a different pair of numbers every time you run it. It is OK if during some (but not all) runs the two random numbers are the same.

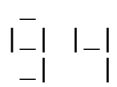
SAMPLE RUN #1:



SAMPLE RUN #2:



SAMPLE RUN #3:



3. Bisection Method (15 points)

MATH BACKGROUND:

The bisection method is one of the simplest algorithms for finding the roots of a function f(x), i.e., for finding when f(x)=0. To use this method, however, one must provide an initial interval [a,b] that contains or brackets the root, such that f(a) and f(b) have opposite signs.

The algorithm repeatedly divides the interval [a,b] into two equal subintervals [a,c] and [c,b] and focuses its attention on the one that continues to bracket the root. This condition is met if the function has opposite signs at the beginning and at the end of the subinterval.

More formally the iterative algorithm consists of the following steps:

Step 1: calculate the mid-point c = (a+b)/2 of the interval [a,b]

Step 2: if f(c) == 0, then c is the root and we are done
else if f(a)*f(c) < 0, then make b = c
else if f(c)*f(b) < 0, then make a = c

Step 3: if |a-b| < EPSILON, then stop and declare that c is the root

Step 4: Otherwise go to Step 1 using the new values for and and b

PROBLEM DESCRIPTION:

Write a complete C program that implements the bisection method and uses it to calculate the root of a given function in a user specified interval (e.g., f(x) = x*x -2 in the interval [1,2]). You can hardcode several functions in your code and uncomment one of them for testing. Something like this will do:

```
double f(double x)
{
     return x*x -2;  // the root is sqrt(2)
     //return x*x -4;  // the root is 2
}
```

Format the output similar to the sample runs provided below. These values were obtained with EPSILON=0.00001 and using %.7f to print them.

SAMPLE RUN #1: (f(x) = x*x -2; the root should be sqrt(2))

```
Please enter a: 1
Please enter b: 2
      a           b           b-a         f(a)       f(b)
-------------------------------------------------------------
[ 1.0000000,  2.0000000]   1.0000000  -1.0000000  2.0000000
[ 1.0000000,  1.5000000]   0.5000000  -1.0000000  0.2500000
[ 1.2500000,  1.5000000]   0.2500000  -0.4375000  0.2500000
[ 1.3750000,  1.5000000]   0.1250000  -0.1093750  0.2500000
[ 1.3750000,  1.4375000]   0.0625000  -0.1093750  0.0664063
[ 1.4062500,  1.4375000]   0.0312500  -0.0224609  0.0664063
[ 1.4062500,  1.4218750]   0.0156250  -0.0224609  0.0217285
[ 1.4140625,  1.4218750]   0.0078125  -0.0004272  0.0217285
[ 1.4140625,  1.4179688]   0.0039063  -0.0004272  0.0106354
[ 1.4140625,  1.4160156]   0.0019531  -0.0004272  0.0051003
[ 1.4140625,  1.4150391]   0.0009766  -0.0004272  0.0023355
[ 1.4140625,  1.4145508]   0.0004883  -0.0004272  0.0009539
[ 1.4140625,  1.4143066]   0.0002441  -0.0004272  0.0002633
[ 1.4141846,  1.4143066]   0.0001221  -0.0000820  0.0002633
[ 1.4141846,  1.4142456]   0.0000610  -0.0000820  0.0000906
[ 1.4141846,  1.4142151]   0.0000305  -0.0000820  0.0000043
[ 1.4141998,  1.4142151]   0.0000153  -0.0000388  0.0000043

The root of the function in this interval is x=1.41421
```

SAMPLE RUN #2: (f(x) = x*x -4; the root should be 2)

```
Please enter a: -1
Please enter b: 3
      a           b           b-a         f(a)       f(b)
-------------------------------------------------------------
[-1.0000000,  3.0000000]   4.0000000  -3.0000000  5.0000000
[ 1.0000000,  3.0000000]   2.0000000  -3.0000000  5.0000000
Found it at c=2.0000000!

The root of the function in this interval is x=2
```

4. Binary to ASCII Converter (15 points)

Write a complete C program that takes a message encoded in binary and prints its letters in ASCII format on the screen. You can assume that the message is stored in a two dimensional integer array that is hardcoded in the program. In other words, you don't have to read it from the keyboard. The following will suffice:

```
int len = 3; // number of letters
int message[3][8] = {{0,1,0,0,0,0,1,1},  // C
                     {0,1,1,0,0,0,0,1},  // a
                     {0,1,1,1,0,1,0,0}}; // t
```

To test the program you can keep several messages (commented out) and uncomment only one for testing. Here is another example:

```
/*
int len = 5;
int message[5][8] = {{0,1,0,0,1,0,0,0},  // H
                     {0,1,1,0,0,1,0,1},  // e
                     {0,1,1,0,1,1,0,0},  // l
                     {0,1,1,0,1,1,0,0},  // l
                     {0,1,1,0,1,1,1,1}}; // o
*/
```

Each letter of the message is encoded as an 8-bit binary number that corresponds to the ASCII code of the letter. The binary digits for each letter are stored in a separate row of the array. The program must first print the message in binary (one 8-bit number per line) and then print the decoded message in ASCII (see the sample runs below).

Sample Run #1:

```
01000011
01100001
01110100
Cat
```

Sample Run #2:

```
01001000
01100101
01101100
01101100
01101111
Hello
```

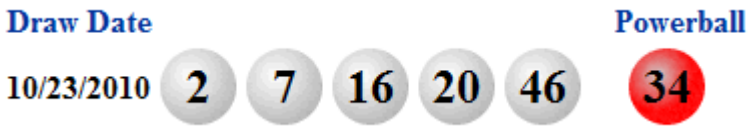
Sample Run #3:

```
01001000
01100001
01101100
01101100
01101111
01110111
01100101
01100101
01101110
Halloween
```

5. Powerball Lottery (15 points)

The Powerball Lottery (http://www.powerball.com/) is quite popular in the US because it often has huge jackpots that could make the owner of a winning ticket an instant millionaire.

Every Wednesday and Saturday night they draw 5 balls from a drum with 59 numbered white balls and one ball (the Powerball) from a drum with 39 numbered red balls. Here are the winning numbers from last Saturday:



To play the game you have to buy a $1 ticket and mark five numbers and one Powerball number. After the drawing you can check how many of your five numbers match the numbers on the five white balls (in any order) and whether or not your Powerball number matches the number on the red ball. There are nine possible ways to win a prize:

Your Numbers Match		Prize
5 Balls + Powerball	->	Jackpot
5 Balls	->	$200,000
4 Balls + Powerball	->	$10,000
4 Balls	->	$100
3 Balls + Powerball	->	$100
3 Balls	->	$7
2 Balls + Powerball	->	$7
1 Balls + Powerball	->	$4
0 Balls + Powerball	->	$3

Otherwise you win nothing.

Part A: (5 points) [Save your file as 5a.c]

Write a complete C program that generates a random drawing of the Powerball Lottery. In other words, it prints five numbers (white balls) in the range [1...59] and one number (red ball) in the range [1...39]. The white ball numbers can be drawn/printed in any order, but the balls are drawn without replacement so the same number cannot appear twice.

Part B: (10 points) [Save your file as 5b.c]

Modify your program from Part A to print a list of 20 computer generated tickets for the Powerball Lottery that players can purchase (about 80% of all tickets sold are computer generated). For each ticket the program must print if the ticket wins any prize (given the randomly drawn winning numbers from part A and the prize table given above).

SAMPLE RUN:

```
White Balls: 16, 12, 26, 42, 46,   Powerball: 37    <--DRAWING

White Balls:  2, 54, 55, 28, 34,   Powerball: 16    nothing   (0+0)
White Balls: 52, 26, 22,  5, 51,   Powerball: 37         $4   (1+1)
White Balls: 20, 58, 46, 47, 50,   Powerball: 28    nothing   (1+0)
White Balls: 44, 39, 36, 19, 50,   Powerball: 30    nothing   (0+0)
White Balls: 32, 17, 15,  2, 44,   Powerball: 14    nothing   (0+0)
White Balls: 36, 28, 52, 11,  8,   Powerball: 20    nothing   (0+0)
White Balls: 44,  5, 57,  6, 53,   Powerball: 28    nothing   (0+0)
White Balls: 27,  9, 48, 53, 40,   Powerball: 23    nothing   (0+0)
White Balls: 23, 18, 43, 35, 49,   Powerball: 35    nothing   (0+0)
White Balls:  7,  4, 26, 50, 46,   Powerball: 18    nothing   (2+0)
White Balls: 29,  4, 40, 37, 49,   Powerball: 37         $3   (0+1)
White Balls: 59,  6, 46, 48, 15,   Powerball: 17    nothing   (1+0)
White Balls:  4, 23, 52, 31, 55,   Powerball: 24    nothing   (0+0)
White Balls:  8, 24, 35,  5, 11,   Powerball: 16    nothing   (0+0)
White Balls: 15, 58, 20, 28, 42,   Powerball: 21    nothing   (1+0)
White Balls:  5, 34, 51,  2, 15,   Powerball:  6    nothing   (0+0)
White Balls: 10, 29, 30, 36, 21,   Powerball:  8    nothing   (0+0)
White Balls:  8,  1, 48, 18,  3,   Powerball: 27    nothing   (0+0)
White Balls:  1, 41, 56,  5, 14,   Powerball: 18    nothing   (0+0)
White Balls: 42,  1, 43,  5, 11,   Powerball: 32    nothing   (1+0)
```