**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. Integer Array (10 points)**

Use the following code snippet to initialize an integer array of size N with random numbers (both positive and negative) in the range [-20,20].

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 10

int main(int argc, char* argv[])
{
      srand(time(NULL));
      int a[N];
      int i;
      for(i=0; i<N; i++)    // initialize the array with random
            a[i]= rand()%41-20; // numbers in the range [-20,20]
}
```

The rest of your C program should print the following four things:

- the elements of the array on one line
- the smallest number in the array
- the largest number in the array
- the number of negative numbers in the array

```
============== Sample Output ==============
The array elements are:
 20,  -4, -11, -10,   5, -15,   2,   5,  11,  -1,
The minimum number is -15
The maximum number is 20
The array has 5 negative numbers
==========================================
```

**2. Parentheses (15 points)**


You may have heard of the programming language Lisp, or its more modern descendant, Scheme. One inescapable fact of Lisp programming is that you must use a lot of parentheses.

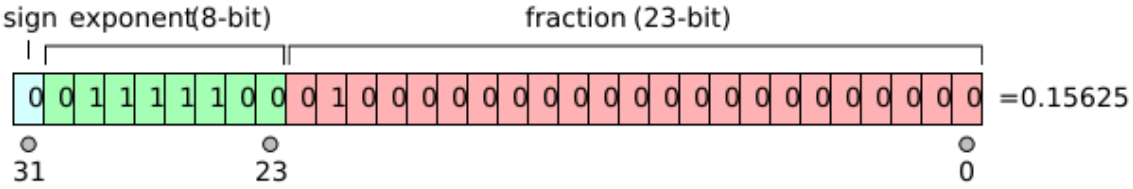
Suppose that you've just started learning Lisp. You keep writing expressions in which you forget to close some of the parenthesis. Write a program that validates that for every left-parenthesis character '(' input to your program, there is a corresponding right-parenthesis character ')'.

Hint: All inputs will be a single line of characters terminated with the newline character.

```
============= START OF SAMPLE RUN =========================
---------------- START OF INPUT --------------------------
(+ 7 (* 5 (+ 1 4)))
----------------- END OF INPUT ---------------------------
---------------- START OF OUTPUT -------------------------
Valid Input
----------------- END OF OUTPUT ---------------------------
============= END OF SAMPLE RUN ===========================


============= START OF SAMPLE RUN =========================
---------------- START OF INPUT --------------------------
(((((())))
----------------- END OF INPUT ---------------------------
---------------- START OF OUTPUT -------------------------
Invalid Input
----------------- END OF OUTPUT ---------------------------
============= END OF SAMPLE RUN ===========================


============= START OF SAMPLE RUN =========================
---------------- START OF INPUT --------------------------
)(
----------------- END OF INPUT ---------------------------
---------------- START OF OUTPUT -------------------------
Invalid Input
----------------- END OF OUTPUT ---------------------------
============= END OF SAMPLE RUN ===========================


============= START OF SAMPLE RUN =========================
---------------- START OF INPUT --------------------------
(cdr (car '((peas carrots tomatoes) (pork beef chicken))))
----------------- END OF INPUT ---------------------------
---------------- START OF OUTPUT -------------------------
Valid Input
----------------- END OF OUTPUT ---------------------------
============= END OF SAMPLE RUN ===========================
```

**3. Single-Precision Floating-Point Numbers (15 points)**

Just like we learned in class, floating-point numbers are stored in the IEEE 754 format. The figure shown below should refresh your memory.




$$v = (-1)^{sign} \times 2^{exponent - exponent\ bias} \times 1.fraction$$

Write a C program that reads a 32-bit single-precision binary number and prints its value using the "%g" formatting inside printf.

HINT: It may be easier to read the values of the individual bits as characters, and calculate the sign-bit, the 8 exponent bits, and the 23-bit mantissa bits separately.

```
============== Sample Output 1 ===============
Enter a single-precision binary number:

00111110001000000000000000000000

The value entered is: 0.15625
==============================================


============== Sample Output 2 ===============
Enter a single-precision binary number:

11000011110010010001000000100000

The value entered is: -402.126
==============================================


============== Sample Output 3 ===============
Enter a single-precision binary number:

01000000011000000000000000000000

The value entered is: 3.5
==============================================
```

**4. Circular Shift Register (15 points)**

In digital circuits, a circular shift register is a construct similar to an array that can shift its contents left or right as if they are arranged in a circle. During a right shift, each element in the register is moved one space to the right, and the last element goes to the beginning of the register. Similarly, during a left shift, each element is moved to the left and the first element goes to the end.

For example, a shift register with the values [1,0,1,1,1,0] would behave as follows:

```
Right shift by 1: [0,1,0,1,1,1]
Right shift by 2: [1,0,1,0,1,1]
Right shift by 5: [0,1,1,1,0,1]

Left shift by 1:  [0,1,1,1,0,1]
Left shift by 3:  [1,1,0,1,0,1]
```

You must write a complete C program that models a circular shift register. Your program must read in the contents of the register and take in a shift direction and amount. It must then output the shifted contents of the register. You must define two functions that shift the register by 1 in the specified direction (one function for left and one for right shift). You must call the appropriate function repeatedly to shift the register by the specified amount.

HINT: The register will contain at most 1024 bits. Do not worry about error checking – assume all inputs are correctly formatted.

```
====== SAMPLE RUN #1 ======
Enter the register size: 5
Enter the register contents: 1 1 0 1 0
Enter the shift direction [-1 for left, 1 for right]: 1
Enter the shift amount: 3

The contents of the shifted register are: 0 1 0 1 1
============================

====== SAMPLE RUN #2 ======
Enter the register size: 10
Enter the register contents: 1 1 0 0 0 1 1 0 1 0
Enter the shift direction [-1 for left, 1 for right]: -1
Enter the shift amount: 4387

The contents of the shifted register are: 0 1 0 1 1 0 0 0 1 1
============================
```

**5. Look-And-Say Sequence (15 points)**

The idea for this problem comes from Wikipedia:
http://en.wikipedia.org/wiki/Look-and-say_sequence

"The look-and-say sequence is the sequence of integers beginning as follows:
1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, ...

To generate a member of the sequence from the previous member, read off the digits of the previous member, counting the number of digits in groups of the same digit. For example:

1 is read off as "one 1" or 11.
11 is read off as "two 1s" or 21.
21 is read off as "one 2, then one 1" or 1211.
1211 is read off as "one 1, then one 2, then two 1s" or 111221.
111221 is read off as "three 1s, then two 2s, then one 1" or 312211."

You must write a complete C program that reads in an element of the sequence and outputs the element immediately after it. For example, entering 312211 should produce an output of 13112221.

You may assume that the element that was entered (and the element immediately after it) will be at most 1024 digits in length. The input will be a sequence of digits [1-3] and will be terminated by a single period character ('.'). Do not worry about error checking -- assume all inputs are correctly formatted.

```
====== SAMPLE RUN #1 ======
Enter the element: 1.
The next element in the sequence is: 11
===========================

====== SAMPLE RUN #2 ======
Enter the element: 111221.
The next element in the sequence is: 312211
===========================


====== SAMPLE RUN #2 ======
Enter the element: 13211311123113112211.
The next element in the sequence is: 11131221133112132113212221
===========================
```