University of Toronto
APS105H1F — Computer Fundamentals
Lab 7: Recursion

Introduction
This lab is designed to give you practice writing and using recursive C++ functions.

This lab has six parts. By the end of the lab session, you will have created six files, namely files `Lab7Part1.c` through `Lab7Part6.c`. Please keep in mind that the marking process will begin approximately sixty minutes before the end of the lab.

Before attempting this lab, you should review your lecture notes and read Chapter 14 (Recursion) from the Savitch text.

You should try to complete the lab in advance of your practical session and get help from a TA soon after arriving at SF 1013 so that you may promptly overcome any remaining difficulties.

A word of advice before you begin: Some students find recursion "mind-bending" others find it "obvious". You will move from the former group to the latter with enough thoughtful practice.

Part 1
Suppose that you have somehow been transported back to 1777 and the American Revolutionary War. You have been assigned a dangerous reconnaissance mission: evaluate the amount of ammunition available to the British for use with their large cannon which has been shelling the Revolutionary forces. Fortunately for you, the British, being neat and orderly, have stacked the cannonballs into a single pyramid-shaped stack with one cannonball at the top, sitting on top of a square composed of four cannonballs, sitting on top of a square composed of nine cannonballs, and so forth. Unfortunately, however, the British Redcoats are also vigilant, and you only have time to count the number of layers in the pyramid before you are able to escape back to your own troops.
Your mission is to write a recursive function named `cannonball` that takes as its argument the height of the pyramid and returns the number of cannonballs therein.
In addition, write a short `main` function that prompts the user for a pyramid height, calls your `cannonball` function, and prints the number of cannonballs in the pyramid.
Write your solution to this problem in a file named `Lab7Part1.c`.
HINT: The number of cannonballs in a pyramid of height h is h^2 plus the number of cannonballs in a pyramid of height $h-1$.

Part 2
Consider the mathematical function M that maps integers to integers defined by

$$M(n) = \begin{cases} n-10, & \text{if } n > 100, \\ M(M(n+11)), & \text{if } n \le 100. \end{cases}$$

In a file named Lab7Part2.c, write a `C++` function that can be used to evaluate M. Include an statement in your function that outputs the value of the parameter so that you can observe the parameter values for each function call.
And then write a `main` function that prompts the user for an integer value (say i), and then computes the value of $M(i)$. In total, your program should output the parameter values for each function call, and the values of both i and $M(i)$.

Run your program for various value of i and note any properties of the function M. What does the graph of i vs. $M(i)$ look like?

Part 3

Write a C++ function that uses recursion to print the binary (base-2) representation of a decimal integer. An algorithm for this conversion is to repeatedly divide the decimal number by 2, until it is zero. Each division produces a remainder of 0 or 1, which becomes a bit in the binary number. For example, if we want the binary representation of decimal 13, we find it with the following series of divisions:

```
13/2 = 6      remainder 1
 6/2 = 3      remainder 0
 3/2 = 1      remainder 1
 1/2 = 0      remainder 1
```

The binary representation of 13 is thus 1101. The only problem with this algorithm is that the first remainder generates the low-order bit, the next remainder generates the second-order bit, and so on, until the last remainder produces the high-order bit. Thus, if we output the bits as they are generated, they are in reverse order. Your recursive procedure should correct the order of the output bits.
You conversion function should first print the sign of the decimal integer and then call your recursive procedure for printing the bits.
Write a short `main` function that prompts the user for a decimal integer, and calls your decimal-to-binary conversion function which will output the binary representation of the decimal integer.
Write your solution to this problem in a file named `Lab7Part3.c`.

Part 4

Update your program from Part 3 so that it prints the hexadecimal (base-16) representation of a decimal integer. In hexadecimal, the letters 'A', 'B', 'C', 'D', 'E' and 'F' are used to represent the numbers 10, 11, 12, 13, 14 and 15, respectively.
Write your solution to this problem in a file named `Lab7Part4.c`.

Part 5

Write a C++ program that uses recursion to separate the negative and positive numbers in a list of integers, printing the negative numbers on the first line and the positive numbers on the second line. Write a brief main program that prompts the user for a number `n` between 5 and 50. Then read `n` numbers into an array and call your recursive function (described next) to separate them. Write a recursive function named `negFirst` to process the array. If an element is negative, the function should print it before making the next recursive call. If it is positive, the function should call itself recursively to print the other numbers and then print the positive number afterward.
Write your solution to this problem in a file named `Lab7Part5.c`.

Part 6

Update your program from Part 3 so that it prints the binary representation of a `double` type number in fixed-point form. Numbers of type `double` have floating-point representation

$$\pm 0.d_1 d_2 d_3 \ldots d_{53} \times 2^E$$

where d_1 is 1 and d_i is either 0 or 1, $i = 2, 3, 4, \ldots, 53$, and the integer E satisfies $-1021 \leq E \leq 1024$.
The number $\pm 0.d_1 d_2 d_3 \ldots d_{53}$ is called the mantissa of the floating-point number and the number E is called the exponent.
For example, the fixed-point binary representation of the decimal number 9.625, which is $0.10011010\ldots 0 \times 2^4$ in binary, is 1001.101. And the fixed-point binary representation of the decimal number 18, which is $0.10010\ldots\ 0 \times 2^5$, is 10010.0.
You may assume that the `double` numbers to be printed are between 1 and $2^{31} - 1$ in size.

Your program may not contain any loops and must use recursion.
You may care to review the `cmath` functions on page 185 of the text.
Write a short `main` function that prompts the user for a “real” number, stores it as a type `double` variable, and calls your binary conversion function which will output the fixed-point binary representation of the `double` variable.
Write your solution to this problem in a file named `Lab7Part6.c`.

BONUS: (2 marks) Modify your program so that it can print `double` values that are bigger than $2^{31} - 1$ or smaller than 1 in size.
Note that the fixed-point binary representation of the decimal number 0.375, which is $0.110\ldots0 \times 2^{-1}$ in binary, is 0.011.
You may find the following functions helpful:

```
// Helper function that calls the cmath frexp function.
// Returns the mantissa of x and sets the variable e to
// the value of the exponent of x
double h_frexp( double x, int& e )
{
    return frexp( x, &e );
}

// The ldexp function returns the result of multiplying the
// floating-point number x by 2 raised to the power exp
// It is in the cmath library.
double ldexp(double x, int exp);
```

Requirements

Your programs must be in files named `Lab7Part1.c` through `Lab7Part6.c`. There should be additional meaningful comments in your source code.

You must be able to compile your programs on the ECF system and demonstrate them for the TA who marks your lab.

Submission

Before your lab is over you **must** submit your file using the unix `submitaps105f` command. Use the command `submitaps105f 7 Lab7Part[1-6].c` to submit your files. The manual page for this command may be read by executing the unix `man submit` command.

The command `submitaps105f -l 7` may be used to confirm that you have successfully submitted your files.