CSC230 Lab 5

Goal: This lab includes function and recursion.

Please try your best to finish the lab in class, and submit it to CANVAS.

In this lab, please write a **Lab5.cpp** file. There is main function in this file. Please define one more function in the lab.

In the lecture, we know that a recursive definition has two parts:

- Base case
- Reduction step

A good example of recursive definition is factorial:

$$\begin{cases} 0! = 1 \\ N! = N * (N-1)!, \ \ if\ N > 0 \end{cases}$$

The corresponding recursive program can be:

```
int factorial(int number) {
   int temp;
   if(number <= 1) return 1;
   temp = number * factorial(number - 1);
   return temp;
}
```

The iteration (loop) solution can be:

```
int factorial(int n) {
     int temp = 1;
     for (int counter = 1; counter <= n; counter++)
          temp = temp * counter;
     return fact;
}
```

As you can see, the recursion and iteration perform the same task. Both of them solve a large task one piece at a time, and combine the results. However, the emphasis of iteration is to keep repeating until the task is "done". While the emphasis of recursion is to break up the large problem into smaller pieces until you can solve it, then combine the results.

There is no clear answer to which one is better. But usually mathematicians often prefer recursive definition; programmers often prefer iterative solutions. As a programmer, when you define recursive functions, please keep in mind that whenever a function is called, it will incur some overhead (prepare for the stack, switch control from one function to another function, clean up the stack frame, etc.). If a recursive function causes

excessive function invocations, the added overhead may be overwhelming. Think about the Fibonacci sequence as an example.

An important notation related to recursion is **backtracking**. "Backtracking is a general algorithm for finding all (or some) solutions to some computational problem, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons each partial candidate c (backtracking) as soon as it determines that c cannot be completed to a valid solution." ----- wikipedia

A classic example of using backtracking is the eight queen puzzle, that asks for all arrangements of eight chess queens on a standard chessboard so that no queen attacks any other. Google Developer has very good explanation of a generalized eight queen puzzle – N-queen Problem at https://developers.google.com/optimization/puzzles/queens . A C++ solution can be found at https://en.wikibooks.org/wiki/Algorithm_Implementation/Miscellaneous/N-Queens. Please take some time to read these two webpages, understand the idea of the algorithm and the code. It is a popular interview question.

In this lab, define a global array called A with size 100. The data type of the array is int. Design a recursive function called **com**(int n, int t) inside **Lab5.cpp** file. The return type is void. The com(int n, int t) function prints out all strings with size t. Each char in the string is letter between 0 and 9. That means each letter has 10 choices. The generated string should be stored in array A. Of course, at any given time, array A can store at most one complete string.

The first parameter of the function com() indicates which letter the function is working on. The second parameter of the function com() indicates the length of a desired string.

The following texts are several execution examples,

```
jli$ ./a.out 1
0
1
2
3
4
5
6
7
8
9

jli$ ./a.out 2
00
10
```

```
20
30
40
50
60
70
80
90
01
11
21
31
41
51
…
```

In above examples, the argument of the command is the string length. The first example prints out ten strings, the second one prints out one hundred strings (partial results displayed).

Hints:

- If n < 1, print out the string stored in array A.
- If n >=1, uses a for loop to give different values to A[n-1], recursively call com(n-1, t).

Wrap up

When you're done, zip three files to lab5.zip

zip lab5.zip *

Submit lab5.zip to Canvas.

Make sure you logout before you leave!

If you cannot finish lab in class, please save all your files. Next time you login the computer in the lab, you can continue work on your files. Please save them before you logout. If you work in a Linux lab, please save the file to your machine. However, if you are working in the Mac lab, please save the file to a CLOUD. The Mac machine will erase everything you saved once you logout.