

## Welcome to Lab Quacks!

Course Website: <https://courses.engr.illinois.edu/cs225/fa2019>

### Overview

In this week's lab, you will refresh your memory in an important programming concept: recursion, and you will have a chance to practice with two new data structures we learned in lecture: stacks and queues.

### Recursion

Recursion refers to a style of writing functions where: **a function calls itself within its definition**. If you have already taken CS 125 or ECE 220 this will be a familiar topic for you. One advantage to writing functions recursively is that more often than not, it makes function definitions shorter, more elegant, and easier to follow for a human reader. However, recursion often times trades runtime efficiency for its sleek style; we must carefully decide when it is advantageous to use recursion. Always remember that while not every function can be written recursively, **every recursive function can be rewritten iteratively (using loops instead of recursion)**.

**Exercise 1.1:** The Fibonacci sequence start with 0 and 1:  $F_0 = 0$ ,  $F_1 = 1$ . The rule for the  $n$ th Fibonacci number is:  $F_n = F_{n-1} + F_{n-2}$ . The iterative function `iterativeFib()` has been provided for you. Complete the function `recursiveFib` to calculate the  $n$ th Fibonacci number recursively.

main.cpp

```

1  int recursiveFib(int n){
2      cout << "recursiveFib gets called!" << endl;
3
4  //YOUR CODE HERE
5      if (n == 0 || n == 1)
6          return n;
7      return recursiveFib(n-1)+recursiveFib(n-2);
8
9
10
11
12
13 }
```

```

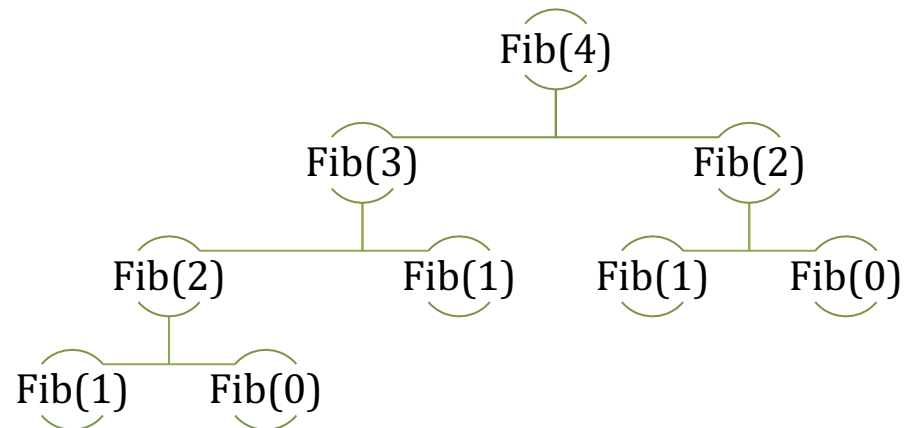
14 int iterativeFib(int n){
15     int n0 = 1; int n1 = 1; int result = 1;
16     for(int i = 2; i < n; i++) {
17         cout << "one iteration!" << endl;
18         result = n0 + n1; n0 = n1; n1 = result;
19     }
20     return result;
21 }
22
23 int main() {
24     int recur = recursiveFib(4);
25     int iter = iterativeFib(4);
26 }
```

### Exercise 1.2:

In the main function,  $F_4$  gets computed by the two different algorithms.

How many print statements will be called in the **recursive algorithm**? To better understand the number of recursion calls, draw the recursion tree of **recursiveFib(4)** in the space below.

9 print statements will be called in recursiveFib(4).



How many print statements will be called in the **iterative algorithm** to calculate the same result?

Iterative algorithm will print out 2 lines.

## Stack and Q

Stacks and queues are two of the most popular one-dimensional data structures in CS. Remember from lecture that elements in a queue follow the FIFO rule: First In First Out, while elements in a stack follow the LIFO rule: Last In First Out. In this lab we will use the STL's (Standard Template Library) `queue<T>` and `stack<T>` classes.

**Exercise 2.1:** Complete the definitions of the following two functions. `popLast()` takes in a queue by reference and pops out the last element of the queue (at the back of the queue), the remaining elements in the queue should maintain their initial order. `reverseQ()` takes in a queue by reference and reverses the order of its elements.

**Do not create new queues/stacks in your implementation!**  
(you can use the stack that is already give to you)

Useful stack and queue functions:

`queue.front()`   `stack.top()`

`push()`   `pop()`   `size()`

main.cpp

```
1  template <typename T>
2  void popLast(queue<T> &q) {
3      //YOUR CODE HERE
4      size_t s = q.size();
5      for (int i=0; i<s-1; i++){
6          T temp = q.front();
7          q.pop();
8          q.push(temp);
9      }
10     q.pop();
11 }
12 }
13 template <typename T>
14 void reverseQ(queue<T> &q) {
15     stack<T> s;
16     //YOUR CODE HERE
17     while (!q.empty()){
18         s.push(q.front());
19         q.pop();
20     }
21     while (!s.empty()){
22         q.push(s.top());
23         s.pop();
24     }
25 }
```

## Time Complexity

Recall that in computer science we use Big O notation to describe the runtimes of functions and programs. In Big O runtime analysis, **n** usually denotes the size of the arguments/objects/variables/data structures that the function or program manipulates. In the following exercise, you will analyze and compare the run time of Stack and Queue operations on elements in different positions.

**Exercise 3:** Looking at the main function below, write the time complexities (in big O) of **popping out** the following elements.

1 on **q**            **O(1)**  
1 on **s**            **O(n)**  
**n** on **q**           **O(n)**  
**n** on **s**            **O(1)**

main.cpp

```
1  int main() {
2      queue<int> q;
3      stack<int> s;
4
5      for(int i = 1; i <=n; i++) {
6          q.push(i);
7          s.push(i);
8      }
9  }
10 }
```

In the programming part of this lab, you will:

- Get familiar with Stacks and Queues
- Practice writing recursive functions
- Have fun solving clever Queue and Stack puzzle problems!

**As your TA and CAs, we're here to help with your programming for the rest of this lab section! ☺**