lab_quacks : Spiteful Stacks, Questionable Queues

Week #5 – September 26-28, 2018

Welcome to Lab Quacks!

Course Website: https://courses.engr.illinois.edu/cs225/labs

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; so 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 nth Fibonacci number is: $F_n=F_{n-1} + F_{n-2}$. Complete the function recursiveFib to calculate the nth Fibonacci number recursively, the iterative function iterativeFib() has been provided for you.

main.cpp

```
int recursiveFib(int n){
//YOUR CODE HERE



}
int iterativeFib(int n){
  int n1 = 1; int n2 = 1; int res = 1;
  for(int i = 2; i < n; i++) {
    res = n1 + n2; n2 = n1; n1 = res;
  }
  return res;
}
int main() {
  int recur = recursiveFib(5);
  int iter = iterativeFib(5);
}
```

Exercise 1.2: Fibonacci sequences are used to approximately calculate the Golden Ratio. The Golden Ratio is approximately F_n/F_{n-1} and this approximation gets better the bigger n gets. Suppose you can only afford 14 activations of recursiveFib() , what will your best approximation of the Golden Ratio be? Draw the activation diagram for the numerator of your best estimate: F_n= recursiveFib(n)

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. Using linked lists is one way to implement queues and stacks; however in this lab we will not be concerned about queue and stack implementation details, instead we will be using 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. In both functions you are not allowed to define any additional data structures other than those already given.

Useful stack and queue functions:

queue.front() **stack.top()**

push() **pop()** **size()**

main.cpp
```
template <typename T>
T popLast(queue<T> &q ){
  //YOUR CODE HERE









}
template <typename T>
void reverseQ(queue<T> &q){
  stack<T> s;
  //YOUR CODE HERE









}
```

Big O

In computer science we use Big O notation to describe the runtimes of functions and programs. For example, a function that takes in a stack and empties it by repeatedly popping off elements from the top, will run in **O(n)** time, where n = number of elements in the original stack. In Big O runtime analysis, n usually denotes the size of the arguments/objects/variables/data structures that the function or program manipulates.

Exercise 3.1: Looking at the main function below, write the time complexities of retrieving the following elements:

1 on q .... 1 on s ....

10 on q .... 10 on s ....

main.cpp
```
int main() {
    queue<int> q;
    stack<int> s;

    for(int i = 1; i <=10; i++) {
        q.push(i);
        s.push(i);
}
}
```

Exercise 3.2: Rank the following big O runtimes from fastest to slowest:
(1 = fastest, 7 = slowest):

O(n) .... O(2^n) .... O(log(n)) .... O(n*log(n)) ....

O(n!) .... O(1) .... O(n^100) ....

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! ☺