CS 367 - Introduction to Data Structures  
Week 8, 2017

Homework 6: available Wednesday due by 10pm on Friday
Program 2: quiz:
Program 3: assigned soon and due 3/19, FINISH IT BEFORE BREAK (3/17)

Last Week
  Exam, PriorityQueue, Java’s Comparable Interface, Heap Data Structure, insert, removeMax

This Week
  Java’s Stack, Queues, PriorityQueue
  Call Stack Tracing
  Recursion
  • recursion vs. iteration
  • rules of recursion
  • constructing recursive code
  • practice writing recursive code
  Exam 1 returned
  Recursion
  • more practice writing recursive code
  • complexity of recursive methods
  • practice analyzing complexity

Next Week
  Read: finish Recursion, Search
  Read: Trees
  Execution tree tracing
  Searching
  Categorizing ADTs Part 1
  General Trees
  • implementing
  • determining tree height
Java’s types: Stack, Queue, PriorityQueue, Deque

MAY NOT USE THESE TYPES ON PROGRAMMING ASSIGNMENTS, unless explicitly allowed.

In java.util package

**Class Stack<E>** (LIFO)
- boolean empty()
- E peek()
- E pop()
- E push( E item )

**Interface Queue<E>** (Can be FIFO)
- methods that throws exception on error
  - add( E item )
  - remove()
  - element()

- methods that return special value on error
  - offer(E item)
  - poll()
  - peek()

**class PriorityQueue<E>** (Not FIFO)
- order based on natural ordering via Comparable
- can use a Comparator

**class Deque<E>** (Double-ended queue)
- Deque<Integer> deque = new Deque<Integer>();
- deque.addFirst
deque.addLast
deque.removeFirst
deque.removeLast
deque.peekFirst
deque.peekLast
Call Stack Tracing - Displaying a Singly-Linked Chain of Nodes

Method Call:
print(head);

Iterative Implementation:
void print (Listnode<String> curr) {
while (curr != null) {
    System.out.println(curr.getData());
    curr = curr.getNext();
}
}

Recursive Implementation:
void print(Listnode<String> curr) {
    if (curr == null) return;
    System.out.println(curr.getData());
    print(curr.getNext());
}

How do these work?
Recursion vs. Iteration

Recursion is like iteration:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Recursion</th>
</tr>
</thead>
</table>

Recursion is NOT like iteration:

- Each loop iteration
- A loop with a bad stopping condition

Rules for Recursion

1.

2.
Recursion

What is it?

Why use it?

→ How would you modify the print method to display a singly-linked chain of nodes in reverse order?
Factorials: n!

Consider the factorial of n (assume n >=0):

\[ n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 2 \times 1 \]

\[ 6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720 \]

Method Call:

\[ \text{factorial}(6); \]

Iterative Implementation:

\[
\text{int factorial(int n) }
\{
\text{int result = 1;}
\text{for (; n > 1; n--)}
\{ \text{result = result * n;}
\text{return result;}
\}
\]

Recursive Definition:

\[ \text{Complete the Recursive Implementation:} \]

\[
\text{int factorial(int n) }
\{ \]
Constructing Recursive Code

→ Write a recursive method that computes $n^m$
that is, it computes double $n$ raised to an int power $m$?

recursive definition:

recursive implementation:

Key Questions:

1.

2.

3.

4.
→ Write a recursive method that displays the values in a (non-null) list of strings.

void display(ListADT<String> list) {
Making a Reversed Copy of a Chain of Nodes

Method Use:

```java
Listnode<String> reversed = reverseCopy(head, null);
```

Method Implementation:

```java
Listnode<String> reverseCopy
    (Listnode<String> curr, Listnode<String> rev) {
        if (curr == null) return rev;
        rev = new Listnode<String>(curr.getData(), rev);
        return reverseCopy(curr.getNext(), rev);
    }
```

How does it work?

```
head -> "a" -> "b" -> "c"
```
Practice – Array

Write a recursive method that counts the number of even values in an (non-null) array filled with integers.

1.

2.

3.

4.

```java
int evenCount(int[] array) {
```
Analyzing Complexity of Recursive Methods

Options:
1.
2.

Steps
1.

2.

3.

4.
Practice – Complexity of Recursive evenCount

Problem size $N$ is

1. Equations

2. Table

3. Verify

4. Complexity
Practice – Strings

→ Write a recursive method that determines if a string is a palindrome.

Examples:
  * eye
  * mom
  * radar
  * racecar
  * Rise to vote, sir!
  * Never odd or even!
  * A nut for a jar of tuna.
  * Campus Motto: Bottoms up, Mac.
  * Ed, I saw Harpo Marx ram Oprah W aside!

Assumptions: non-null input string, all spaces and punctuation removed, all lower-case

Useful string methods:
  * char charAt(int index)
  * int length()
  * String substring(int begin, int one_past_last)
Practice – Complexity of Recursive isPalindrone

Problem size N is

1. Equations

2. Table

3. Verify

4. Complexity
Towers of Hanoi

Algorithm

```c
solveTowers(count, src, dest, spare) {
```

Complexity

Problem size N is

1. Equations

2. Table

3 Verify

4. Complexity