

CS 102: Practice Test Problems for the Material after Test 2

SOLUTIONS

1. Consider G_n defined by

$$G_n = \begin{cases} 0 & \text{if } n = 0 \\ 3 & \text{if } n = 1 \\ 2 + G_{n-1}G_{n-2} & \text{if } n > 1 \end{cases}$$

- (a) Compute G_4 . Show your work.

Solution: $G_0 = 0$, $G_1 = 3$, $G_2 = 2 + G_1G_0 = 2 + 3(0) = 2$,
 $G_3 = 2 + G_2G_1 = 2 + 2(3) = 8$, $G_4 = 2 + G_3G_2 = 2 + 8(2) = 18$.

- (b) Write a recursive method whose return value is G_n .

Solution:

```
public static int G(int n){
    if (n==0) return 0;
    else if (n==1) return 3;
    else return 2 + G(n-1)*G(n-2);
}
```

- (c) Write an iterative method whose return value is G_n .

Solution:

```
public static int G(int n){
    if (n==0) return 0;
    else{
        int[] ar = new int[n+1];
        ar[0] = 0;
        ar[1] = 3;
        for (int i=2; i<=n; i++)
```

```
        ar[i] = 2 + ar[i-1]*ar[i-2];
    return ar[n];
    }
}
```

2. For `LStack` is an implementation of the `Stack` interface which stores a stack of integers, what is the output to the screen of the following code fragment?

```
Stack s = new LStack();
s.push(5);
s.push(10);
s.push(3);
System.out.println(s.pop());
int v = s.pop()*s.pop();
s.push(v);
s.push(13);
int w = s.pop();
int x = s.pop();
s.push(x-w);
System.out.println(s.pop());
```

Solution:

```
3
37
```

3. Write a method

```
public void resize()
```

which takes an array `ar` of `Objects` (implemented as global instance data), replaces `ar` with an array of twice the size of the original `ar`, and copies all the original values of `ar` into the first slots of the new `ar`.

Solution:

```

public void resize(){
    Object[] temp = new Object[2*ar.length];
    for (int i=0; i<ar.length; i++)
        temp[i] = ar[i];
    ar = temp;
}

```

4. What is the output to the screen for the following Towers of Hanoi program? Note the main method is inside the TowersOfHanoi class, but the rest of the code is the same as we discussed in class.

```

public class TowersOfHanoi{
    private int totalDisks;
    public static void main(String[] args){
        TowersOfHanoi towers = new TowersOfHanoi(3);
        towers.solve();
    }
    public TowersOfHanoi (int disks){ totalDisks = disks; }
    public void solve(){
        moveTower(totalDisks,1,3,2);
    }
    private void moveTower(int numDisks, int start, int end, int temp){
        if (numDisks == 1) moveOneDisk(start,end);
        else{
            moveTower(numDisks-1, start, temp, end);
            moveOneDisk(start, end);
            moveTower(numDisks-1, temp, end, start);
        }
    }
    private void moveOneDisk(int start, int end){
        System.out.println("Move one disk from " + start + " to " + end);
    }
}

```

Solution:

Move one disk from 1 to 3

Move one disk from 1 to 2

```
Move one disk from 3 to 2
Move one disk from 1 to 3
Move one disk from 2 to 1
Move one disk from 2 to 3
Move one disk from 1 to 3
```

5. Consider the function $f(x) = e^x - x - 2$. Note that $f(0) = -1$ and $f(2) = e^2 - 3 > 0$. Assume $f(x)$ is implemented in a method

```
public double f(double x){ return Math.exp(x) - x - 2; }
```

Implement a binary search algorithm which returns a root of $f(x)$ in the interval $(0, 2)$, within a tolerance of $1E-10$.

Solution:

```
public class ComputeRoot{
    private static final double TOLERANCE = 1E-10;
    public static void main (String[] args){
        System.out.println("The root of f is " + root(0,2));
    }
    public static double root(double min, double max){
        double mid = (min + max)/2;
        if (max-mid<TOLERANCE)
            return mid;
        else if (f(mid)<0)
            return root(mid, max);
        else
            return root(min, mid);
    }
    public static double f(double x){
        return Math.exp(x) - x - 2;
    }
}
```

6. Recall that a queue has two basic operations, **enqueue** and **dequeue**. Assume q is a queue which holds `Objects`. Write code which sends the `Object` at the front q to the end of q .

Solution:

```
q.enqueue(q.dequeue());
```

7. Fill in the following code for a linked list of integers:

```
public class MyList{
    private Node head;

    public MyList(){ // constructs an empty list: fill in code:

    }

    public void insertAtHead(int n){
        // inserts new node containing n at head of list: fill in code:

    }

    public void insertAtIndex(int index, int n){
        // inserts new node at index i: fill in code:

    }

    private class Node{
        public int num;
        public Node next;
        public Node(int n){
            num = n;
            next = null;
        }
    }
}
```

```
    }  
}
```

Solution:

```
public MyList(){  
    head = null;  
}  
public void insertAtHead(int n){  
    Node newNode = new Node(n);  
    newNode.next = head;  
    head = newNode;  
}  
public void insertAtIndex(int index, int n){  
    if (index==0)  
        insertAtHead(n);  
    else{  
        prev = head;  
        curr = head.next;  
        for (int i=1; i<index; i++){  
            prev = curr;  
            curr = curr.next;  
        }  
        Node newNode = new Node(n);  
        prev.next = newNode;  
        newNode.next = curr;  
    }  
}
```

8. Consider Postfix.java as discussed in class:

```
import java.util.Scanner;  
public class Postfix{  
    public static void main(String[] args){  
        Scanner scan = new Scanner(System.in);  
        System.out.println("Enter a postfix expression:");  
        String st = scan.nextLine();  
        Stack s = new LStack();
```

```

Scanner stringScan = new Scanner(st);
while (stringScan.hasNext()){
    String token = stringScan.next();
    if (token.equals("+") || token.equals("-")
        || token.equals("*") || token.equals("/")
        || token.equals("%")){
        //binary operation code
        int operand2 = s.pop();
        int operand1 = s.pop();
        int val;
        if (token.equals("+"))
            val = operand1 + operand2;
        else if (token.equals("-"))
            val = operand1 - operand2;
        else if (token.equals("*"))
            val = operand1 * operand2;
        else if (token.equals("/"))
            val = operand1 / operand2;
        else // token is "%"
            val = operand1 % operand2;
        s.push(val);
    }
    else{ // assume number
        int val = Integer.parseInt(token);
        s.push(val);
    } // end else
} // end while
System.out.println("The evaluation of this postfix"
    + " expression is " + s.pop());
} // end main
} // end Postfix class

```

Trace through the value of the stack if the following postfix string expression is entered by the user:

"16 3 - 7 10 5 / + 2 + %"

What is the evaluation of this postfix expression?

Solution:

token: 16 s: 16

token: 3 s: 3
 16

token: - s: 13

token: 7 s: 7
 13

token: 10 s: 10
 7
 13

token: 5 s: 5
 10
 7
 13

token: / s: 2
 7
 13

token: + s: 9
 13

token: % s: 4

The value of the postfix expression is then given by the remaining item on the stack, 4.

9. List all the elements (from head to tail) of an initially empty queue `q` of integers after the following code is performed:

```
q.enqueue(7);  
q.enqueue(9);
```

```
q.enqueue(4);  
q.enqueue(q.dequeue() + q.dequeue());  
q.enqueue(15);  
q.dequeue();  
q.enqueue(12);
```

Solution:

16 15 12