Instructions: This homework assignment covers some of the later course topics: Recursion, Stacks, Queues and Algorithm Analysis. The answers may be determined by reading the CS 1706 course notes and experimenting with code.

The quizzes will be submitted on-line through the Computer Science Curator server. No late submissions will be accepted.

Consider the following non-member function to display a List object (see the Java API List interface):

```java
// Prints the data elements contained in a List object,
// assumes that the List objects implements a toString() member function.
//
// Pre:  the List object is properly constructed;
// Post: any data elements in the LinkList object have been
//       printed to standard output, in order of occurrence in
//       the list
//
void public PrintList(List list) {   // line 1
   if (______________) return;   // line 2: stop
   PrintIt(list.iterator());    // line 3: start recurse
}

// Prints the data elements contained in a List object,
// assuming that the List objects implements a toString() member function.
//
// Pre:  the Iterator pointer is at the Head of the List prior to
//       the first call;
// Post: any data elements in the List object have been
//       printed to standard output, in order of occurrence in
//       the list
//
void private PrintIt(Iterator it) {   // line 4
   if (______________) return;   // line 5: stop recurse
   Object obj = ____________;   // line 6: advance
   System.out.println( obj.toString() );  // line 7: print data
   PrintIt(it);      // line 8: recurse
}
```

1. In order for the helper method to be setup/invoked properly, how should the blank in line 2 be filled?

   1) list.isEmpty()
   2) ! list.isEmpty()
   3) hasNext()
   4) ! hasNext()
   5) list.lastIndex()
   6) None of these

2. In order for the recursion to stop properly, how should the blank in line 5 be filled?

   1) list.isEmpty()
   2) ! list.isEmpty()
3. In order for the recursion to operate properly, how should the blank in line 6 be filled?

1) list.size()
2) list.iterator()
3) it.hasNext()
4) it.next()
5) list.nextIndex()
6) None of these

4. What would happen if the statement in line 5 were omitted from the function definition?

1) The code would no longer compile.
2) The code would compile but it would not execute.
3) The function would fall into an infinite recursion unless the list was initially empty.
4) The function would fall into an infinite recursion no matter whether the list was empty or not.
5) The function would perform correctly no matter whether the list was empty or not.
6) None of these, (execution would stop due to an exception).

5. The above method PrintIt() is an example of what form of recursion?

1) Head Recursion.
2) Tail Recursion.
3) Middle Decomposition Recursion.
4) Center and Edges Decomposition Recursion.
5) Backtracking Recursion.
6) None of these
For the next two questions, consider the following recursive power function definition:

```java
// Returns value of x raised to the power exp.
// Pre: x is a positive real number
//      exp is a nonnegative exponent
public double Power1(double x, int exp) {
    if (exp < 0 || x <= 0.0)  // logical error
        return -1.0;
    if (exp == 0)
        return 1.0;
    if (exp == 1)
        return x;
    return ( x * Power1(x, exp - 1) );
}
```

Suppose the function is called as: `Power1(3.0, 19);`

6. How many multiplications will be performed?

1) 1   4) 12   7) 19   10) None of these
2) 5   5) 15   8) 20
3) 10   6) 18   9) 0

7. How many function calls (including the first) will be performed?

1) 1   4) 12   7) 19   10) None of these
2) 5   5) 15   8) 20
3) 10   6) 18   9) 0
For questions 8 through 10, we consider a non-recursive version of the `intComma()` function discussed in the course notes. The recursion is eliminated by making use of an internal Stack object, (see Java Stack API).

```java
public void intComma(int num) {
    Stack parts = new Stack();
    if (num < 0) {
        System.out.print("-");
        num = -num;
    }
    while (num >= 1000) {
        String out = "";
        int tail = num % 1000; // Line 1
        if (tail < 10)
            out = "00";
        else if (tail < 100)
            out = "0";
        out = out.concat(new Integer(tail).toString()); // Line 2
        num = num / 1000;
    }
    System.out.print( num );
    Object next = null;
    while ( ! parts.empty() ) {
        System.out.print( "," + next ); // Line 3
    }
}
```

**intComma(-45002130) must print:**

-45,002,130

8. How should the blank in Line 1 be filled?

1) `num / 1000`
2) `num % 1000`
3) `num - 1000`
4) `num`
5) None of these

9. How should the blank in Line 2 be filled?

1) `System.out.print(parts.peek());`
2) `System.out.print( parts.pop() );`
3) `parts.push(out)`
4) `parts.pop()`

10. How should the blank in Line 3 be filled?

1) `next = parts.peek()`
2) `next = parts.pop()`
3) `next = parts.push()`
4) `next = parts.search()`
5) None of these