Topic: Recursion

Activity Guidelines

Group Size: 3

Method of Assigning Students: Count the number of students in the class, divide by 3, count off from 1 to the quotient, and group identical numbers.

Materials:

- ✓ Handout (one copy per group) with questions to be answered at the end of the session

Roles:

Coordinator/Leader: Clarifies goals and objectives, allocates roles for each team member and divides the tasks within the group.

Monitor/Evaluator: Person designed to evaluate the different ideas to approach the problem and make an accurate judgment of the most beneficial option.

Implementer: Person in charge to transform discussions and ideas into a technical solution for the given problem.

Individual Accountability: Each team member gets assigned a specific role in order to ensure every student within a team participates and contributes to reach a solution for each problem presented in the activity.

Activity Summary

Students are assigned 4 problems to be solved both iteratively and recursively. By finding a solution to the problem iteratively, students will learn to transform iterative solutions into recursive methods.

a) Create a method to calculate the factorial of n number.

b) Create a method that outputs the Fibonacci numbers up to n number.

c) Create a method to print the elements on a Linked List.

d) Create a method to search for an element on a Linked List.

CNS#1042341

ELEMENTARY DATA STRUCTURES

PEER SESSION

Recursion

1. Create a method ***factorial*(int n)** that returns an integer containing n's factorial (that is n x n-1 x … x 3 x 2 x 1)

   a. Using an iterative method. (with loops)

```
public static long factorial(int num) {
        long result = 1;
        if(num == 0) {
            return 1;
        }
        else {
            for(int i = 2; i <= num; i++) {
                result *= i;
            }
            return result;
        }
}
```

   b. Using a recursive method. (without loops)

```
public static long factorial(long number) {
    if (number <= 1) // test for base case
      return 1; // base cases: 0! = 1 and 1! = 1
    else
      // recursion step
      return number * factorial(number - 1);
  }
}
```

2. Create a recursive method fibonacci(int n) giving you Fn that is the nth Fibonacci number, note that Fn=Fn-1+Fn-2 , F1=1 and F2=1.

   a. Iterative method

```
public int fibonacci(int n)  {
    if(n == 0)
        return 0;
    else if(n == 1)
      return 1;
   else
      return fibonacci(n - 1) + fibonacci(n - 2);
}
```

b. Recursive method

```
public int fibonacci(int n) {
        int x = 0, y = 1, z = 1;
        for (int i = 0; i < n; i++) {
            x = y;
            y = z;
            z = x + y;
        }
        return x;
    }
```

3. Create a method to print out all the elements contained in a Linked List.

a. Iterative method

```
public String toString() {
        Node current = head.getNext();
        String output = "";
        while (current != null) {
            output += current.getData().toString() + " ";
            current = current.getNext();
        }
        return output;
    }
```

b. Recursive method

```
 public void printRecursive(Node head) {
      if (head!=null)
        System.out.println(head.getData().toString() + " ");
      if(head.getNext()!=null)
        printRecursive(head.getNext());
      else
        return;
 }
```

4. Create a method to search for an element in a Linked List.

a. Iterative method

```
public boolean search(int element) {
      Node current = head.getNext();
      String e = Integer.toString(element);
        while (current != null) {
          if(current.getData().toString().equals(e))
            return true;
          current = current.getNext();
```

```
            }
            return false;
        }
```

b. Recursive method

```
public void searchRecursively(Node head, int element) {
      String e = Integer.toString(element);
      if (head!=null) {
        if(head.getData().toString().equals(e)) {
          System.out.println("The element is in the list");
          return;
        }
      }
      if(head.getNext()!=null){
        searchRecursively(head.getNext(), element);
      }
      else  {
        System.out.println("The element is not in the list");
        return;
      }
    }
```