PEER-LED TEAM LEARNING
COMPUTER SCIENCE

MEETING 11 – STUDENT VERSION
PRACTICE WITH RECURSION

BARBARA G. RYDER AND PRADIP HARI

Key steps in recursive algorithms:

1. **Need a base case to end the recursion**
2. **Need to decompose the problem into smaller problems of the 'same shape'**
3. **Need to solve smaller problems recursively**
4. **Need to compose the solutions of the smaller problems into a solution for the larger problem**

Exercise 1. Choosing teams for a tug-of-war competition at a picnic.
i. Assume there are 4 players who have to be divided into 2 teams. How many different possible teams are there? Hint: think of first choosing 1 player and putting her on the team; then there are 3 players left from which to choose the other player for this team. Otherwise, you can choose 1 player and NOT put her on the team; then there are 3 players left from which you have to choose the other 2 players for the team.

Draw a tree of the possibilities, much akin to the game tree for Nim3 you used in Exercise 1.

ii. What we did in part i. was to explore ways of choosing 2 players from a group of 4. Now think about how many different ways there are to choose k players from n players (*choose(n,k)*). Derive a formula for this function.

iii. Now write the Java code for the method *choose(n,k)*.

Exercise 2. Playing a game

This week we will learn a new game called **NIM3.** Initially, there is a pile of stones (for us, Hershey's kisses) in the middle of the table. Players sit in a circle around the pile. Each player must pick up 1,2, or 3 stones as they take their turn. The loser is the person who picks up the last stone.

i. Split into pairs and play several rounds of the NIM3 game. Think about the strategies you are using to try to win this game.

ii. Can you suggest the objects and operations needed to have a Java program 'play' the game? (i.e., you versus a computer simulation of the other player)? How would you use these operations to 'play' the game in the simulation?

iii. Now think about your strategies for winning. Think about how you try to choose a good move when it is your turn to play. Assume both players do play to win. Starting with 5 kisses, what are the ways the game can proceed? Pretend the two players are Me and You. Here's a picture of one possible game:

Me	**You**
Remove 2 leaving 3	
	Remove 2 leaving 1
Remove 1, Whoops, I lose!	

Your peer leader will show you how to represent these moves as a tree of possible games. Continue to explore ways that the game can succeed, until you are sure there are no other possibilities left, thus completing the tree. Can you see a way to ensure winning at NIM3? Think about how a computer could simulate being your opponent in NIM3. What would the computer have to know?

iv. Now we can associate a Win or Loss with each Nim board, depending on whether the current player (whose turn it is) can ensure a Win for herself or is forced to Lose no matter what move she chooses. Label the nodes in the game tree with Win or Loss from the perspective of the player whose turn it is at each level of the tree.

Imagine that you are the current player and you can see where you are in the game tree. Think of an algorithm that describes the strategy you should use to evaluate your next move. Now think about how to use recursion to program the strategy you just devised. Produce a flowchart for your *move()* method.

v. Putting it all together. Below is an outline of the **NimState** class. List the set of methods (including those above) you will need to simulate the play of Nim3 with a program. Assign pairs of students in the group to write the Java code for each of these methods. Enter them into the computer and watch the analysis of the game, or watch our NIM3 program play the game.

```
//NimState-10.java program, CS111, OUTLINE
//
import java.io.*; //allows access to builtin io package of
                  //classes in Java
class NimState extends Object{
      // instance variable of every NimState object
      private int count;  //count only to be used within
                          // the NimState class
public NimState(int cnt) {//constructor function
   setcount(cnt);
}
 public int getcount()
{  return count;
}
public void setcount(int s)
{  count = s;
}
….
public static void main (String[ ] arg) {
      //create new game object initialized with i stones
   for (int i=10; i>0; i--)
   {  NimState st = new NimState(i);
   System.out.print (" For a pile of ");
   System.out.print (i);
   System.out.print( " stones, first player can ");
   //test if game is winnable by first player
      if(st.win()) System.out.println ("win, remove " + st.move());
      else System.out.println("lose, remove " + st.move());
   }
   //show a game being played
   NimState st = new NimState(10);//initialize the board
   int k,play;
   play=1;
   while ((st.getcount())>0) //note last move always gets pile to 0
      { k = st.move();
      System.out.print ("player ");
      System.out.print (play);
      System.out.print (" removes ");
      System.out.print(k);
      System.out.println( " stones ");
      st.setcount((st.getcount())-k);
      if (play == 1) play=2;
      else play=1;
      }

}

   //NimState-10.java program, CS111, complete program
```

```
//
import java.io.*; //allows access to builtin io package of
                  //classes in Java
class NimState extends Object{
      // instance variable of every NimState object
      private int count;  //count only to be used within
                          // the NimState class
public NimState(int cnt) {//constructor function
   setcount(cnt);
}

   public int getcount()
   {
      return count;
   }
   public void setcount(int s)
   {
      count = s;
   }


// private: for use within NimState only; used for auxiliary
// fcns not available to user of class NimState
private NimState removeOne(){
// creates new NimState object dynamically at run-time
      return new NimState(count-1);
}

private NimState removeTwo(){
// creates new NimState object dynamically at run-time
      return new NimState(count-2);
}

private NimState removeThree(){
//creates new NimState object at runtime
   return new NimState(count-3);
}


// function checks if current player can win
// game from current pile of count stones
// and returns true, if so, else returns false
public boolean win() {
      switch (count) {
      case 1:     return false; //must lose
      case 2:  return true;// remove 1 and force opponent to lose
      case 3:  return true; //remove 2 and force opponent to lose
//plays game out by trying both possible next moves by
```

```
//opposing player; continues until can see if game is a win
//for original player
            default: if( ! (removeOne()).win() ) return true;
                  else if( !  (removeTwo()).win() ) return true;
                  else if ( ! (removeThree()).win()) return true;
                  else return false;
      }
}


//chooses move for current player to accomplish her win
public int move() {
      switch (count) {
            case 1:  return 1;//moving player loses
            case 2:  return 1;//moving player wins
            case 3:  return 2;//moving player wins
            default:
      // if she removes 1, can opponent win? if not, remove 1
                  if (!  (removeOne()).win() ) return 1;
      // if she removes 2, can opponent win? if not, remove 2
            else if (!  (removeTwo()).win() ) return 2;
      // if she removes 3 can opponent win? if not, remove 3
            else if (! (removeThree()).win() ) return 3;
      // otherwise just remove 1 and continue game
            else return 1;
      } }
public static void main (String[ ] arg) {
      //create new game object initialized with i stones
   for (int i=10; i>0; i--)
   {  NimState st = new NimState(i);
   System.out.print (" For a pile of ");
   System.out.print (i);
   System.out.print( " stones, first player can ");
   //test if game is winnable by first player
      if(st.win()) System.out.println ("win, remove " + st.move());
      else System.out.println("lose, remove " + st.move());
   }
   //show a game being played
   NimState st = new NimState(10);//initialize the board
   int k,play;
   play=1;
   while ((st.getcount())>0) //note last move always gets pile to 0
      { k = st.move();
      System.out.print ("player ");
      System.out.print (play);
      System.out.print (" removes ");
      System.out.print(k);
      System.out.println( " stones ");
```

```
    st.setcount((st.getcount())-k);
    if (play == 1) play=2;
    else play=1;
    }  }
}
```

Cite this module as: Ryder, B.G., Hari, P. (2012). Peer-Led Team Learning Computer Science: Meeting 11 - Student Version; Practice with Recursion. Online at http://www.pltlis.org. Originally published in *Progressions: The Peer-Led Team Learning Project Newsletter*, Volume 9, Number 1, Fall 2007.