Matthew A. Krieger

CS 254

Homework #3

10/27/2015

**Problem 1**

***Classes***

All four classes (.java files) have been submitted along with this document.

**TautologyTester -** *Takes a logical expression in as a String, places it into a queue in infix form, and uses the Logic class to determine if it's a tautology*

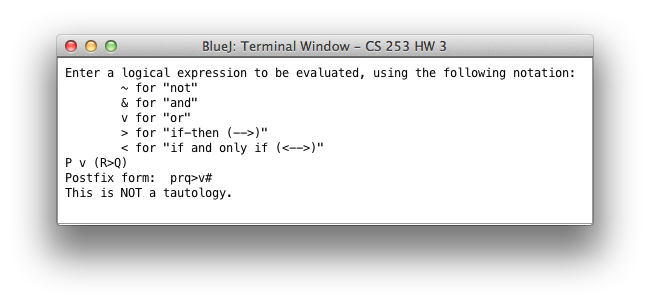
**Logic -** *Converts logical expressions from infix form to postfix form and determines if they are tautologies*

**LLStack -** *Linked list implementation of a stack with nodes that may contain char or boolean data*

**LLQueue -** *Linked list implementation of a queue with nodes that may contain char or boolean data*

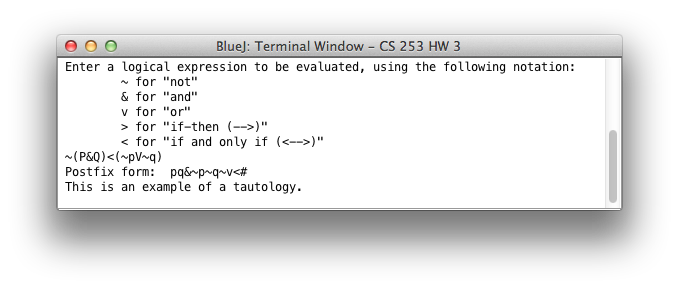
***Test cases***

***Case 1 (not a tautology****)*

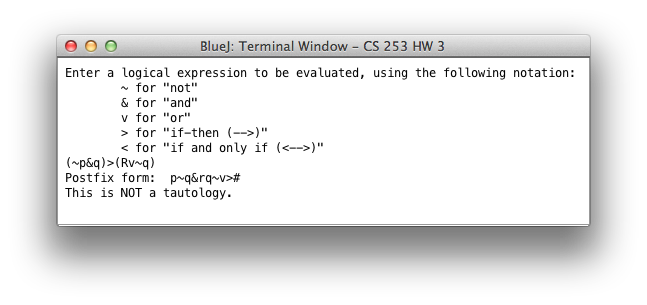
*P v (R 🡪 Q)*

***Case 2 (tautology)***

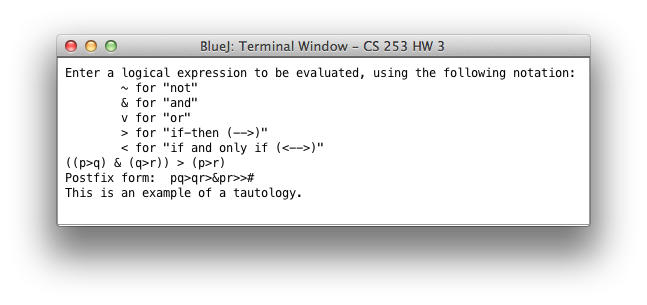
*¬(P & Q) 🡨🡪 (¬P v ¬Q)*

******

***Case 3 (not a tautology)***

****** *(¬P & Q) 🡪 (R v ¬Q)*

***Case4 (tautology)***

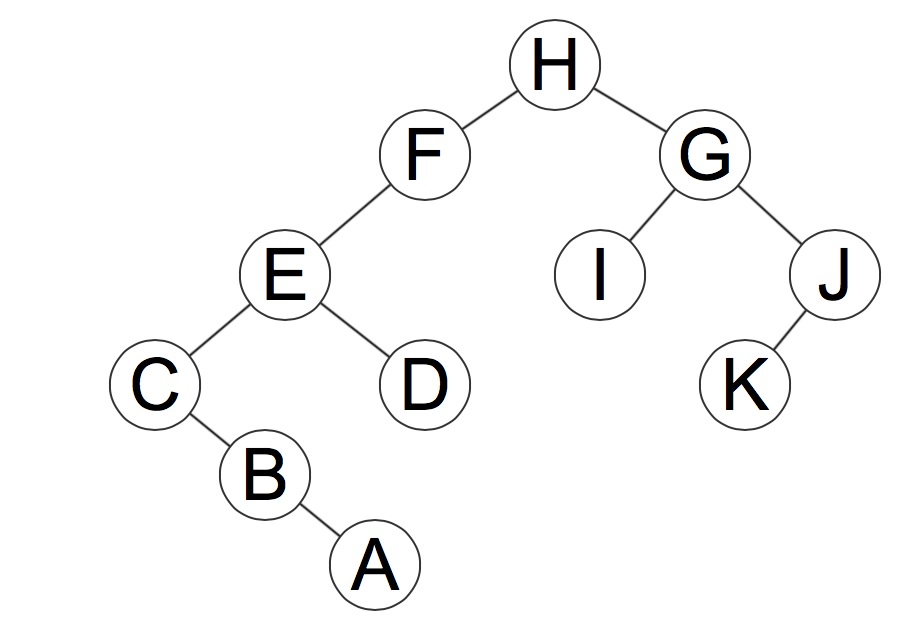
****** *( (P🡪Q) & (Q 🡪 R) ) 🡪 (P 🡪 R)*

**Problem 2**

***Traversals of the Binary Tree (given in hw):***

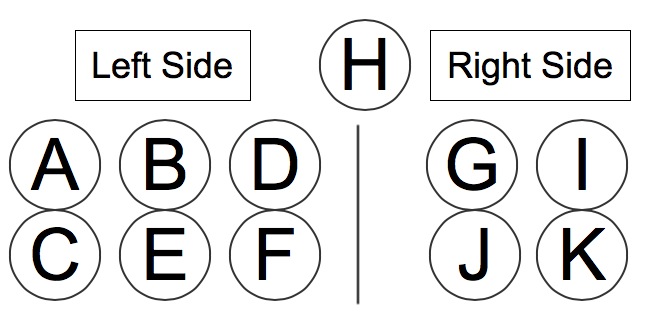
***Postorder:*** A B C D E F I K J G H

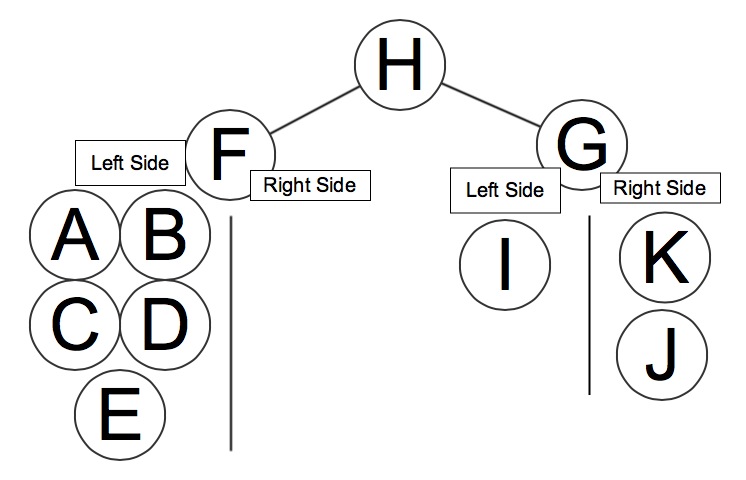
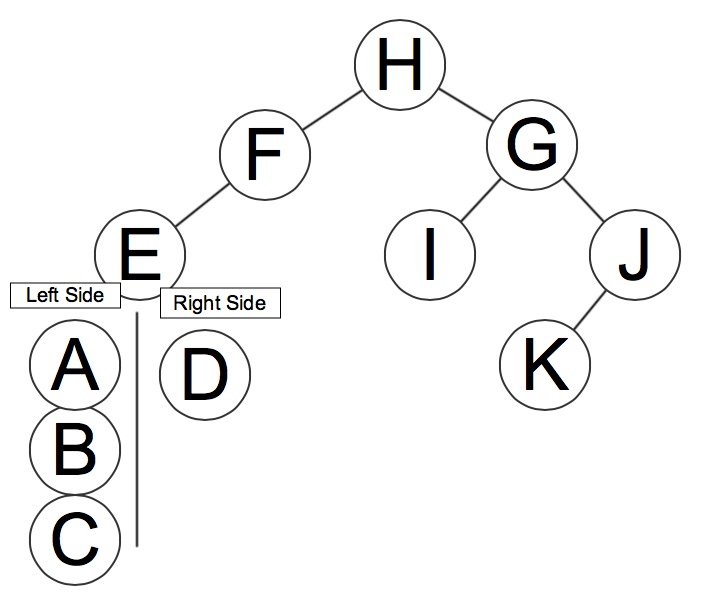
***Inorder:*** C B A E D F H I G K J

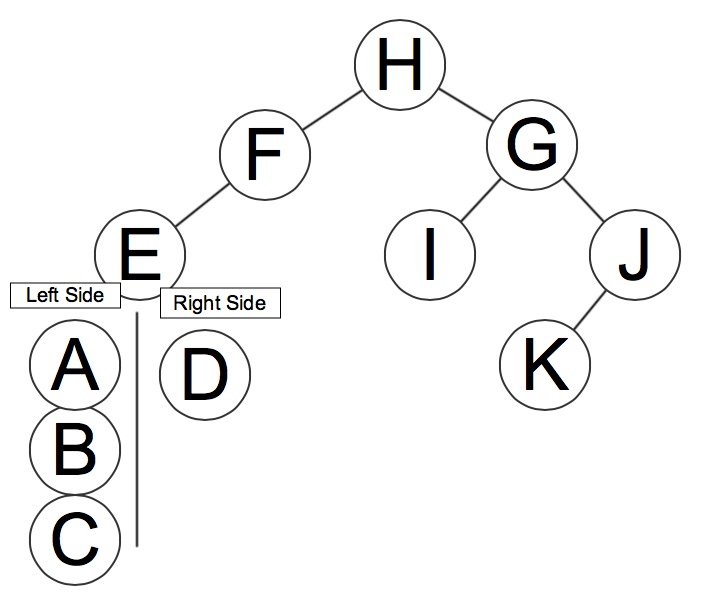
***Answer:***

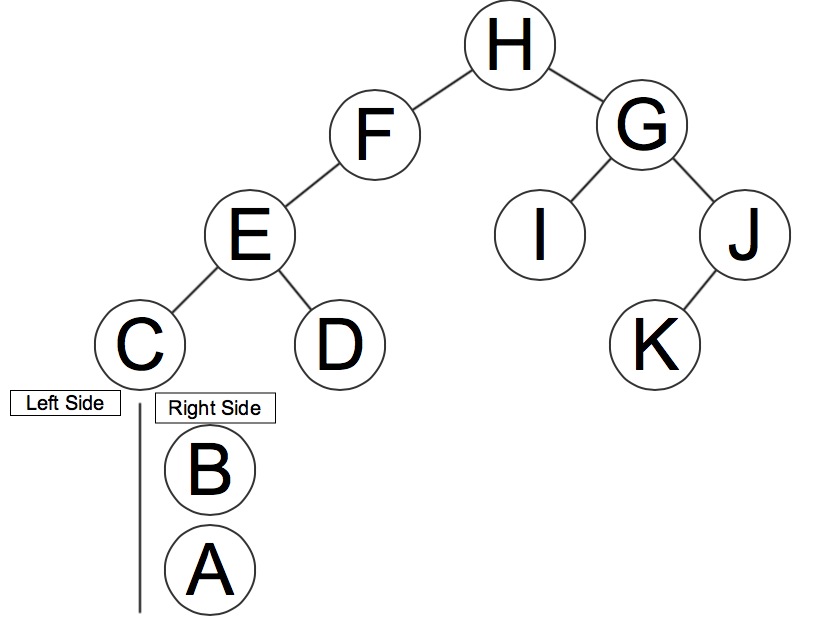
***Step-by-step:***

1. First, I looked at the last value in the postorder traversal (as the root is the last node to be visited). Thus, I determined that H was the root node. This also allowed me to determine that A, B, C, D, E & F were on the left side of the root and G, I, J, & K were on the right side from the inorder traversal.



1. Next, I looked at which nodes came last for each side in the postorder traversal to determine the two children of the root. These were revealed to be F (left child) and G (right child). By looking at the inorder traversal, I was then able to determine that A, B, C, D & E were all to the left of F and I was the left of G while J and k were to its right.
2. Since I was the only node on the left side of G, that meant it was the left child of G; looking at the postorder traversal, the node that came just before G was J, indicating that it is the right child of G. Examining the inorder traversal revealed that K was the to left of J (and thus its left child) and nothing was to its right (making it the furthest right node, which matches up with the sequence of inorder traversals as J is the last node in the inorder traversal – the position held by the right-most node). This also completed the right branch of the binary tree.
3. I then shifted my attention to the left branch, whose “root” was F. The other nodes in the left branch were all shown to be to the left of F (as determined in step 2), so I looked to see which node came just before F in the postorder traversal. This ended up being E. Thus, E was the left child of F. Examining the inorder traversal revealed that A, B, & C were to the left of E and D was to its right. (Corresponding figure is on next page)



1. Since D was the only node on the right side of E, it meant that D was the right child of E. Therefore, only the left child of E needed to be determined; this would be the node that came just before D in the postorder traversal – C in this case. Examining the inorder traversal revealed that A, & B were to the right of C, with nothing to its left, thus making it the left-most node (this matches up with the sequence of inorder traversals as C is first node in the inorder traversal – the position held by the left-most node).
2. Finally, only the positions of B & A remained to be found. From step 5, it was determined that they were both to the right of C. The postorder traversal showed that B came just before C, meaning that B was the right child of C (since C had no left child). An examination of the inorder traversal revealed that A was to the right of B with nothing to its left; therefore, A was the right child of B. Thus, completing the left branch and the binary tree. (Completed tree can be seen on page 3 under **Answer** section)

TautologyTester.java

|  |
| --- |
| import java.util.\*;  public class TautologyTester  {  public static void main (String[] args)  {  boolean tautology;  LLQueue infix, postfix;  Scanner scan = new Scanner(System.in);  String input;  infix = new LLQueue();  postfix = new LLQueue();    System.out.println ("Enter a logical expression to be evaluated, using the following notation:\n\t~ for \"not\"\n\t& for \"and\"\n\t" +  "v for \"or\"\n\t> for \"if-then (-->)\"\n\t< for \"if and only if (<-->)\"");  input = scan.nextLine().toLowerCase();    for (int i=0; i<input.length(); i++)  {  char item = input.charAt(i);  if (item != ' ')  infix.enqueue(item);  }  infix.enqueue('#');    postfix = Logic.postfix(infix);  System.out.print("Postfix form: ");  postfix.printQueue();    tautology = Logic.isTautology(postfix);    if (tautology)  System.out.println("\nThis is an example of a tautology.");  else  System.out.println("\nThis is NOT a tautology.");  }  } |

Logic.java

|  |
| --- |
| public class Logic  {  //gives priority of each logical operator  public static int priority(char item)  {  if (item == '<')  return 1;    if (item == '>')  return 2;    if (item == '&')  return 4;    if (item == 'v')  return 3;    if (item == '~')  return 5;    if (item == '(')  return 6;    return 0;  }    //converts logical expression in infix form to postfix form  public static LLQueue postfix(LLQueue input)  {  LLQueue output = new LLQueue();  LLStack operators = new LLStack();  char item;  int size = input.size();  operators.push('#');  for (int i = 0; i<size; i++)  {  item = input.dequeueChar();    if (item == 'p' || item == 'q' || item == 'r')  output.enqueue(item);  else if (item == '#')  {  int stackSize = operators.size();  for (int j = 0; j<stackSize; j++)  output.enqueue(operators.popChar());  }  else if (item == ')' )  {  while(operators.onTopChar() != '(')  output.enqueue(operators.popChar());  operators.popChar();  }  else if (priority(item) > priority(operators.onTopChar()) || operators.onTopChar() =='(')  operators.push(item);  else  {  while(priority(item)<=priority(operators.onTopChar()) && operators.onTopChar() !='(')  output.enqueue(operators.popChar());  operators.push(item);  }  }  return output;  }    //returns true if postfix expression is a tautology and false otherwise  public static boolean isTautology(LLQueue postfixInput)  {  boolean test1, test2, test3, test4, test5, test6, test7, test8;  boolean v1, v2;  char item;  int size = postfixInput.size();  LLQueue temp = new LLQueue();  LLStack value = new LLStack();    for (int i = 0; i<size; i++)  {  item = postfixInput.dequeueChar();  temp.enqueue(item);    if (item == 'p' || item == 'q' || item == 'r')  value.push(true);  else if (item == '~')  {  v1 = value.popBoolean();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push((v1 && v2) || (!v1 && !v2));  }  }  test1 = value.popBoolean();    if(test1)  {  for (int i = 0; i<size; i++)  {  item = temp.dequeueChar();  postfixInput.enqueue(item);    if (item == 'p' || item == 'q')  value.push(true);  else if (item == 'r')  value.push(false);  else if (item == '~')  {  v1 = value.popBoolean();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test2 = value.popBoolean();    if(test2)  {  for (int i = 0; i<size; i++)  {  item = postfixInput.dequeueChar();  temp.enqueue(item);    if (item == 'p' || item == 'r')  value.push(true);  else if (item == 'q')  value.push(false);  else if (item == '~')  {  v1 = value.popBoolean();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test3 = value.popBoolean();    if(test3)  {  for (int i = 0; i<size; i++)  {  item = temp.dequeueChar();  postfixInput.enqueue(item);    if (item == 'p')  value.push(true);  else if (item == 'r' || item == 'q')  value.push(false);  else if (item == '~')  {  v1 = value.popBoolean();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test4 = value.popBoolean();    if(test4)  {  for (int i = 0; i<size; i++)  {  item = postfixInput.dequeueChar();  temp.enqueue(item);    if (item == 'r' || item == 'q')  value.push(true);  else if (item == 'p')  value.push(false);  else if (item == '~')  {  v1 = value.popBoolean();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test5 = value.popBoolean();    if(test5)  {  for (int i = 0; i<size; i++)  {  item = temp.dequeueChar();  postfixInput.enqueue(item);    if (item == 'p' || item == 'r')  value.push(false);  else if (item == 'q')  value.push(true);  else if (item == '~')  {  v1 = value.popBoolean();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test6 = value.popBoolean();    if(test6)  {  for (int i = 0; i<size; i++)  {  item = postfixInput.dequeueChar();  temp.enqueue(item);    if (item == 'p' || item == 'q')  value.push(false);  else if (item == 'r')  value.push(true);  else if (item == '~')  {  v1 = value.popBoolean();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test7 = value.popBoolean();    if(test7)  {  for (int i = 0; i<size; i++)  {  item = temp.dequeueChar();  postfixInput.enqueue(item);    if (item == 'p' || item == 'q' || item == 'r')  value.push(false);  else if (item == '~')  {  v1 = value.popBoolean();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBoolean();  v2 = value.popBoolean();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;    return true;  }  } |

LLStack.java

|  |
| --- |
| public class LLStack  {  private int size;  private Node top;    public LLStack()  {  size = 0;  top = null;  }  public boolean empty()  {  return (top == null);  }    public int size()  {  return size;  }    public void push (char item)  {  Node newNode = new Node (item);  newNode.setNext(top);  top = newNode;  size++;  }    public void push (boolean value)  {  Node newNode = new Node (value);  newNode.setNext(top);  top = newNode;  size++;  }    public void push (char item, boolean value)  {  Node newNode = new Node (item, value, top);  top = newNode;  size++;  }    public char popChar ()  {  char item;  item = top.getData();  top = top.getNext();  size--;  return item;  }    public boolean popBoolean ()  {  boolean value;  value = top.getValue();  top = top.getNext();  size--;  return value;  }    public char onTopChar ()  {  return top.getData();  }    public boolean onTopBoolean ()  {  return top.getValue();  }    private class Node  {  private char data;  private boolean truthValue;  private Node next;    public Node()  {  this('0', true, null);  }    public Node(char c)  {  data = c;  }    public Node(boolean value)  {  truthValue = value;  }    public Node(char c, boolean value)  {  data = c;  truthValue = value;  next = null;  }    public Node(char c, boolean value, Node n)  {  data = c;  truthValue = value;  next = n;  }    public void setData (char newData)  {  data = newData;  }    public void setValue (boolean newValue)  {  truthValue = newValue;  }    public void setNext (Node newNext)  {  next = newNext;  }    public char getData()  {  return data;  }    public boolean getValue()  {  return truthValue;  }    public Node getNext()  {  return next;  }    public void displayNode()  {  System.out.println (data);  }  }  } |

LLQueue.java

|  |
| --- |
| public class LLQueue  {  private int size;  private Node front;  private Node rear;    public LLQueue()  {  size = 0;  front = null;  rear = null;  }  public boolean empty()  {  return (size == 0);  }    public int size()  {  return size;  }    public void enqueue (char item)  {  Node newNode = new Node(item);  if(this.empty())  front = newNode;  else  rear.setNext(newNode);  rear = newNode;  size++;  }    public void enqueue (boolean value)  {  Node newNode = new Node(value);  if(this.empty())  front = newNode;  else  rear.setNext(newNode);  rear = newNode;  size++;  }    public void enqueue (char item, boolean value)  {  Node newNode = new Node(item, value, null);  if(this.empty())  front = newNode;  else  rear.setNext(newNode);  rear = newNode;  size++;  }    public char dequeueChar ()  {  char item;  item = front.getData();  front = front.getNext();  size--;  if (this.empty())  rear = null;  return item;  }    public boolean dequeueBoolean ()  {  boolean value;  value = front.getValue();  front = front.getNext();  size--;  if (this.empty())  rear = null;  return value;  }    public char frontChar()  {  return front.getData();  }    public boolean frontBoolean()  {  return front.getValue();  }    public void printQueue()  {  if (empty())  System.out.print("The queue is empty.");  else  {  Node temp = front;  for (int i = 0; i<size(); i++)  {  temp.displayNode();  temp = temp.getNext();  }  }  }    private class Node  {  private char data;  private boolean truthValue;  private Node next;    public Node()  {  this('0', true, null);  }    public Node(char c)  {  data = c;  next = null;  }    public Node(boolean value)  {  truthValue = value;  next = null;  }    public Node(char c, boolean value)  {  data = c;  truthValue = value;  next = null;  }    public Node(char c, boolean value, Node n)  {  data = c;  truthValue = value;  next = n;  }    public void setData (char newData)  {  data = newData;  }    public void setValue (boolean newValue)  {  truthValue = newValue;  }    public void setNext (Node newNext)  {  next = newNext;  }    public char getData()  {  return data;  }    public boolean getValue()  {  return truthValue;  }    public Node getNext()  {  return next;  }    public void displayNode()  {  System.out.print (data);  }  }  } |