Matthew A. Krieger

November 08, 2015

CS 253: Data & File Structures

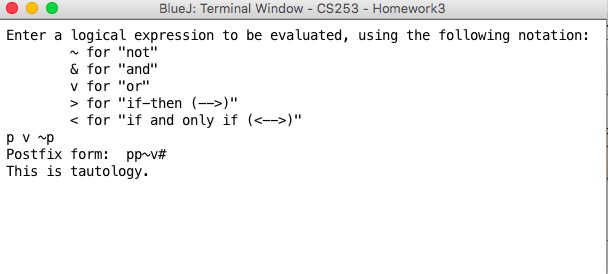
Homework III

Neli P. Zlatareva, Ph.D.

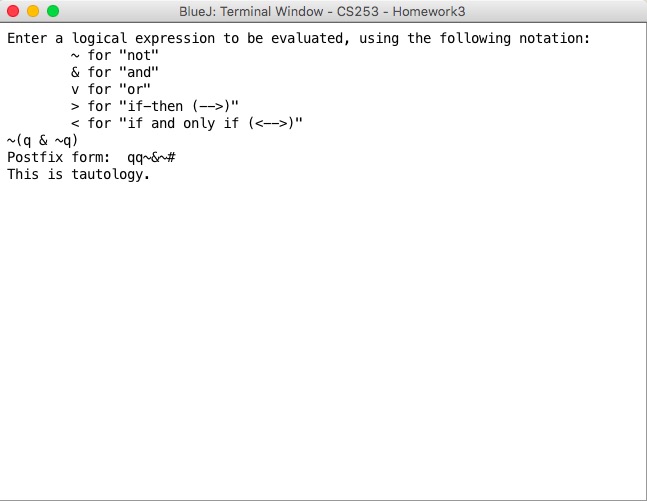
Homework III

**Problem I:**

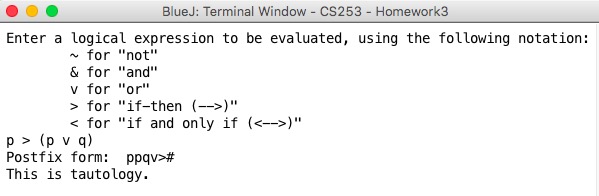
**Example of Tautology I:**

****

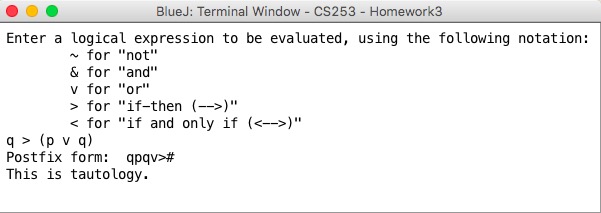
**Example of Tautology II:**

****

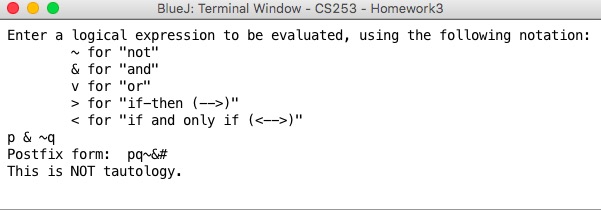
**Example of Tautology III:**

****

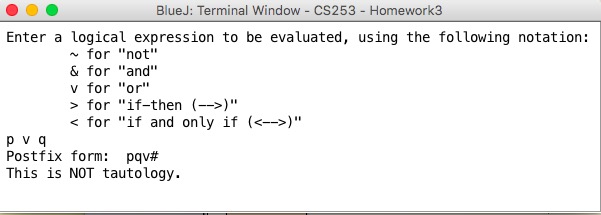
**Example of Tautology IV:**

****

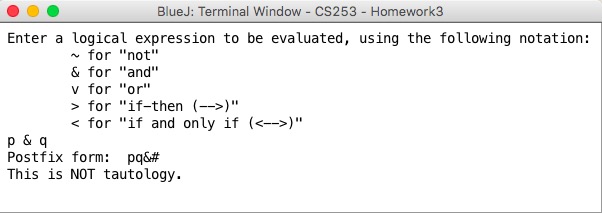
**Example of NOT Tautology I:**

****

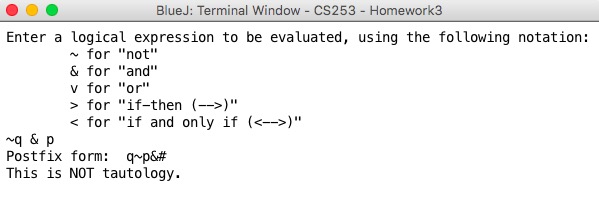
**Example of NOT Tautology II:**

****

**Example of NOT Tautology III:**

****

**Example of NOT Tautology IV:**

****

**Source Code:**

**LLQueue:**

|  |
| --- |
| /\* Created By: Matthew Krieger  \* Date: October 25, 2015  \* Assignment: Homework III  \*/  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);  }  }  } |

**LLStack:**

|  |
| --- |
| /\* Created By: Matthew Krieger  \* Date: October 25, 2015  \* Assignment: Homework III  \*/  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 popBool ()  {  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);  }    }  } |

**LTester:**

|  |
| --- |
| /\* Created By: Matthew Krieger  \* Date: October 25, 2015  \* Assignment: Homework III  \*/  public class LTester  {    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;  }      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;  }      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.popBool();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBool();  v2 = value.popBool();  value.push((v1 && v2) || (!v1 && !v2));  }  }  test1 = value.popBool();    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.popBool();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBool();  v2 = value.popBool();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test2 = value.popBool();    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.popBool();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBool();  v2 = value.popBool();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test3 = value.popBool();    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.popBool();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBool();  v2 = value.popBool();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test4 = value.popBool();    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.popBool();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBool();  v2 = value.popBool();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test5 = value.popBool();    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.popBool();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBool();  v2 = value.popBool();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test6 = value.popBool();    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.popBool();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBool();  v2 = value.popBool();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;  test7 = value.popBool();    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.popBool();  value.push(!v1);  }  else if (item == 'v')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1||v2);  }  else if (item == '&')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(v1&&v2);  }  else if(item == '>')  {  v1 = value.popBool();  v2 = value.popBool();  value.push(!v2 || v1);  }  else if (item == '<')  {  v1 = value.popBool();  v2 = value.popBool();  value.push((v1 && v2) || (!v1 && !v2));  }  }  }  else  return false;    return true;  }  } |

**TTester:**

|  |
| --- |
| /\* Created By: Matthew Krieger  \* Date: October 25, 2015  \* Assignment: Homework III  \*/  import java.util.\*;  public class TTester  {  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 = LTester.postfix(infix);    System.out.print("Postfix form: ");  postfix.printQueue();    tautology = LTester.isTautology(postfix);    if (tautology)  System.out.println("\nThis is tautology.");  else  System.out.println("\nThis is NOT tautology.");  }  } |

**Problem II:**

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

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

Recursive & Systematic Algorithm:

1. Identify & mark the last element in the Postorder result 🡪 **this is the root element**

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

1. Identify & mark the root element in the inorder result

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

1. The root element will divide the inorder result into left sub tree & right sub tree. 🡪 Identify & Label the left sub tree & right sub tree

Left Sub Tree: C B A E D F

Right Sub Tree: I G K J

1. Apply the Inorder & PostOrder results on the left sub tree to the algorithm above. Identify & mark the last element in the Postorder result 🡪 **this is the root element**.

PostOrder: A B C D E F

1. Identify & mark the root element in the inorder result

Inorder: C B A E D F

1. The root element will divide the inorder result into left sub tree & right sub tree. 🡪 Identify & Label the left sub tree & right sub tree

Left Sub Tree: C B A E D

Right Sub Tree: N/A

1. Apply the Inorder & PostOrder results to the child of ‘F’ to the algorithm above. Identify & mark the last element in the Postorder result 🡪 **this is the root element**.

PostOrder: A B C D E

1. Identify & mark the root element in the inorder result

Inorder: C B A E D

1. The root element will divide the inorder result into left sub tree & right sub tree. 🡪 Identify & Label the left sub tree & right sub tree

Left Sub Tree: CBA

Right Sub Tree: D

1. Apply the Inorder & PostOrder results to the left child of ‘E’ to the algorithm above. Identify & mark the last element in the Postorder result 🡪 **this is the root element**.

PostOrder: A B C

1. Identify & mark the root element in the inorder result

Inorder: C B A

1. The root element will divide the inorder result into left sub tree & right sub tree. 🡪 Identify & Label the left sub tree & right sub tree

Left Sub Tree: N/A

Right Sub Tree: B A

1. Apply the Inorder & PostOrder results to the child of ‘C’ to the algorithm above. Identify & mark the last element in the Postorder result 🡪 **this is the root element**.

PostOrder: A B

1. Identify & mark the root element in the inorder result

Inorder: B A

1. The root element will divide the inorder result into left sub tree & right sub tree. 🡪 Identify & Label the left sub tree & right sub tree

Left Sub Tree: N/A

Right Sub Tree: A

1. Apply the Inorder & PostOrder results to the child of ‘C’ to the algorithm above. Identify & mark the last element in the Postorder result 🡪 **this is the root element**.

PostOrder: I K J G

1. Identify & mark the root element in the inorder result

Inorder: I G K J

1. The root element will divide the inorder result into left sub tree & right sub tree. 🡪 Identify & Label the left sub tree & right sub tree

Left Sub Tree: I

Right Sub Tree: K J

1. Apply the Inorder & PostOrder results to the right child of ‘G’ to the algorithm above. Identify & mark the last element in the Postorder result 🡪 **this is the root element**.

PostOrder: K J

1. Identify & mark the root element in the inorder result

InOrder: K J

1. The root element will divide the inorder result into left sub tree & right sub tree. 🡪 Identify & Label the left sub tree & right sub tree

Left Sub Tree: K

Right Sub Tree: N/A

Binary Tree – Complete