CSCE 310 Data Structures & Algorithms

Fall 2004
Steve Goddard

Homework 2, September 21, 2004
(Total of 100 points)

Chapter 3 and Logic

Due: 9:00pm Tuesday, October 5, 2004

Exercises (50 points):

1. (3 pts) Complete the following truth table.

a	b	c	$a \otimes b$	$a \rightarrow b$	$(\neg a) \leftrightarrow b$	$(a \wedge b) \vee c$	$(a \otimes b) \rightarrow (\neg c)$
0	0	0					
1	0	0					
0	1	0					
1	1	0					
0	0	1					
1	0	1					
0	1	1					
1	1	1					

Consider the following expression for 2 and 3: $\neg(p \wedge (r \vee s)) \vee (\neg p \vee \neg q)$

2. (3 pts) Construct a truth table for this expression.

p	q	r	s				

3. (7 pts) Reduce this expression as much as possible, showing your work and stating the reason for each step of your reduction.

4. (7 pts) Show that $[(p \vee q) \wedge (p \rightarrow r) \wedge (q \rightarrow r)] \rightarrow r$ is a tautology. Do this both by giving equivalences and by completing a truth table.

5. (5 pts) Rewrite the expression $\neg \forall x(\exists y \forall z\ P(x, y, z) \wedge \exists z \forall y\ P(x, y, z))$ so that no negations appear to the left of a quantifier.

6. (5 pts) Solve the following recurrence relations
   a. $T(n) = 2T(n/2) + n^3$
   b. $T(n) = T(n-1) + n$
   c. $T(n) = T(n/2) + T(n/4) + T(n/8) + n^2$

7. (4 pts each) Complete the following exercise from the textbook.
   2.4 #6
   3.1 #1
   3.2 #6
   3.3 #5
   3.4 #1

Programming Exercise (50 points):

For this assignment you will be making a data structure to hold information from the Bill Of Materials (BOM) file. This file contains a list of components followed by the subcomponents that are required to create them. For example, consider this sample input file SampleBom (you can find another online sample at http://www.cse.unl.edu/~goddard/Courses/CSCE310J/Project/BOMwithNames.csv):

```
Wagon: Nut 3, Bolt (1/2") 3, Box 1,Front Wheel Assembly 1, Rear Wheel Assembly 1;
Scooter: Nut 4, Platform 1, Front Wheel Assembly 1, Rear Wheel Assembly 1;
Box: Sheet Aluminum 1, Radio Flyer Emblem 2, AWT Emblem 1;
Front Wheel Assembly: Bolt (1") 1, Nut 1, Front Assembly Bracket 1;
Rear Wheel Assembly: Bolt (1") 1, Nut 1, Rear Assembly Bracket 1;
```

Each line begins with the component being described. Following it is a list of subcomponents that come later in the file which are required to make that component. Each subcomponent name is followed by a space, and then the quantity of that subcomponent required. The whole list is then terminated by a semicolon. For the purposes of this assignment, you may ignore the quantity. However, keep in mind that for your MRP project you will have to use the quantity, so you may want to store the quantity for future use.

Your job is to create a forest of trees that hold all of the top-level components and their respective subcomponent trees. You should write a data structure which holds the roots of all of the top level components (in this example, the Wagon and the Scooter). Then, you will add all of the subcomponents as child items of the respective tree(s). Note that in this example, the Front Wheel Assembly is used by 2 products, so you will have to link that node in with both trees.

To test your forest structure, you will write a program named bomtree which *recursively* prints out a preorder traversal of any component (including a subcomponent). The program will be invoked by passing the name of the BOM file as the first argument, and the name of the component to print the tree for as the second argument. The output should be the traversal printed out on a single line (unless, of course, the line is longer than 80 characters, in that case the terminal will automatically wrap. The point is you should not manually insert new lines). See the following example for the exact output your program should generate for the above example file.

```
prompt > bomtree SampleBom Scooter
Scooter: (Nut, Platform, Front Wheel Assembly: (Bolt (1"), Nut, Front Assembly Bracket), Rear
Wheel Assembly: (Bolt (1"), Nut, Rear Assembly Bracket))
prompt >
```

Note that components that have subcomponents should be followed by a colon, while components without subcomponents should not. Each subcomponent list should also be enclosed in parenthesis, even if there is only 1 subcomponent.

Here are some things that your forest and trees need to do to work correctly:

1. You will need to consider the implications of order in the file. For example, both the Wagon and the Scooter use the Nut part, but the Nut subcomponent doesn't need to be created again when the Scooter line is parsed. You should therefore design a mechanism to see if a subcomponent already exists to be linked to elsewhere in the forest.
2. Each component can have any number of subcomponents, so you must program for a dynamic tree with flexibility for adding more subcomponents as the file is parsed.

You may assume that there are no circular dependencies inside the file (i.e. there are no components that have themselves in a subcomponent list somewhere). You may also assume that the components are listed in order of use, so if you don't find a component already in your forest while parsing the beginning of a line, that component must be a top-level component and should be a new root of a tree.

Analysis: Prepare a report in which the following are considered.

- Is this an efficient method for storing the BOM? Why?
- Compute the worst-case asymptotic complexity of the insertion, searching, and preorder traversal functions of the tree.
- Critically analyze the structure of your tree. If you stored the number of subcomponents required, indicate if you think it is an appropriate storage method. If you did not store the number of subcomponents required, analyze the steps that would be needed to add them to your tree.

This analysis report should also (as in previous assignments) describe how your program works, what design choices (if any) were made and why, and any known defects. You may also discuss enhancements you think should be made, but were beyond the scope of the assignment (i.e., inadequacy of the specifications).

Deliverables:
You should turn in your analysis document, all code, and a makefile to build your program into the web-handin. You should also submit a hard copy of your analysis to me. I should be able to type `make` to make your programs.

This problem will be scored at follows:

Program correctness (as defined)	30% (15 pts)
Quality of design/readability	16% (10 pts)
In-line documentation/coding standard	20% (8 pts)
Design and analysis document	24% (12 pts)
Thoroughness of test cases	10% (5 pts)