CMPSC 460 – Principles of Programming Languages

Assignment 3 - Spring 2017

Total Points: 100

Due Date: March 17th, 2017

Question 1: (30 points)

Given below is the BNF representation of a binary tree that stores integers. We use any empty list to represent an empty binary tree. On the other hand, a nonempty binary tree can be represented as a list of nested lists, where each list has three parts: an integer node, a left child and a right child.

```
<emptytree> ::= []
<bintree> ::= <emptytree>
<bintree> ::= [<int> <bintree> <bintree>]
```

a) Write the recursive rule *btree*, which is a predicate of arity 1: `btree(T)`. This predicate takes a tree in a list representation and checks whether the given tree represents a binary tree or not.
b) Write the recursive rule *full*, which is a predicate of arity 1: `full(T)`. This predicate takes a tree in a list representation and checks whether the given tree represents a full binary tree.
c) Write the recursive rule *height*, which is a predicate of arity 2: `height(T, H)`. This predicates takes a tree in a list representation and determines the height of the given binary tree.
d) Write the recursive rule *nodes*, which is a predicate of arity 2: `nodes(T, N)`. This predicate takes a tree in a list representation and determines the number of nodes of the given binary tree.
e) Write the recursive rule *preorder*, which is a predicate of arity 2: `preorder(T, P)`. This predicate takes a tree in a list representation and builds a list that represents the preorder traversal of the given binary tree.
f) Write the recursive rule *inorder*, which is a predicate of arity 2: `inorder(T, I)`. This predicate takes a tree in a list representation and builds a list that represents the inorder traversal of the given binary tree.

For Example:

```
?- btree([1, [2,[],[]], [3,[],[]]]).
yes.
```

```
?- full([2, [3, [], []], []]).
no.

?- nodes([1, [2,[],[]], [3,[],[]]], N).
N = 3.

?- inorder([1, [2,[],[]], [3,[],[]]], I).
I = [2, 1, 3].
```

Question 2: (10 points)

Write a Prolog program called *ntoe*, which is a predicate of arity 2: `ntoe(Integer, List)`. This predicate will convert an integer number into its text version. For example:

```
?- ntoe(12345, X).
X = [one, two, three, four, five].

?- ntoe(12, [one, two]).
yes.

?- ntoe(-123, N).
N = [negative, one, two, three].
```

Also provide another predicate called *eton*, which is a predicate of arity 2: `eton(Integer, List)`. This predicate will transform a list of text into an integer number. For example:

```
?- eton(X, [one, two, three]).
X = 123.
```

Question 3: (10 points)

Write a Prolog program called *countconsonants*, which is a predicate of arity 2: `countconsonants (atom, Integer)`. This predicate will count the number of consonants in a word. For example:

```
?- countconsonants(that, C).
C = 3.
```

Question 4: (10 points)

Write a rule *count*, which is a predicate of arity 3: `count(Lb, Ub, C)`. This predicate will count all the numbers starting from Lb to Ub. We assume that Lb and Ub are integers from -9 to 9. The value of C will be the text version of the counter. For example:

```
?- count(0, 3, C).
C = zero ?;
C = one ?;
C = two ?;
```

```
C = three ?;
no

?- count(-3, 1, C).
C = [negative,three] ?;
C = [negative,two] ?;
C = [negative,one] ?;
C = zero ?;
C = one ?;
no
```

Question 5: (40 points)

1. A binary tree is either empty or a tree with a number as an intermediate node and a left and right subtrees. Write a procedural representation in Scheme of the binary tree. Since trees are represented procedurally, then an empty tree cannot be '(). You have to implement the following functions:
   - *consEmpty:* constructs an empty tree
   - *consTree*: constructs a non-empty tree
   - *empty?:* checks whether a binary tree is empty or not
   - *sum*: returns the sum of all the interior nodes
   - *left*: returns the left subtree
   - *right*: returns the right subtree
   - *preorder*: returns a list that represents the preorder traversal of a tree. Each contents of a tree is represented using a list.
   - *postorder*: returns a list that represents the postorder traversal of a tree. Each contents of a tree is represented using a list.

   Here is an example of how we can use this binary tree representation:

```
(define t (consEmpty))
(define t2 (consTree 10 (consTree 15 t t) t))
(t2 'empty?) ==> #f
(t2 'sum) ==> 25
(t2 'preorder) ==> '(10 (15 () ()) ())
((t2 'left) 'preorder) ==>'(15 () ())
((t2 'right)'preorder) ==> '()
```

Useful built-in Prolog predicates:

- The predicate `integer(X)`, returns true if X is an integer.
- The == operator can be used for integer equality.
- The predicate `atom_chars(Atom, List)`, transforms an atom into a list of one-character atoms.
- The predicate `name(Atom,List)`. One way to use this predicate is to split up an atom into a list of ASCII characters. Another usage is to convert a list of ASCII characters back into an atom:

```
?- name(1234, List).
List = [49, 50, 51, 52].

?- name(Number, [55, 50, 56, 52, 48]).
Number = 72840.
```

Submission:

You should submit the following:

- The four Prolog programs and a single Scheme file
- Enough test cases and screenshots of the results to demonstrate that your programs are working properly.