Practice Problem Set – 3

Trees

1. We want to represent UNIX directories in the form of trees. For this purpose, we have defined the following structures:

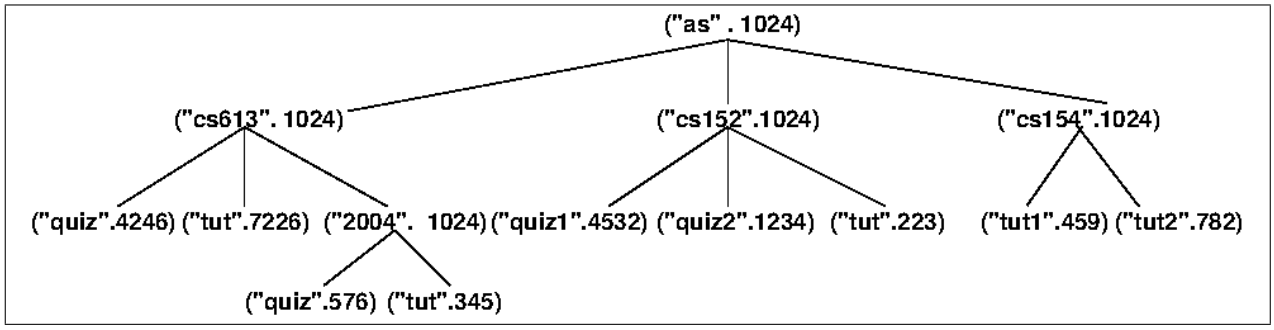
```
(struct dir (info fdlist) #:transparent)
(struct file (info contents) #:transparent)
```

Here

(a) `info` is a cons-ed pair `(name.size)`, where `name` is a string representing the name of the file or the directory and `size` is a n integer representing its size.

(b) `fdlist` is a list of files and directories and contents is a string representing the contents of the file.

(c) `pathname` is a list of strings. For example, `("as" "cs613" "2004")` is an example of a pathname.



(a) Write function `findtree` which takes a directory tree and a pathname and returns the subdirectory represented by the pathname.

(b) Define a function `ls` which takes a directory and a pathname an d lists the names of all the files and directories in the sub-directory represented by the pathname. The files under a directory should be listed immediately after the directory is listed.

(c) Define a function called `size` which takes a directory tree and a pathname, and returns the size of the entire directory tree represented by the pathname.

(d) Define a function called delete which takes a directory tree and a pathname and returns the tree with the subdirectory or the file represented by the pathname deleted.

To keep things simple, assume that the file or the directory represented by the pathname exists in the directory tree. You can use the following example:

```
(define thistree
  (dir
   (cons "as" 1024)
   (list (dir
          (cons "cs613" 1024)
          (list (file (cons "quiz" 4246) "junk")
                (file (cons "tut" 7226) "junk")
                (dir (cons "2004" 1024)
                     (list (file (cons "quiz" 576) "junk")
                           (file (cons "tut" 345) "junk")))))
         (dir (cons "cs152" 1024)
              (list (file (cons "quiz1" 4532) "junk")
                    (file (cons "quiz2" 1234) "junk")
                    (file (cons "tut" 1223) "junk")))
         (dir (cons "cs154" 1024)
              (list (file (cons "tut1" 459) "junk")
                    (file (cons "tut2" 782) "junk"))))))
```

2. We shall represent a binary list (a list of 0's and 1's) using a binary tree in the following way. If the list consists of all 1's or 0's, it is represented as a leaf node with the value 1 or 0. Otherwise the list is divided into two equal sublists and represented as a tree with each sub-tree being a representation of a sublist. Assume that the length of all the lists is the same, and this length, `len`, is some power of 2.

   (a) Define appropriate structures to represent binary lists.

   (b) Define a function called `tobintree` which will take a binary list and convert it into a binary tree.

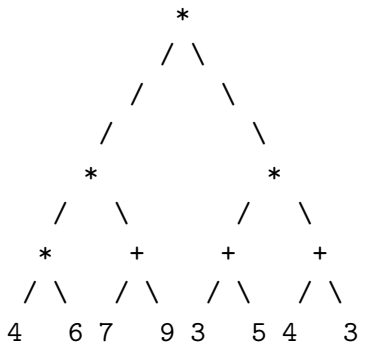
   (c) Define functions `bitwise-or` and `bitwise-and` which will take the binary tree representation of two binary lists and return a binary tree representing the result of the corresponding operation performed bit-wise.

   (d) Finally, define a function `(value? i bt)` which will take as input an integer `i` representing an index in a binary list and a binary tree representation of the list, `bt`, and determine whether the value at the index is 0 or 1.

3. Assume that we have defined expression trees using the `structs`:

   ```
   (struct node(op ltree rtree) #:transparent)
   (struct leaf (num) #:transparent)
   ```

   Write a function `convert` which will take an expression represented as a tree and converts it into string of parenthesized expression. Assume that the only operators are `*` and `+`. As an example, if `convert` is called on the tree shown below, we get `(4 * 6 * ( 7 + 9 ) + ( 3 + 5 ) * ( 4 + 3))` *Note that the parenthesized expression should use minimum number of parenthesis.*

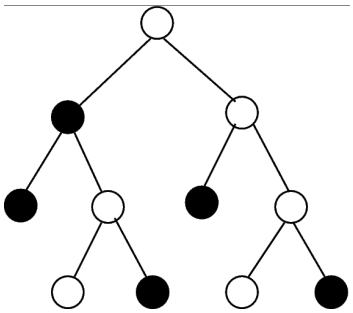


In other words, if `convert` is called on the tree shown above, we get `(4 * 6 * ( 7 + 9 ) * ( 3 + 5 ) * ( 4 + 3))`.

4. A radix tree is defined by the following `define-struct`

```
(define-struct node(color ltree rtree))
(define-struct leaf(color))
```

A radix tree represents a set of binary strings. Given a binary string, a radix tree is traversed by starting from the root and going down the tree in the following manner. At any node, if the binary string starts with a zero, then we go down the left subtree, else we go down the right subtree. The first bit then discarded and the same step is repeated. The binary string is contained in the tree, if and only if the traversal ends in a white node or a white leaf. As an example, the radix tree shown below represents the following strings apart from the empty string: 1, 01, 11, 010, 110.



(a) Write a function `insert` which will take a radix tree and and a binary string and insert the binary string in the tree. Assume that binary strings are represented as lists of `0`'s and `1`'s and colors are `'white` and `'black`.

(b) Write a function `delete` which will take a radix tree and a binary string and delete the string from the tree. (15 Marks).

5. Consider a general tree defined by the following struct:

```
(struct gnode  (val  lst) #:transparent)
```

A node of a general tree is at level n if the path from the root to the node has length $n-1$. The root node is at level 1. Write a function `(atlevel t n)` to collect all nodes at a given level n in a tree t. If n is greater than the height of the tree, the `atlevel` returns the null list.

6. Let us call a binary tree symmetric if its right subtree is the mirror image of its left subtree. Write a function `(all-sym-trees n)` which will generate a list of all symmetric trees which have exactly `n` interior nodes. Assume that `n` is odd.

   In this question, we shall only be interested in the structure of the tree. Therefore use the following `structs` to represent your trees.

```
(struct node (ltree rtree) #:transparent)
(struct leaf () #:transparent)
```

7. Consider the structures defined by:

```
(define-structure bnode ltree rtree)
(define-structure leaf val)
```

   A has written the following functions based on the above structure, and plans to write many more similar functions

```
(define (leaves tr)
  (if (leaf? tr) (list (leaf-val tr))
      (append (leaves (bnode-ltree tr))
              (leaves (bnode-rtree tr)))))

(define (height tr)
  (if (leaf? tr) 1
      (+ 1 (max (height (bnode-ltree tr))
                (height (bnode-rtree tr))))))
```

   Obviously the student has not done CS152. She does not how to abstract through higher-order functions, nor has she seen functions like foldr. What would you advise her to do?

8. Consider the expression `f(g(x,l), h(y), z)` written in a C-language like notation. A natural representation for this expression is in terms of a general tree, defined as:

   `(define-structure gnode val list)`

   Using such trees, the above expression would be represented as:

```
(gnode 'f
   ((gnode 'g (gnode 'x ())
              (gnode 'l ()))
    (gnode 'h (gnode 'y ()))
    (gnode 'z ())))
```

The same expression is written in Scheme as `(f (g x l) (h y) z)`. This can be represented in terms of a binary tree, defined as:

```
(define-structure bnode ltree rtree)
(define-structure leaf val)
```

Using this, `(f (g x l) (h y) z)` is written as:

```
(bnode
  (bnode
    (bnode (leaf 'f)
           (bnode (bnode (leaf 'g)
                         (leaf 'x))
                  (leaf 'l)))
    (bnode (leaf 'h) (leaf 'y)))
  (leaf 'z))
```

(a) Write a function c-to-scheme which will convert a C-style expression to an equivalent Scheme-style expression.

(b) Write a function scheme-to-c which will convert a Scheme-style expression to an equivalent C-style expression.

(c) Consider trees defined using the `structs`:

```
(struct node (ltree rtree) #:transparent)
(struct nultree () #:transparent)
```

Define a function `all-trees n` which returns all possible trees having `n` nodes.