**CSCI 520.001**

**Programming Assignment #8**

Turn in your work in the drop box for Assignment 8 (for Week 8) in eCollege 520.001 course by 11:59pm on Thursday, Nov. 6

Write a C++ program that creates a tree from given input, and prints the current tree using level-order traversal. Implement the following steps:

The following is how we define tree nodes:

struct node

{ int item; node \*l, \*r;

node(int x)

{ item = x; l = 0; r = 0; }

};

typedef node\* link;

The trees for this assignment have the following properties:

1. Each node has zero, one, or two children (left child and right child);
2. Each node has a unique key. That, the item in each node is different from items in all other nodes.
3. Each input item is also unique.

You may find the following function useful. This function visits nodes in a tree in level-order starting at the given root h (but please note that you only want to print the currently visited node: you may need to change this code for your purposes):

void traverse(link h)

{ QUEUE q(100);

q.put(h);

while (!q.empty())

{

h = q.get(); cout << h ;

if (h->l != 0) q.put(h->l);

if (h->r != 0) q.put(h->r);

}

}

Where QUEUE is defined as follows:

**class QUEUE**

**{**

**private:**

**int \*q; int N, head, tail;**

**public:**

**QUEUE(int maxN)**

**{ q = new link[maxN+1];**

**N = maxN+1; head = N; tail = 0; }**

**int empty() const**

**{ return head % N == tail; }**

**void put(link item)**

**{ q[tail++] = item; tail = tail % N; }**

**link get()**

**{ head = head % N; return q[head++]; }**

**};**

Your program will receive the following input and do the following for each input:

C <int r> : Create the root node with item r. If the tree is not empty (i.e. the root exists), return an error

L <int p> <int l>: Find the node with item p. Let u be that node. Create a new node v with item l. Make node v node u’s left child. If node u has already a left child, return an error

R <int p> <int r>: Find the node with item p. Let u be that node. Create a new node v with item r. Make node v node u’s right child. If node u has already a right child, return an error

D <int p>: Delete the subtree rooted at node with item p. Print out nodes deleted in **pre-order** manner (i.e. pre-order: print the current-node, print left-subtree, and print right subtree, in this order). This should be the order you physically delete the nodes also.

P: Print the items in nodes in the entire tree in level-order traversal manner

S <int p>: Find the node with item p. Let u be that node. Print the items in nodes in the subtree rooted at u in level-order traversal manner

See the example dialogue below:

C 5

Root node with item 5 has been created

C 10

Error: Tree is not empty

L 5 3

Node with item 3 has been added

R 5 4

Node with item 4 has been added

P

Level-order traversal of the entire tree: 5 3 4

L 4 7

R 7 8

R 3 9

L 9 11

R 4 12

L 7 13

L 8 14

R 12 15

R 8 16

L 13 17

P

Level-order traversal of the entire tree: 5 3 4 9 7 12 11 13 8 15 17 14 16

S 4

level-order traversal of the asked subtree: 4 7 12 13 8 15 17 14 16

S 7

level-order traversal of the asked subtree: 7 13 8 17 14 16

D 3

Nodes 3, 9, 11 have been deleted

D 7

Nodes 7, 13, 17, 8, 14, 16 have been deleted

L 5 19

R 19 20

L 4 21

S 4

level-order traversal of the asked subtree: 4 21 12 15

D 15

Nodes 15 have been deleted

L 21 29

R 21 23

L 12 25

P

Level-order traversal of the entire tree: 5 19 4 20 21 12 29 23 25