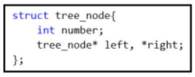
Data Structures 2012 Fall

Practice Session 6

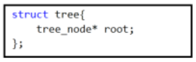
Contents

- Implementation of the following functions for tree structure:
  - A function to traverse a tree breadthwise.
  - A function to count leaves in a tree.
  - A function to calculate the sum of all node values in a tree.

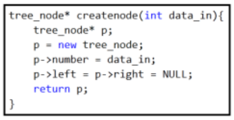
Tree Structure



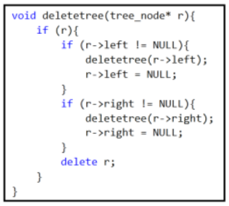
```
struct tree_node{
    int number;
    tree_node* left, *right;
};
```



```
struct tree{
    tree_node* root;
};
```

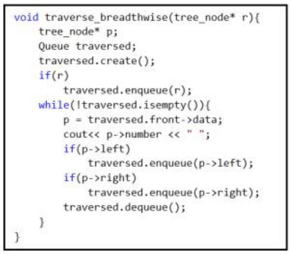


```
tree_node* createnode(int data_in){
    tree_node* p;
    p = new tree_node;
    p->number = data_in;
    p->left = p->right = NULL;
    return p;
}
```



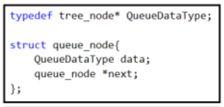
```
void deletetree(tree_node* r){
    if (r){
        if (r->left != NULL){
            deletetree(r->left);
            r->left = NULL;
        }
        if (r->right != NULL){
            deletetree(r->right);
            r->right = NULL;
        }
        delete r;
    }
}
```

Breadthwise Traversal of the Tree

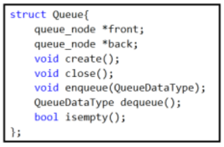


```
void traverse_breadthwise(tree_node* r){
    tree_node* p;
    Queue traversed;
    traversed.create();
    if(r)
        traversed.enqueue(r);
    while(!traversed.isempty()){
        p = traversed.front->data;
        cout<< p->number << " ";
        if(p->left)
            traversed.enqueue(p->left);
        if(p->right)
            traversed.enqueue(p->right);
        traversed.dequeue();
    }
}
```

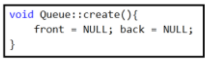
Structure of the Queue Which is Used During Breadthwise Traversal



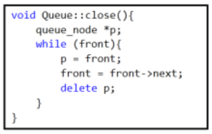
```
typedef tree_node* QueueDataType;

struct queue_node{
    QueueDataType data;
    queue_node *next;
};
```



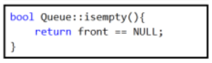
```
struct Queue{
    queue_node *front;
    queue_node *back;
    void create();
    void close();
    void enqueue(QueueDataType);
    QueueDataType dequeue();
    bool isempty();
};
```



```
void Queue::create(){
    front = NULL; back = NULL;
}
```

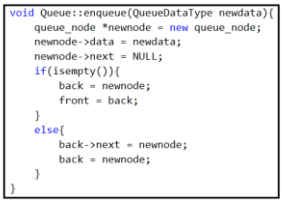


```
void Queue::close(){
    queue_node *p;
    while (front){
        p = front;
        front = front->next;
        delete p;
    }
}
```

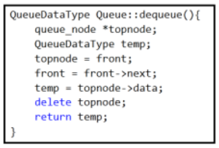


```
bool Queue::isempty(){
    return front == NULL;
}
```

Structure of the Queue Which is Used During Breadthwise Traversal



```
void Queue::enqueue(QueueDataType newdata){
    queue_node *newnode = new queue_node;
    newnode->data = newdata;
    newnode->next = NULL;
    if(isempty()){
        back = newnode;
        front = back;
    }
    else{
        back->next = newnode;
        back = newnode;
    }
}
```



```
QueueDataType Queue::dequeue(){
    queue_node *topnode;
    QueueDataType temp;
    topnode = front;
    front = front->next;
    temp = topnode->data;
    delete topnode;
    return temp;
}
```

Counting Leaves

```
int count_leaves(tree_node *r){
    if (!r->left && !r->right)
        return 1;
    else if(!r->left)
        return count_leaves(r->right);
    else if(!r->right)
        return count_leaves(r->left);
    else
        return count_leaves(r->left) + count_leaves(r->right);
}
```

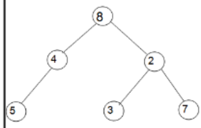
Calculating the Sum of Node Values

```
void get_sum_of_node_values(tree_node *r, int *sum){
    if(r){
        *sum += r->number;
        get_sum_of_node_values(r->left, sum);
        get_sum_of_node_values(r->right, sum);
    }
}
```

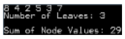
Test Program

```
int main(){
    tree t;
    t.root = createnode(8);
    tree_node *add = t.root;
    add->left = createnode(4);
    add->right = createnode(2);
    add = t.root->left;
    add->left = createnode(5);
    add = t.root->right;
    add->left = createnode(3);
    add->right = createnode(7);
    traverse_breadthwise(t.root);
    int numOfLeaves = count_leaves(t.root);
    cout<< "\nNumber of Leaves: " << numOfLeaves << endl;
    int sum = 0;
    get_sum_of_node_values(t.root, &sum);
    cout<< "\nSum of Node Values: " << sum << endl;
    deletetree(t.root);
    return 0;
}
```

Tree:



Screenshot:



```
8 4 2 5 3 7
Number of Leaves: 3

Sum of Node Values: 29
```