Review Activity 11

Tree Traversals using Stack

1) Implement a non-recursive function named **inorder_using_stack** that performs in-order traversal of a tree starting at a given **BinaryTreeNode* T**. Each **BinaryTreeNode** includes **data** of integer type, and pointers to **left** and **right** nodes. The function has friend access to **BinaryTreeNode**.

   Your function may only use one **stack** instance as part of its implementation; that is, the new **stack s**. You may also use a number of primitive temporary variables, but no other data structures.

   In your code, you may only call the **<stack>** methods, such as **bool empty(), int size(), void push (BinaryTreeNode* node), void pop(), and BinaryTreeNode* top()**. To process (visit) each node, output its **data** value to the console.

```
void inorder_using_stack(BinaryTreeNode* T) {
     stack<BinaryTreeNode*> s;
     // write your code here
     BinaryTreeNode* curr = T;
     while (curr || !s.empty()) {
        if (curr){
           s.push(curr);
           curr = curr -> left;
        }
        else if(!s.empty()) {
           curr = s.top(); s.pop();
           cout << curr -> data; << endl;
           curr = curr->right; }

     }
}
```

2) Implement a function named **print_BFT** that will print a given binary tree using breadth-first traversal. In your code, you may only call the **<queue>** methods, such as **bool empty(), int size(), void push (BinaryTreeNode* node), void pop(), and BinaryTreeNode* top()**. The function signature is as follows: **void print_BFT(BinaryTreeNode* T).** To process (visit) each node, output its **data** value to the console.

// Implement your code below

```
void print_BFT(BinaryTreeNode* T){
    queue<BinaryTreeNode*> q;
    if(T)
      q.pop(T);
    while(!q.empty()){
      BinaryTreeNode* cur = q.front();
      cout<< cur->data <<endl;
      if (cur->left )
          q.push(cur->left);
      if(cur->right)
          q.push(cur->right);
      q.pop();
    }
}
```