Implementing Forward Iterator in BST

Given a Binary search tree, the task is to implement forward iterator on it with the following functions.

1. **curr():** returns the pointer to current element.
2. **next():** iterates to the next smallest element in the Binary Search Tree.
3. **isEnd():** returns true if there no node left to traverse else false.

Iterator traverses the BST in sorted order(increasing). We will implement the iterator using a stack data structure.

Initialisation:

- *We will create a stack named "q" to store the nodes of BST.*
- *Create a variable "curr" and initialise it with pointer to root.*
- *While "curr" is not NULL*
    - *Push "curr" in the stack 'q'.*
    - *Set curr = curr -> left*

curr()

Returns the value at the top of the stack 'q'.

***Note:** It might throw segmentation fault if the stack is empty.*

Time Complexity: O(1)

next()

- *Declare pointer variable "curr" which points to node.*
- *Set curr = q.top()->right.*
- *Pop top most element of stack.*
- *While "curr" is not NULL*
    - *Push "curr" in the stack 'q'.*
    - *Set curr = curr -> left.*

Time Complexity: O(1) on average of all calls. Can be O(h) for a single call in the worst case.

isEnd()

Returns true if stack "q" is empty else return false.

Time Complexity: O(1)

Worst Case space complexity for this implementation of iterators is O(h). It

should be noticed that
iterator points to the top-most element of the stack.
Below is the implementation of the above approach:

C++

```
// C++ implementation of the approach

#include <bits/stdc++.h>

using namespace std;


// Node of the binary tree

struct node {

    int data;

    node* left;

    node* right;

    node(int data)

    {

        this->data = data;

        left = NULL;

        right = NULL;

    }

};


// Iterator for BST

class bstit {

private:

    // Stack to store the nodes

    // of BST

    stack<node*> q;
```

```
public:
    // Constructor for the class
    bstit(node* root)
    {
        // Initializing stack
        node* curr = root;
        while (curr != NULL)
            q.push(curr), curr = curr->left;
    }

    // Function to return
    // current element iterator
    // is pointing to
    node* curr()
    {
        return q.top();
    }

    // Function to iterate to next
    // element of BST
    void next()
    {
        node* curr = q.top()->right;
        q.pop();
        while (curr != NULL)
            q.push(curr), curr = curr->left;
    }
```

```
    // Function to check if
    // stack is empty
    bool isEnd()
    {
        return !(q.size());
    }
};

// Function to iterator to every element
// using iterator
void iterate(bstit it)
{
    while (!it.isEnd())
        cout << it.curr()->data << " ", it.next();
}

// Driver code
int main()
{
    node* root = new node(5);
    root->left = new node(3);
    root->right = new node(7);
    root->left->left = new node(2);
    root->left->right = new node(4);
    root->right->left = new node(6);
    root->right->right = new node(8);
```

```
    // Iterator to BST
    bstit it(root);

    // Function to test iterator
    iterate(it);

    return 0;
}
```

Output:

```
2 3 4 5 6 7 8
```