Class Names:

We are implementing three classes called StackType, FullStack, and EmptyStack. StackType is a template using the class ItemType

---

DESCRIPTION OF FUNCTIONS:

StackType Class Functions:

This is a template class with a template parameter "ItemType" and has the following functions:

1. **bool IsFull() const;**
   This function determines whether the stack is full.

2. **bool IsEmpty() const;**
   This function determines whether the stack is empty.

3. **void Push(ItemType item);**
   This function adds newItem to the top of the stack.

4. **void Pop();**
   This function removes the top item from the stack.

5. **ItemType Top();**
   This function returns a copy of the top item on the stack.

StackType Class Variables:

1. **Int top;**
   This variable holds the value that's at the top of the stack

2. **Int maxStack;**
   This variable holds the max size of the stack

3. **ItemType* item;**
   This variable is the stack

FullStack Class Functions:

This is an exception class thrown by the Push function when the stack is full and does not have any functions.

EmptyStack Class Functions:

This is an exception class thrown by the Pop and Top functions when the stack is empty.

FUNCTION TEST CASES:

Test cases for StackType Functions:

1. IsFull() has **two** possible test case(s):
   a. The stack is full.
   b. The stack is not full.

2. IsEmpty() has **two** possible test case(s):
   a. The stack is empty.
   b. The stack is not empty.

3. Push() has **two** possible test case(s):
   a. The item is added to the top of the stack.
   b. The item is not added to the top of the stack.

4. Pop() has **two** possible test case(s):
   a. The item is removed from the top of the stack.
   b. The item is not removed from the top of the stack.

5. Top() has **two** possible test case(s):
   a. A copy of the top item on the stack is returned.
   b. A copy of the top item on the stack is not returned.

LOGIC DESCRIPTION FOR REVERSING :

```
while (!stack.IsEmpty()) {
    item = stack.Top();
    stack.Pop();
    outFile << item;
}
```

This code reverses the order of items in the stack and writes them to an output file. It does this by repeatedly taking the top item from the stack, removing it, and writing it to the file until the stack is empty. This effectively reverses the order of items, with the original bottom item being written first in the output file.