```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// stackcpp.h
// A stack implementation in C++.
// Author: Lauren E. Scott
// June 27, 2014
//
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <iostream>
#include <stdlib.h>
#include <stdbool.h>

using namespace std;

template <class T>
class Stack {
public:
    Stack(int s) { array = new T[s]; size = s; contains = 0; }
    ~Stack() {}

    void push(T data);
    T pop();
    T top();
    bool isEmpty();
    bool isFull();
    bool hasTwoElements();
    void print();


private:
    T* array;
    int size, contains;
};

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// Function: push
// Push an item onto the top of the stack.
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

template <class T>
void Stack<T>::push(T data) {

    if(isFull()) {
        cout << "Stack is full. " << endl;
        cout << "Size: " << size << endl;
```

```
46          cout << "Contains: " << contains << endl;
47          return;
48      }
49      array[contains] = data;
50      contains++;
51      cout << "Successfully pushed " << data << endl;
52
53  }
54
55  /* * * * * * * * * * * * * * * * * * * * * * */
56  // Function: pop
57  // Pop an item off of the top of the stack, returning that item.
58  /* * * * * * * * * * * * * * * * * * * * * * */
59
60  template <class T>
61  T Stack<T>::pop() {
62      if(isEmpty()) {
63          cout << "Stack is empty. " << endl;
64      } else {
65          T result = array[contains-1];
66  //      delete array[contains];         // Set current position to null.
67  //      [array]contains = 0;
68          contains--;
69          return result;
70      }
71  }
72
73  /* * * * * * * * * * * * * * * * * * * * * * */
74  // Function: top
75  // Access the top item of the stack without returning it.
76  /* * * * * * * * * * * * * * * * * * * * * * */
77
78  template <class T>
79  T Stack<T>::top() {
80      if(isEmpty()) {
81          cout << "Stack is empty. " << endl;
82  //      return -1;
83      } else {
84          return array[contains-1];
85      }
86  }
87
88  /* * * * * * * * * * * * * * * * * * * * * * */
89  // Function: isEmpty
90  // Checks to see whether the stack (thus, underlying array) is empty.
```

```
/* * * * * * * * * * * * * * * * * * * * * * */

template <class T>
bool Stack<T>::isEmpty() {
    if (contains <= 0) {
        return true;
    }
    return false;
}

/* * * * * * * * * * * * * * * * * * * * * * */
// Function: isFull
// Checks to see whether the stack (thus, underlying array) is full
/* * * * * * * * * * * * * * * * * * * * * * */

template <class T>
bool Stack<T>::isFull() {
    if (contains >= size) {
        return true;
    }
    return false;
}

/* * * * * * * * * * * * * * * * * * * * * * */
// Function: hasTwoElements
// Checks to see whether the stack has at least two elements.
// A specific function written for the stack game.
/* * * * * * * * * * * * * * * * * * * * * * */

template <class T>
bool Stack<T>::hasTwoElements() {
    return (contains >= 2);
}

/* * * * * * * * * * * * * * * * * * * * * * */
// Function: print
// Prints the contents of the stack.
/* * * * * * * * * * * * * * * * * * * * * * */

template <class T>
void Stack<T>::print() {
    for(int i = contains-1; i >= 0; i--) {
        cout << " | " << array[i];
    }
    cout << " ||" << endl;
```

```
136 }
137
138
```