

```

/* *****
   Demonstrates a generic stack ADT.
   Uses composition with a RAW C_Array as its underlying storage.
   Uses NO dynamic storage allocation/deallocation.
   Mimics std::stack<T> for member function signatures.
   Advice: always choose std::stack<T> when in need of a stack.
   *****/
#ifndef STACK_H
#define STACK_H
#include <exception>
template <typename T, int capacity>
class Stack {
protected:
    T container[capacity]; // container for stack elements (cap known at compile time)
    // The following top_index data member plays two important roles:
    int top_index = 0; // (1) index to NEXT top element. (2) the current size of stack.
public:
    Stack() = default;
    virtual ~Stack() = default;

    Stack(const Stack<T, capacity>& other) = default; // copy constructor
    Stack<T, capacity>& operator=(const Stack<T, capacity>& other) = default; // copy assignment

    Stack(Stack<T, capacity>&& other) = default; // move constructor
    Stack<T, capacity>& operator=(Stack<T, capacity>&& other) = default; // move assignment

    // Returns the number of elements in the stack
    int size() const;

    // Returns whether the stack is empty
    bool empty() const;
    // Returns whether the stack is full
    bool full() const;

    // Inserts a new element at the top of the stack
    void push(const T& elem);

    // Removes the element on top of the stack.
    void pop();

    // Returns a reference to the top element in the stack.
    T& top();

    // Returns a constant reference to the top element in the stack.
    const T& top() const;
};
// Include Stack.cpp. Only for class templates.
// Be sure to remove Stack.cpp from your IDE's project.
#include "Stack.cpp"
#endif // STACK_H

```

```

// #include "Stack.h" // our own Stack<T, capacity> class
// The include line above is commented out because this entire
// stack.cpp file is already included at the bottom of stack.h

// Returns the number of elements in the stack
template<typename T, int capacity> // <a type param T, a non-type param capacity>
inline // a suggestion, the compiler will make final decision
int Stack<T, capacity>::size() const { return top_index; }

// Returns whether the stack is empty
template<typename T, int capacity>
inline bool Stack<T, capacity>::empty() const { return top_index == 0; }

// Returns whether the stack is full
template<typename T, int capacity>
inline bool Stack<T, capacity>::full() const { return top_index == capacity; }

// Inserts a new element at the top of the stack
template<typename T, int capacity>
inline void Stack<T, capacity>::push(const T & elem)
{
    if (full()) { throw std::overflow_error("Full stack - no push()"); }
    container[top_index] = elem;
    ++top_index;
}

// Removes the element on top of the stack.
template<typename T, int capacity>
inline void Stack<T, capacity>::pop()
{
    if (empty()) { throw std::underflow_error("Empty stack - no pop()"); }
    --top_index;
}

// Returns a reference to the top element in the stack.
template<typename T, int capacity>
inline T & Stack<T, capacity>::top()
{
    if (empty()) { throw std::underflow_error("Empty stack - no top()"); }
    return container[top_index - 1];
}

// Returns a constant reference to the top element in the stack.
template<typename T, int capacity>
inline const T & Stack<T, capacity>::top() const
{
    if (empty()) { throw std::underflow_error("Empty stack - no top()"); }
    return container[top_index - 1];
}

```

```

#include <iostream>
#include "Stack.h"      // use our own Stack<T> class
using namespace std;

// test drive for our stack class
int main()
{
    try {
        Stack<int, 10> intStack;

        // push five elements onto the stack
        for(int x = 1; x <=5; ++x)
            intStack.push(x);

        // pop and print three elements from the stack
        for (int x = 1; x <= 3; ++x)
        {
            cout << intStack.top() << " "; // process top element
            intStack.pop();                 // then pop it
        }
        cout << endl;

        // modify top element
        intStack.top() = 111;

        // push three new elements
        for (int x = 6; x <= 8; ++x)
            intStack.push(x);

        // print and pop six elements, one element too many
        for (int x = 1; x <= 6; ++x)
        {
            cout << intStack.top() << " "; // process top element
            intStack.pop();                 // then pop it
        }
        cout << endl;
    }
    catch (const exception& e)
    {
        cerr << "\nException: " << e.what() << endl;
    }
    return 0;
}

```

### Output

```

5 4 3
8 7 6 111 1
Exception: Empty stack - no top()

```