**Stacks**

The class stack<> implements a stack (also known as LIFO).

header file #include <stack>

The class stack is defined as follows:

namespace std {

template <typename T,

typename Container = deque<T>>

class stack;

}

**T:** Type of the elements

**Container:** optional parameter. Defines the container that the stack uses internally for its elements. The default container is a deque.

The stack implementation simply maps the operations into appropriate calls of the container that is used internally. You can use any sequence container class that provides the member functions back(), push\_back(), and pop\_back().

# The Core Interface

The core interface of stacks is provided by the member functions push(), top(), and pop().

**push():** inserts an element into the stack.

**top():** returns the next element in the stack without removing it

**pop():** removes an element from the stack but does not return it

Call both functions top() and pop() to process and remove the next element from the stack.

The behavior of top() and pop() is undefined if the stack contains no elements.

The member functions size() and empty() are provided to check whether the stack contains elements.

# Example

#include <iostream>

#include <stack>

using namespace std;

int main(void) {

stack<int> st;

// push three elements into the stack

st.push(1);

st.push(2);

st.push(3);

// pop and print two elements from the stack

cout << st.top() << " "; st.pop();

cout << st.top() << " "; st.pop();

// modify top element

st.top() = 77;

// push two new elements

st.push(4);

st.push(5);

// pop one element without processing it

st.pop();

// pop and print remaining elements

while (!st.empty()) {

cout << st.top() << " "; st.pop();

}

cout << endl;

return 0;

}

Output: 3 2 4 77

# Class stack<> in Detail

The stack<> interface maps more or less directly to corresponding members of the container internally used.

namespace std {

template <typename T, typename Container = deque<T>>

class stack {

protected:

Container c; // container

public:

typedef typename Container::value\_type value\_type;

typedef typename Container::reference reference;

typedef typename Container::const\_reference const\_reference;

typedef typename Container::size\_type size\_type;

typedef Container container\_type;

public:

bool empty() const { return c.empty(); }

size\_type size() const { return c.size(); }

void push(const value\_type& x) { c.push\_back(x); }

void push(value\_type&& x) { c.push\_back(move(x)); }

void pop() { c.pop\_back(); }

value\_type& top() { return c.back(); }

const value\_type& top() const { return c.back(); }

template <typename... Args>

void emplace(Args&&... args) {

c.emplace\_back(std::forward<Args>(args)...); }

void swap (stack& s) { swap(c,s.c); }

};

}

# END