C++ STL Stack

Constructors

```
explicit stack ( const Container& ctnr = Container() );
```

Construct stack

Constructs a `stack` container adaptor object.
A container adaptor keeps a container object as data. This container object is a copy of the argument passed to the constructor, if any, otherwise it is an empty container.

Example

```
  deque<int> mydeque (3,100);     // deque with 3 elements


  stack<int> first;               // empty stack
  stack<int> second (mydeque);    // stack initialized to copy of deque
  stack<int,vector<int> > third;  // empty stack using vector

  vector<int> myvector (2,200);   // vector with 2 elements
  stack<int,vector<int> > fourth (myvector);  // stack initialized with myvector
```

stack::empty()

```
bool empty ( ) const;
```

Test whether container is empty

Returns whether the stack is empty, i.e. whether its size is `0`.

stack::size()

```
size_type size ( ) const;
```

Return size

Returns the number of elements in the `stack`.

stack::top()

```
      value_type& top ( );
const value_type& top ( ) const;
```

Access next element

Returns a reference to the next element in the stack. Since stacks are last-in first-out containers this is also the last element pushed into the stack.

Member type `value_type` is defined to the type of value contained by the underlying container, which shall be the same as the first template parameter (`T`).

Example

```
  stack<int> mystack;

  mystack.push(10);
  mystack.push(20);

  mystack.top() -= 5;
```

stack::push()

```
void push ( const T& x );
```

Add element

Adds a new element at the top of the stack, above its current top element. The content of this new element is initialized to a copy of *x*.

Example

```
 for (int i=0; i<5; ++i) mystack.push(i);
```

stack::pop()

```
void pop ( );
```

Remove element

Removes the element on top of the stack, effectively reducing its size by one. The value of this element can be retrieved before being popped by calling member stack::top.

This calls the removed element's destructor.

Example

```
while (!mystack.empty())
{
   cout << " " << mystack.top();
   mystack.pop();
}
```

C++ STL Queue

Constructors

```
explicit queue ( const Container& ctnr = Container() );
```

Construct queue

Constructs a `queue` container adaptor object.
A container adaptor keeps a container object as data. This container object is a copy of the argument passed to the constructor, if any, otherwise it is an empty container.

Example

```
  deque<int> mydeck (3,100);   // deque with 3 elements
  list<int> mylist (2,200);    // list with 2 elements

  queue<int> first;            // empty queue
  queue<int> second (mydeck);  // queue initialized to copy of deque

  queue<int,list<int> > third; // empty queue with list as underlying container
  queue<int,list<int> > fourth (mylist);
```

queue::empty()

```
bool empty ( ) const;
```

Test whether container is empty

Returns whether the queue is empty, i.e. whether its size is `0`.

queue::size()

```
size_type size ( ) const;
```

Return size

Returns the number of elements in the `queue`.

queue::front()

```
      value_type& front ( );
const value_type& front ( ) const;
```

Access next element

Returns a reference to the next element in the queue. This is the "oldest" element in the queue and the same element that is popped out from the queue if member queue::pop is called.

Member type `value_type` is defined to the type of value contained by the underlying container, which shall be the same as the first template parameter (`T`).

Example

```
  myqueue.front() -= myqueue.back();
```

---

queue::back()

```
      value_type& back ( );
const value_type& back ( ) const;
```

Access last element

Returns a reference to the last element in the queue. This is the "newest" element in the queue i.e. the last element pushed into queue.

Member type `value_type` is defined to the type of value contained by the underlying container, which shall be the same as the first template parameter (`T`).

Example

```
  myqueue.front() -= myqueue.back();
```

---

queue::push()

```
void push ( const T& x );
```

Insert element

Adds a new element at the end of the queue, after its current last element. The content of this new element is initialized to a copy of x.

Example

```
myqueue.push (myint);
```

---

queue::pop()

```
void pop ( );
```

Delete next element

Removes the next element in the queue, effectively reducing its size by one. The element removed is the "oldest" element in the queue whose value can be retrieved by calling member queue::front.

This calls the removed element's destructor.

Example

```
while (!myqueue.empty())
{
  cout << " " << myqueue.front();
  myqueue.pop();
}
```