C++ deque

Constructor

Syntax:

```
explicit deque ( const Allocator& = Allocator() );
explicit deque ( size_type n, const T& value= T(), const Allocator& = Allocator() );
template <class InputIterator>
         deque ( InputIterator first, InputIterator last, const Allocator& = Allocator() );
deque ( const deque<T,Allocator>& x );
```

Default constructor: constructs an empty deque container, with no content and a size of zero.
Repetitive sequence constructor: Initializes the container with its content set to a repetition, n times, of copies of value.
Iteration constructor: Iterates between first and last, setting a copy of each of the sequence of elements as the content of the container.
Copy constructor: The deque container is initialized to have the same contents (copies) and properties as deque container x.

For example,

```
  deque<int> first;              // empty deque of ints
  deque<int> second (4,100);    // four ints with value 100
  deque<int> third (second.begin(),second.end());  // iterating through second
  deque<int> fourth (third);   // a copy of third
```

begin

Syntax:

```
      iterator begin ();
const_iterator begin () const;
```

Returns an iterator referring to the first element in the container.

Notice that unlike member deque::front, which returns a reference to the first element, this function returns a random access iterator.

end

Syntax:

```
      iterator end ();
const_iterator end () const;
```

Returns an iterator referring to the past-the-end element in the deque container.

Notice that, unlike member deque::back, which returns a reference to the element preceding this one, this function returns a random access iterator.

---

size

Syntax:

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

Returns the number of elements in the deque container.

---

resize

Syntax:

```
void resize ( size_type sz, T c = T() );
```

Resizes the container to contain sz elements.

If sz is smaller than the current container size, the content is reduced to its first sz elements, the rest being dropped.

If sz is greater than the current container size, the content is expanded by inserting at the end as many copies of c as needed to reach a size of sz elements.

Notice that this function changes the actual content of the container by inserting or erasing elements from it.

---

empty

Syntax:

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

Returns whether the deque container is empty, i.e. whether its size is 0.

operator[]

Syntax:

```
      reference operator[] ( size_type n );
const_reference operator[] ( size_type n ) const;
```

Returns a reference to the element at position n in the deque container.

For example,

```
  for (i=0; i<sz; i++)
    cout << " " << mydeque[i];
```

front

Syntax:

```
      reference front ( );
const_reference front ( ) const;
```

Returns a reference to the first element in the deque container.

back

Syntax:

```
      reference back ( );
const_reference back ( ) const;
```

Returns a reference to the last element in the container.

assign

Syntax:

```
template <class InputIterator>
  void assign ( InputIterator first, InputIterator last );
void assign ( size_type n, const T& u );
```

Assigns new content to the container, dropping all the elements contained in it before the call and replacing them by those specified by the parameters:

In the first version (with iterators), the new contents are a copy of the elements contained in the sequence between first and last (in the range [first,last)).

In the second version, the new content is the repetition n times of copies of element u.

For example,

```
  deque<int> first;
  deque<int> second;
  deque<int> third;

  first.assign (7,100);             // a repetition 7 times of value 100

  deque<int>::iterator it;
  it=first.begin()+1;

  second.assign (it,first.end()-1); // the 5 central values of first

  int myints[] = {1776,7,4};
  third.assign (myints,myints+3);   // assigning from array.

  cout << "Size of first: " << int (first.size()) << endl;
  cout << "Size of second: " << int (second.size()) << endl;
  cout << "Size of third: " << int (third.size()) << endl;
```

Output:

```
  Size of first: 7
  Size of second: 5
  Size of third: 3
```

push_back

Syntax:

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

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

For example,

```
  deque<int> mydeque;
  int myint;

  cout << "Please enter some integers (enter 0 to end):\n";

  do {
    cin >> myint;
    mydeque.push_back (myint);
  } while (myint);
```

push_front

Syntax:

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

Inserts a new element at the beginning of the deque container, right before its current first element. The content of this new element is initialized to a copy of x.

For example,

```
  deque<int> mydeque (2,100);     // two ints with a value of 100
  mydeque.push_front (200);
  mydeque.push_front (300);

  cout << "mydeque contains:";
  for (unsigned i=0; i<mydeque.size(); ++i)
   cout << " " << mydeque[i];
```

Output:

```
 300 200 100 100
```

pop_back

Syntax:

```
void pop_back ( );
```

Removes the last element in the deque container, effectively reducing the container size by one.

For example,

```
deque<int> mydeque;
int sum (0);
mydeque.push_back (10);
mydeque.push_back (20);
mydeque.push_back (30);

while (!mydeque.empty())
{
  sum+=mydeque.back();
  mydeque.pop_back();
}

cout << "The elements of mydeque summed " << sum << endl;
```

Output:

```
The elements of mydeque summed 60
```

pop_front

Syntax:

```
void pop_front ( );
```

Removes the first element in the deque container, effectively reducing the deque size by one.

For example,

```
deque<int> mydeque;
 int sum (0);
 mydeque.push_back (100);
 mydeque.push_back (200);
 mydeque.push_back (300);

 cout << "Popping out the elements in mydeque:";
 while (!mydeque.empty())
 {
   cout << " " << mydeque.front();
   mydeque.pop_front();
 }

 cout << "\nFinal size of mydeque is " << int(mydeque.size()) << endl;
```

Output:

```
Popping out the elements in mydeque: 100 200 300
Final size of mydeque is 0
```

insert

Syntax:

```
iterator insert ( iterator position, const T& x );
    void insert ( iterator position, size_type n, const T& x );
template <class InputIterator>
    void insert ( iterator position, InputIterator first, InputIterator last );
```

The deque container is extended by inserting new elements before the element at the specified position.

For example,

```
 deque<int> mydeque;
  deque<int>::iterator it;

  // set some initial values:
  for (int i=1; i<6; i++) mydeque.push_back(i); // 1 2 3 4 5

  it = mydeque.begin();
  ++it;

  it = mydeque.insert (it,10);                  // 1 10 2 3 4 5
  // "it" now points to the newly inserted 10

  mydeque.insert (it,2,20);                     // 1 20 20 10 2 3 4 5
  // "it" no longer valid!

  it = mydeque.begin()+2;

  vector<int> myvector (2,30);
  mydeque.insert (it,myvector.begin(),myvector.end());
 // 1 20 30 30 20 10 2 3 4 5

  cout << "mydeque contains:";
  for (it=mydeque.begin(); it<mydeque.end(); ++it)
    cout << " " << *it;
  cout << endl;
```

Output:

```
 mydeque contains: 1 20 30 30 20 10 2 3 4 5
```

erase

Syntax:

```
iterator erase ( iterator position );
iterator erase ( iterator first, iterator last );
```

Removes from the deque container either a single element (position) or a range of elements ([first,last)).

For example,

```
unsigned int i;
 deque<unsigned int> mydeque;

 // set some values (from 1 to 10)
 for (i=1; i<=10; i++) mydeque.push_back(i);

 // erase the 6th element
 mydeque.erase (mydeque.begin()+5);

 // erase the first 3 elements:
 mydeque.erase (mydeque.begin(),mydeque.begin()+3);

 cout << "mydeque contains:";
 for (i=0; i<mydeque.size(); i++)
   cout << " " << mydeque[i];
 cout << endl;
```

Output:

```
mydeque contains: 4 5 7 8 9 10
```

swap

Syntax:

```
iterator erase ( iterator position );
iterator erase ( iterator first, iterator last );
```

Exchanges the content of the vector by the content of dqe, which is another deque object containing elements of the same type. Sizes may differ.

For example,

```
 unsigned int i;
  deque<int> first (3,100);   // three ints with a value of 100
  deque<int> second (5,200);  // five ints with a value of 200

  first.swap(second);

  cout << "first contains:";
  for (i=0; i<first.size(); i++) cout << " " << first[i];

  cout << "\nsecond contains:";
  for (i=0; i<second.size(); i++) cout << " " << second[i];
```

Output:

```
  first contains: 200 200 200 200 200
  second contains: 100 100 100
```

clear

Syntax:

```
void clear ( );
```

All the elements in the deque container are dropped: their destructors are called, and then they are removed from the container, leaving it with a size of 0.

For example,

```
unsigned int i;
deque<int> mydeque;
mydeque.push_back (100);
mydeque.push_back (200);
mydeque.push_back (300);

cout << "mydeque contains:";
for (i=0; i<mydeque.size(); i++) cout << " " << mydeque[i];

mydeque.clear();
mydeque.push_back (1101);
mydeque.push_back (2202);

cout << "\nmydeque contains:";
for (i=0; i<mydeque.size(); i++) cout << " " << mydeque[i];
```

Output:

```
mydeque contains: 100 200 300
mydeque contains: 1101 2202
```