C++ Vector

Constructors

Syntax:

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

Default constructor: constructs an empty vector, with no content and a size of zero.
Repetitive sequence constructor: Initializes the vector 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 vector is initialized to have the same contents (copies) and properties as vector x.

For example,

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

Notice that unlike member vector::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 vector container.
Notice that unlike member vector::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 vector container.

This is the number of actual objects held in the vector, which is not necessarily equal to its storage capacity. Vectors automatically reallocate their storage space when needed or when requested with member resize. To retrieve the current storage capacity of a vector you can call to its member capacity.

---

resize

Syntax:

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

Resizes the vector to contain sz elements.

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

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

---

capacity

Syntax:

```
size_type capacity () const;
```

Returns the size of the allocated storage space for the elements of the vector container.

Notice that, in vectors, the capacity is not necessarily equal to the number of elements that conform the underlying vector content (this can be obtained with member vector::size), but the capacity of the actual allocated space, which is either equal or greater than the content size.

---

empty

Syntax:

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

Returns whether the vector 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 vector container.

For example,

```
cout << "myvector contains:";
for (i=0; i<sz; i++)
    cout << " " << myvector[i];
```

---

front

Syntax:

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

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

---

back

Syntax:

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

Returns a reference to the last element in the vector 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 vector object, dropping all the elements contained in the vector 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 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,

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

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

  vector<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 vector, after its current last element. The content of this new element is initialized to a copy of x.

For example,
```
  vector<int> myvector;
  int myint;

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

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

  cout << "myvector stores " << (int) myvector.size() << " numbers.\n";
```

pop_back

Syntax:
```
void pop_back ( );
```

Removes the last element in the vector, effectively reducing the vector size by one and invalidating all iterators and references to it.

For example,
```
  vector<int> myvector;
  int sum (0);
  myvector.push_back (100);
  myvector.push_back (200);
  myvector.push_back (300);

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

  cout << "The elements of myvector summed " << sum << endl;
```
Output:
```
The elements of myvector summed 600
```

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 vector is extended by inserting new elements before the element at position.

For example,

```
    vector<int> myvector (3,100);
  vector<int>::iterator it;

  it = myvector.begin();
  it = myvector.insert ( it , 200 );

  myvector.insert (it,2,300);

  // "it" no longer valid, get a new one:
  it = myvector.begin();

  vector<int> anothervector (2,400);
  myvector.insert (it+2,anothervector.begin(),anothervector.end());

  int myarray [] = { 501,502,503 };
  myvector.insert (myvector.begin(), myarray, myarray+3);

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

Output:

```
myvector contains: 501 502 503 300 300 400 400 200 100 100 100
```

erase

Syntax:

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

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

For example,

```
  unsigned int i;
  vector<unsigned int> myvector;

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

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

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

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

Output:

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

swap

Syntax:

```
void swap ( vector<T,Allocator>& vec );
```

Exchanges the content of the vector by the content of vec, which is another vector of the same type. Sizes may differ.

After the call to this member function, the elements in this container are those which were in vec before the call, and the elements of vec are those which were in this. All iterators, references and pointers remain valid for the swapped vectors.

For example,

```
  unsigned int i;
  vector<int> first (3,100);   // three ints with a value of 100
  vector<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];

  cout << endl;
```

Output:

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

clear

Syntax:

```
void clear ( );
```

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

For example,

```
  unsigned int i;
  vector<int> myvector;
  myvector.push_back (100);
  myvector.push_back (200);
  myvector.push_back (300);

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

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

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

  cout << endl;
```

Output:

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