Standard Template Library

Container Classes

Sequence Containers

Contiguous blocks of objects; you can insert elements at any point in the sequence, but the performance will depend on the type of sequence and where you are inserting.

- **vector** Fast insertion at end, and allow random access.
- **list** Fast insertion anywhere, but provide only sequential access.
- **deque** Fast insertion at either end, and allow random access.

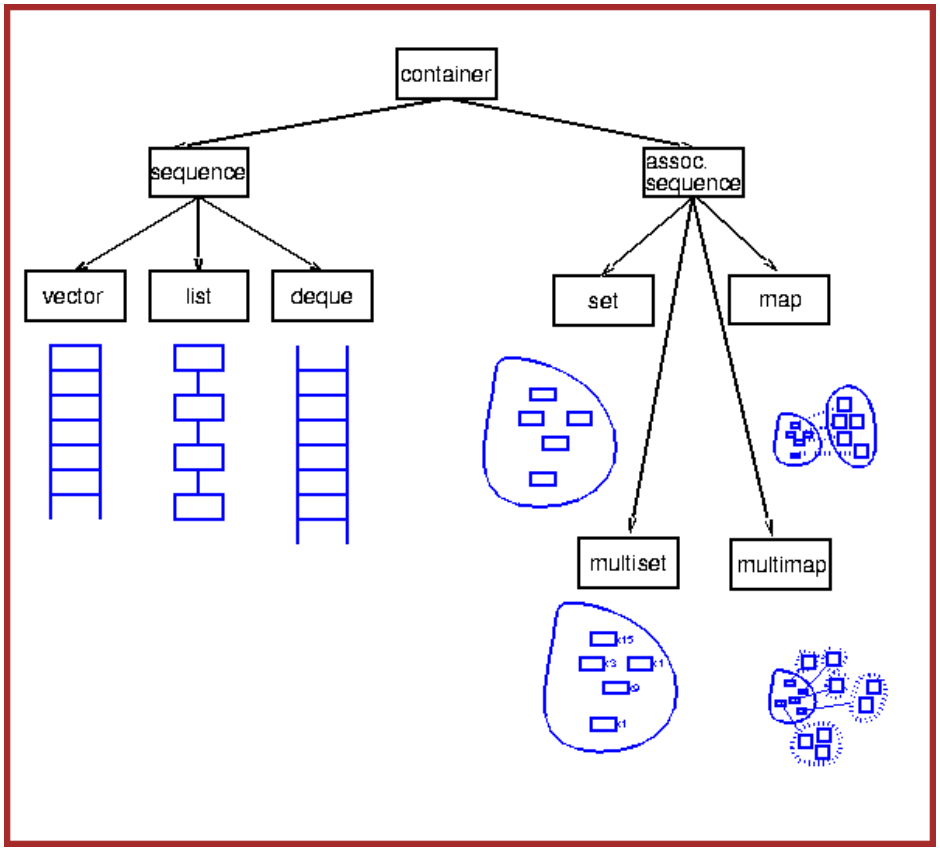
Associative containers

A generalisation of sequences. Sequences are indexed by integers; Associative containers can be indexed by any type.

- **set** Sets allow you to add and delete elements, query for membership, and iterate through the set.
- **multiset** Multisets are just like sets, except that you can have several copies of the same element (these are often called bags).
- **map** Maps represent a mapping from one type (the *key* type) to another type (the *value* type). You can associate a value with a key, or find the value associated with a key, very efficiently; you can also iterate through all the keys.
- **multimap** Multimaps are just like maps except that a key can be associated with several values.

Standard Template Library

Container Classes



From: Jak Kirman: A very modest STL tutorial

STL, vector-container (1)

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{ vector<int> v(10); // create a vector of length 10
  cout << "Size = " << v.size() << endl;  // display original size of v

  for(int i=0; i<10; i++) v[i] = i; // assign the elements of the vector some values

  cout << "Current Contents:\n";   // display contents of vector
  for(int i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  cout << "Expanding vector\n";
  for(int i=0; i<5; i++) v.push_back(i + 10);

  cout << "Size now = " << v.size() << endl;

  cout << "Current contents:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  for(i=0; i<v.size(); i++) v[i] = -v[i];

  cout << "Modified Contents:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;

  return 0;
}
```

Access elements using [] operator

put more values onto **the end** of the vector, it will grow as needed

change contents of vector

```
Size = 10
Current Contents:
0 1 2 3 4 5 6 7 8 9

Expanding vector
Size now = 15
Current contents:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Modified Contents:
0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14
```


STL, vector-container (2)

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{ vector<int> v(10);          // create a vector of length 10
  vector<int>::iterator p; // create an iterator

  int i;

  for(p= v.begin(), i=0; p!= v.end(); p++, i++)
      *p= i;

  // display contents of vector

  cout << "Original contents:\n";

  for(p= v.begin(); p!= v.end(); p++)
      cout << *p << " ";

  cout << "\n\n";

  for(p= v.begin(); p!= v.end(); p++)
      *p = *p * 2;

  for(p= v.begin(); p!= v.end(); p++)
      cout << *p << " ";

  cout << endl;
  return 0;
}
```

Access the elements of a vector through an iterator.

assign elements in vector

display contents of vector

change contents of vector

```
Original contents:
0 1 2 3 4 5 6 7 8 9

0 2 4 6 8 10 12 14 16 18
```

STL, vector-container (3)

Declaration of vector template

```
template<class T, class A = allocator<T> >
class vector
{
 public:
  typedef A allocator_type;
  typedef A::size_type size_type;
  typedef A::difference_type difference_type;
  typedef A::reference reference;
  typedef A::const_reference const_reference;
  typedef A::value_type value_type;

  class iterator;
  class const_iterator;
  typedef typename std::reverse_iterator<iterator>
                   reverse_iterator;
  typedef typename std::reverse_iterator<const iterator>
                   const_reverse_iterator;

//Constructors:
explicit
  vector(const A& al = A());

explicit
  vector(size_type n, const T& v = T(), const A& al = A());


  vector(const vector& x);


template<class InputIt>
  vector(InputIt first, InputIt last,
         const A& al = A());
```

The default constructor. Creates a vector of length zero. The vector will use the allocator alloc for all storage management.

Member-function TEMPLATE (Constructor)!!!
Creates a vector of length last - first, filled with all values obtained by dereferencing the InputIterators on the range [first, last).

STL, vector-container (4)

Declaration of vector template

```
void reserve(size_type n);
size_type capacity() const;

iterator begin();
const_iterator begin() const;
iterator end();
iterator end() const;
reverse_iterator rbegin();
const_reverse_iterator rbegin() const;
reverse_iterator rend();
const_reverse_iterator rend() const;

void resize(size_type n, T x = T());
size_type size() const;
size_type max_size() const;

bool empty() const;
A get_allocator() const;

reference at(size_type pos);
const_reference at(size_type pos) const;
reference operator[](size_type pos);
const_reference operator[](size_type pos);
reference front();
const_reference front() const;
reference back();
const_reference back() const;
```

iterators

accessing elements

STL, vector-container (5)

Declaration of vector template

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

  template<class InputIt>
      void assign(InputIt first, InputIt last);

  template<class Size, class T2>
      void assign(Size n, const T2& x = T2());

  iterator insert(iterator it, const T& x = T());

      void insert(iterator it, size_type n, const T& x);

  template<class InputIt>
    void insert(iterator it, InputIt first, InputIt last);

  iterator erase(iterator it);

  iterator erase(iterator first, iterator last);

  void clear();

  void swap(vector x);

protected:
    A allocator;
    };
```


STL, vector-container (6)

```
#include <iostream>
#include <vector>
using namespace std;

template <class T>
void printVector(vector<T>& v)
{ cout << "Vector size= " << v.size() << endl;
  for(vector<T>::iterator p= v.begin(); p!= v.end(); p++)
      cout << *p << " ";
  cout << "\n\n";
}
```

Insert and Erase Elements

```
int main()
{   vector<int> v1(10);
    vector<int> v2(5);

    vector<int>::iterator p;    // create an iterator

  for(p= v1.begin(), i=0; p!= v1.end(); p++, i++) *p= i;
  for(p= v2.begin(), i=0; p!= v2.end(); p++, i++) *p= 10*i;

  v1.insert(v1.begin()+4, v2.begin(), v2.end());

  printVector(v1);

  v1.erase(v1.begin()+7, v1.begin()+10);

  printVector(v1);

  cout << endl;

  return 0;
}
```

```
Vector size= 15
0 1 2 3 0 10 20 30 40 4 5 6 7 8 9

Vector size= 12
0 1 2 3 0 10 20 5 6 7 8 9
```

STL, vector-container (7)

```
int main()
{ int A[10]={12,23,34,45,56,78,89,90,91,92};
  vector<int> v1(A, A + 10 );

  printVector(v1);

  vector<int> v2;
  vector<int>::iterator p;
  v2= v1;
  printVector(v2);
  v1[2]= -10;
  printVector(v1);  printVector(v2);
  cout << endl;
  return 0;
}
```

Use of constructor template:

```
template<class InputIt>
 vector(InputIt first, InputIt last,
        const A& al = A());
```

Assignment
Deep assignment

```
Vector size= 10
12 23 34 45 56 78 89 90 91 92      // v1

Vector size= 10
12 23 34 45 56 78 89 90 91 92      // v2

Vector size= 10
12 23 -10 45 56 78 89 90 91 92     // v1

Vector size= 10
12 23 34 45 56 78 89 90 91 92      // v2
                                   // not changed !!
```


STL, map-container (1)

An associative container:

- unique keys are mapped with values
- a map ~ a list of key/value pairs
- look-up a value given its key
- key-type must have an order, a Comp-function

```
#include <iostream>
#include <map>
using namespace std;
int main()
{ map<char, int> m;
  int i;

  for(i=0; i<26; i++)
      m.insert(pair<char, int>('A'+i, 65+i));

  map<char, int>::iterator p;

  for(p = m.begin(); p != m.end(); p++)
  { cout << p->first << " has ASCII value of ";
    cout << p->second << endl;
  }

  return 0;
}
```

Cycle through a map using an iterator .

put pairs into map

```
A has ASCII value of 65
B has ASCII value of 66
C has ASCII value of 67
D has ASCII value of 68
E has ASCII value of 69
F has ASCII value of 70
G has ASCII value of 71
H has ASCII value of 72
I has ASCII value of 73
J has ASCII value of 74
K has ASCII value of 75
L has ASCII value of 76
M has ASCII value of 77
N has ASCII value of 78
O has ASCII value of 79
P has ASCII value of 80
…
```

```
template<class T, class U>
  struct pair {
    typedef T first_type;
    typedef U second_type
    T first;
    U second;
    pair();
    pair(const T& x, const U& y);
    template<class V, class W>
      pair(const pair<V, W>& pr);
  };
```

STL, map-container (2)

The [] operator

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

int main()

{  map<char, int> m;
   int i;

   for(i=0; i<26; i++)
      m.insert(pair<char, int>('A'+i, 65+i));

   string s("JAVAKYNDIGE");

   cout << "char" <<  "\t" << "value" << endl;

   for(string::iterator p= s.begin(); p != s.end(); p++)

      { cout << *p <<  "\t" <<  m[*p] << "\n";
      }

   cout << endl;

   return 0;
}
```

look-up characters
Ascii-value

char	value
J	74
A	65
V	86
A	65
K	75
Y	89
N	78
D	68
I	73
G	71
E	69

STL, map-container (3)

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

int main()

{  map<string,int> myPBook;

   myPBook["Ole"]= 45678901;
   myPBook["Peter"]= 36782345;
   myPBook["Camilla"]= 67890123;
   myPBook["Lone"]= 56789012;
   myPBook["Carsten"]= 23456789;

   cout << "phone book size= " << myPBook.size()<< endl;;

   string names[]=
     {"Lone", "Ole", "Carsten", "Mikkel", "Peter", "Susanne"};

   for(int i=0; i<6; i++)
     {cout << (names[i])<< " has number "
           << myPBook[names[i]]<< "\n"; }


   cout << "\nphone book size= " << myPBook.size()<< endl;

   cout << endl;
}
```

Asking for a non-existing key adds a new entry mapping the key to a default value

```
phone book size= 5

Lone has number 56789012
Ole has number 45678901
Carsten has number 23456789
Mikkel has number 0
Peter has number 36782345
Susanne has number 0

phone book size= 7
```

STL, map-container (4)

```
int main()
{  map<string,int> myPBook;
   myPBook["Ole"]= 45678901;
   myPBook["Peter"]= 36782345;
   myPBook["Camilla"]= 67890123;
   myPBook["Lone"]= 56789012;
   myPBook["Carsten"]= 23456789;
   cout << "phone book size= " << myPBook.size()<< endl;;
   string names[]=
     {"Lone", "Ole", "Carsten", "Mikkel", "Peter", "Susanne"};
   for(int i=0; i<6; i++)
     {cout << (names[i])<< " has number " << myPBook[names[i]]<< "\n"; }
   cout << "\nphone book size= " << myPBook.size()<< endl;

map<string,int>::iterator p;
for( p= myPBook.begin(); p!= myPBook.end(); p++)
   {cout << p-> first << " has number " << p->second << "\n";}
}
```

```
Camilla has number 67890123
Carsten has number 23456789
Lone has number 56789012
Mikkel has number 0
Ole has number 45678901
Peter has number 36782345
Susanne has number 0
```

```
phone book size= 5

Lone has number 56789012
Ole has number 45678901
Carsten has number 23456789
Mikkel has number 0
Peter has number 36782345
Susanne has number 0

phone book size= 7
```

Notice, the key's are ordered alphabetically
(the default order < for string's)

STL, map-container (5)

Storing Class Objects in a Map

```
// The key class.
class name
{  public:
     name() { str = ""; }
     name(string s) { str = s; }
     string getName() { return str;}
   private:
     string str;
};
// Define less than relative to
// name objects.
bool operator<(name a, name b)
{  return a.getName() < b.getName();
}
```

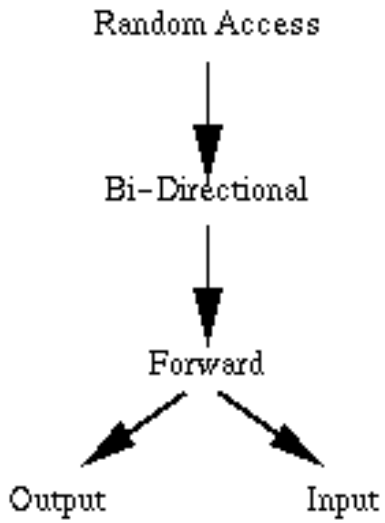
```
// The value class.
class phoneNum
{  public:
     phoneNum() { str = ""; }
     phoneNum(string s) { str = s; }
     string getPnum() { return str; }
   private:
     string str;
};
```

```
int main()
{  map<name, phoneNum> directory;
   // put names and numbers into map
   directory.insert(
     pair<name, phoneNum> (name("Camilla"),phoneNum("67890123")));
   directory.insert(
     pair<name, phoneNum> (name("Ole"),phoneNum("45678901")));
   directory.insert(
     pair<name, phoneNum> (name("Peter"),phoneNum("36782345")));
   cout << directory[name("Ole")].getPnum() << endl;
}
```

Output:

45678901

Iterators



Input Iterator	retrieve, but not store values; forward moving only
Output Iterator	store, but not retrieve values; forward moving only
Forward Iterator	store and retrieve values, forward moving only
Bidirectional Iterator	store and retrieve values, forward and backward moving
Random Access Iterator	store and retrieve values, elements can be accessed randomly

Algorithms

replace

```
template <class ForwardIterator, class T>
void replace( ForwardIterator first, ForwardIterator last,
              const T& old_value, const T& new_value)
```

Description

Replace replaces every element in the range [first, last) equal to old_value with new_value. That is: for every iterator i, if *i == old_value then it performs the assignment *i = new_value.

Example

```
#include <algorithm>
#include <vector>
#include <string>
#include <iostream>

int main()
{
  int A[10]={12,23,34,45,56,78,34,90,91,92};
  vector<int> v1(A, A + 10 );

  printVector(v1);
  replace(v1.begin(), v1.end(), 34, 100 );
  printVector(v1);

  string s("C++ Programmers Reference");
  replace(s.begin(),s.end(),'r', 'X');
  cout << s << endl;

}
```

```
Vector size= 10
12 23 34 45 56 78 34 90 91 92

Vector size= 10
12 23 100 45 56 78 100 90 91 92

C++ PXogXammeXs RefeXence
```

Algorithms

Non-mutating algorithms	Mutating algorithms	Sorting
for_each find find_if adjacent_find find_first_of count count_if mismatch equal search search_n find_end	copy copy_n copy_backward swap iter_swap swap_ranges transform replace replace_if replace_copy replace_copy_if fill fill_n generate generate_n remove remove_if remove_copy remove_copy_if unique unique_copy reverse reverse_copy rotate rotate_copy random_shuffle random_sample random_sample_n partition stable_partition	Sort sort stable_sort partial_sort partial_sort_copy is_sorted nth_element Binary search lower_bound upper_bound equal_range binary_search merge inplace_merge Set operations on sorted ranges includes set_union set_intersection set_difference set_symmetric_difference Heap operations push_heap pop_heap make_heap sort_heap is_heap Minimum and maximum min max min_element max_element lexicographical_compare lexicographical_compare_3way next_permutation prev_permutation
Generalized numeric algorithms iota accumulate inner_product partial_sum adjacent_difference power		

Algorithms

transform

Transform is an overloaded name; there are actually two transform functions.

```
template <class InputIterator, class OutputIterator, class UnaryFunction>
OutputIterator transform(InputIterator first, InputIterator last,
                         OutputIterator result, UnaryFunction op);
```

```
template <class InputIterator1, class InputIterator2, class OutputIterator,
          class BinaryFunction>
OutputIterator transform(InputIterator1 first1, InputIterator1 last1,
                         InputIterator2 first2, OutputIterator result,
                         BinaryFunction binary_op);
```

```
#include <algorithm>
#include <vector>
#include <string>
#include <iostream>

int add(int n, int m){return n+m;}

int main()
{ int A[10]={12,23,34,45,56,78,34,90,91,92};
  vector<int> V1(A,A+10);
  vector<int> V2(10);
  vector<int> V3(10);

  fill(V2.begin(), V2.end(), 100);
  printVector(V1);
  printVector(V2);

  transform(V1.begin(), V1.end(), V2.begin(), V3.begin(),add);
  printVector(V3);
}
```

```
Vector size= 10
12 23 34 45 56 78 34 90 91 92

Vector size= 10
100 100 100 100 100 100 100 100 100 100

Vector size= 10
112 123 134 145 156 178 134 190 191 192
```