C++ Standard Template Library

http://www.yolinux.com/TUTORIALS/
LinuxTutorialC++STL.html

Standard Template Libraries (STL)

- The STL is a collection C++ libraries that allow you to use several well-known kinds of data structures with out having to program them.

- Examples : vectors, lists, stack, queue

- The STL library is available from the STL home page (http://www.sgi.com/tech/stl/)

Containers

- A Container is a data structure that holds several object of the same type or class.
- Lists, Vectors, Stacks, Queues, etc are all Containers.

Iterators

- Items in Containers are referred to be special objects called: *iterators*
- They are generalization of C's pointers.
- With an iterator class, you can process each item in a vector or a list by similar code

```
for( Iter p=c.begin(); p!=c.end(); ++p)
        process(*p);
```

- For any type T, list<T> and vector<T> are Containers. So there are iterator classes called

```
list<T>::iterator
for( list<T>::iterator p=c.begin(); p!=c.end(); ++p)
        process(*p);

vector<T>::iterator
for( vector<T>::iterator p=c.begin(); p!=c.end(); ++p)
        process(*p);
```

Container: vector

- Dynamic array of variables, struct or objects.

```
#include <vector>
vector<T> v;          /*T is any type or class*/
```

v.empty()	test to see if it is empty:
v.size()	find how many items are in it:
v.push_back(t)	push a t:T onto the end of v:
v.pop_back()	pop the front of v off v:
v.front()	get the front item of v:
v.back()	get the back item of v:
v[i]	Access the i'th item (0<=i<size()) without checking to see if it exists:
v.at(i)	Access the i'th item safely:
…….	

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

int main()
{
    vector<string> SS;

    SS.push_back("The number is 10");
    SS.push_back("The number is 20");
    SS.push_back("The number is 30");

    cout << "Loop by index:" << endl;

    int ii;
    for(ii=0; ii < SS.size(); ii++)
    {
        cout << SS[ii] << endl;
    }

    cout << endl << "Constant Iterator:" << endl;

    vector<string>::const_iterator cii;
    for(cii=SS.begin(); cii!=SS.end(); cii++)
    {
        cout << *cii << endl;
    }
}
```

```
Loop by index:
The number is 10
The number is 20
The number is 30

Constant Iterator:
The number is 10
The number is 20
The number is 30
```

```
…
vector<string> SS;
SS.push_back("The number is 10");
SS.push_back("The number is 20");
SS.push_back("The number is 30");

cout << endl << "Reverse Iterator:" << endl;


vector<string>::reverse_iterator rii;
for(rii=SS.rbegin(); rii!=SS.rend(); ++rii)
{
    cout << *rii << endl;
}


cout << endl << "Sample Output:" << endl;


cout << SS.size() << endl;
cout << SS[2] << endl;
swap(SS[0], SS[2]);
cout << SS[2] << endl;
…
```

```
Reverse Iterator:
The number is 30
The number is 20
The number is 10

Sample Output:
3
The number is 30
The number is 10
```

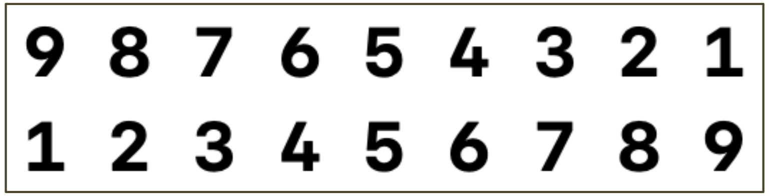
Container: List

- Linked list of variables, struct or objects..

```
#include <list>

list<T> l;
/*T is any type or class*/
```

l.empty()	test to see if it is empty:
l.size()	find how many items are in it:
l.push_back(t)	push a t:T onto the end of l:
l.pop_back()	pop the last off l:
l.push_front(t)	push a t:T onto the start of l:
l.pop_front()	pop the front of l off l:
l.front()	get the front item of l:
l.back()	get the back item of l:
l.sort()	Sort the list:
l.clear()	Clear the list:
l.reverse()	Reverse the list:
…….	

```
//Using a list to sort a sequence of 9 numbers.
#include<list>
#include <iostream>
using namespace std;

//Function to print list using iterators
void print(list<int> a)
{
    list<int>::const_iterator i;
    for(i=a.begin(); i!=a.end(); i++)
        cout << *i << " ";
    cout << endl;
}

int main()
{
    list<int> a;                    //Put 9,8,7,6,5,4,3,2,1 onto the list
    for(int i=0; i<9;++i)
        a.push_back(9-i);           // put new element after all the other
    print(a);
    a.sort();
    print(a);                       //here the list contains (1,2,3,4,5,6,7,8,9)
}
```



```
9 8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 9
```

```
list<int> L;
L.push_back(0);           // Insert a new element at the end
L.push_front(0);          // Insert a new element at the beginning
L.insert(++L.begin(),2);  // Insert "2" before pos of 1st argument
                          // (Place before second argument)
L.push_back(5);
L.push_back(6);

list<int>::iterator i;


for(i=L.begin(); i != L.end(); ++i) cout << *i << " ";
cout << endl;
```

```
0 2 0 5 6
```

```
struct student
{   int id;
    char name[20];
};
typedef struct student STUDENT;

STUDENT stu1, stu2, stu3, stu4, stu5;
list<STUDENT> L;

stu1.id = 1;        strcpy(stu1.name,"First");
stu2.id = 2;        strcpy(stu2.name,"Second");
stu3.id = 3;        strcpy(stu3.name,"Third");
stu4.id = 4;        strcpy(stu4.name,"Fourth");
stu5.id = 5;        strcpy(stu5.name,"Fifth");

L.push_back(stu3);              // Insert a new element at the end
L.push_front(stu1);             // Insert a new element at the beginning
L.insert(++L.begin(),stu2);     // Insert "2" before pos of 1st argument
L.push_back(stu4);
L.push_back(stu5);

list<STUDENT>::iterator i;
for(i=L.begin(); i != L.end(); ++i)
    cout << i->id << " " << i->name << "\n";
```

```
1 First
2 Second
3 Third
4 Fourth
5 Fifth
```

Container: Stacks

- Stack is a "last in first out" (LIFO) data structure

```
#include <stack>
stack<T> s;            /*T is any type or class*/
```

s.empty()	test to see if it is empty:
s.size()	find how many items are in it:
s.push(t)	push a t of type T onto the top:
s.pop()	pop the top off s:
s.top()	get the top item of s

```
#include<stack>

.......
    stack <string> cards;           // Simple enough to create a stack

    cards.push("King of Hearts");   // push() will add a value
    cards.push("King of Clubs");    // adding some cards to the deck
    cards.push("King of Diamonds");
    cards.push("King of Spades");

    cout << "There are " << cards.size () << " cards in the deck" << endl;
    cout << "The card on the top of the deck is " << cards.top() << endl;

    cards.pop();
    cout << "The top card is now " << cards.top() << endl;
    cout << cards.size() << endl;
```

```
There are 4 cards in the deck
The card on the top of the deck is King of Spades
The top card is now King of Diamonds
3
```

```
#include<stack>
………

void reverse(string & x)
{
    stack<char> s;

    for(int i=0; i < x.length(); ++i)
        s.push(x[i]);
    for(int i=0; !s.empty(); ++i, s.pop())
        x[i]=s.top();
}

int main()
{
    string str = "Welcome to Computer Science";
    reverse(str);
    cout << str << endl;

}
```

```
ecneicS retupmoC ot emocleW
```

Container: Queue

- Queues allow data to be added at one end and taken out of the other end.

```
#include <queue>
queue<T> q;          /*T is any type or class*/
```

q.empty()	test to see if it is empty:
q.size()	find how many items are in it:
q.push(t)	push a t of type T onto the end of q:
q.pop()	pop the front of q off q:
q.front()	get the front item of q:
q.back()	get the back item of q:

```
// A simple example of putting three items into a queue and
// then taking them off the queue.

#include <queue>
#include <iostream>
using namespace std;

int main()
{
    queue<char> q;
    q.push('a');
    q.push('b');
    q.push('c');
    cout << q.front();
    q.pop();
    cout << q.front();
    q.pop();
    cout << q.front();
    q.pop();
}
```

```
abc
```

Container: Priority Queue

- a container adaptor that provides constant time lookup of the largest (by default) element.

```
#include <queue>
priority_queue<T> q;        /*T is any type or class*/
```

pq.empty()	test to see if it is empty:
pq.size()	returns the number of element
pq.top()	accesses the top element
pq.push(t)	inserts element and sorts the underlying container
pq.pop()	Remove the top element

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main()
{
    priority_queue<int> pq;
    pq.push(3);
    pq.push(5);
    pq.push(1);
    pq.push(8);
    while ( !pq.empty() )
    {
        cout << pq.top() << endl;
        pq.pop();
    }
}
```

```
8
5
3
1
```