# STANDARD TEMPLATE LIBRARY -- LIST CONTAINER C++

In the interest of generic programming, the STL standard template library was created. It consists of three key components: containers, iterators and algorithms. There are three types of containers: sequence(vector,list,deque), associative(set,map,multiset) and container adaptors(stack,queue,priority_queue). List is a sequence container. Iterators are used to point to the elements of first class containers such as the list container. The iterators for list are bi-directional iterators(I++ I-- *I I=). The list is implemented as a doubly linked list.

| OPERATIONS | DESCRIPTION |
|---|---|
| Constructors | |
| List<t> l; | Construct l as an empty list. |
| List<t> l(n); | Construct l as a list to contain n elements (set to default value). |
| List<t> l(n,initVal); | Construct l as a list to contain n copies of initVal |
| List<t> l(fPtr,lPtr); | Construct l as a list to contain copies of elements in memory locations fptr up to lptr (pointers of type t*). |
| Copy constructor | |
| Destructor | |
| ~list() | Destroy contents, erasing all items. |
| l.empty() | Return true if an only if l contains no values |
| l.size() | Return the number of values l contains |
| l.push_back(value) | Append at end |
| l.push_front(value) | Insert value in front of first element |
| l.insert(pos,value) | Insert value into l at iterator position pos and return an iterator pointing to the new element's position |
| l.insert(pos,n,value); | Insert n copies of value into l at iterator position pos |
| l.insert(pos, fptr,lptr); | Insert copies of all the elements in the range fptr to lptr at iterator position pos |
| l.pop_back() | Erase last element |
| l.pop_front() | Erase first element |
| l.erase(pos) | Erase the value in l at iterator position pos |
| l.erase(pos1,pos2) | Erase the values in l from iterator positions pos1 to pos2 |
| l.remove(value) | Erase all elements in l that match value using == to compare items |
| l.unique() | Replace all repeating sequences of a single element by a single occurrence of that element |

| OPERATIONS | DESCRIPTION |
|---|---|
| l.front()<br>l.back() | Return reference to first element<br>Return a reference to last element |
| l.begin()<br>l.end() | Return an iterator positioned at first value<br>Return an iterator positioned past last value |
| l.rbegin()<br><br>l.rend() | Return a reverse iterator positioned at last value<br>Return a reverse iterator positioned before first value |
| l.sort()<br>l.reverse() | Sort elements using <<br>Reverse the order of the elements |
| L1.merge(L2)<br><br><br><br><br>L1.splice(pos,L2)<br><br>L1.splice(to,L2,from)<br><br><br>L1.splice(pos,L2,first,last)<br><br><br>L1.swap(L2) | Remove all the elements in L2 and merge them into L1, that is, move the elements of L2 into L1 and place them so that the final list of elements is sorted using <.<br>(assumes both lists sorted by <).<br>Remove all the elements in L2 and insert them into L1 at iterator position pos<br>Remove the element in L2 at iterator position from and insert it into L1 at iterator position 'to'<br>Remove all the elements in L2 at iterator positions(first , last) and insert them into L1 at iterator position pos<br>Swap the contents of L1 with L2 |
| L1 = L2 | Assign to L1 a copy of L2 |
| L1 == L2 | True if and only if L1 contains the same items as L2, in the same order |
| L1 < L2 | True if and only if L1 is lexicographically less than L2 |