# Lists in cpp

## Lists

The **list** class supports a bidirectional, linear list. Unlike a vector, which supports random access, a list can be accessed sequentially only. Since lists are bidirectional, they may be accessed front to back or back to front.

A **list** has this template specification:

template <class T, class Allocator = allocator<T> > class list

Here, **T** is the type of data stored in the list. The allocator is specified by **Allocator**, which defaults to the standard allocator.

It has the following constructors:

explicit list(const Allocator &*a* = Allocator( ) );

explicit list(size_type *num*, const T &*val* = T ( ),

const Allocator &*a* = Allocator( ));

list(const list<T, Allocator> &*ob*);

template <class InIter>list(InIter *start*, InIter *end*,

const Allocator &*a* = Allocator( ));

The first form constructs an empty list. The second form constructs a list that has *num*elements with the value *val*, which can be allowed to default. The third form constructs a list that contains the same elements as *ob*. The fourth form constructs a list that contains the elements in the range specified by the iterators *start* and *end*.

The following comparison operators are defined for **list**:

==, <, <=, !=, >, >=

Some of the commonly used **list** member functions are Like vectors, elements may be put into a list by using the **push_back( )** function. You can put elements on the front of the list by using **push_front( )**. An element can also be inserted into the middle of a list by using **insert( )**. Two lists may be joined using **splice( )**. One list may be merged into another using **merge( )**. For maximum flexibility and portability, any object that will be held in a list should define a default constructor. It should also define the < operator, and possibly other comparison

operators. The precise requirements for an object that will be stored in a list vary from compiler to compiler, so you will need to check your compiler's documentation.

Here is a simple example of a **list**.

```
// List basics.
#include <iostream>
#include <list>
using namespace std;
int main()
{
list<int> lst; // create an empty list
int i;
for(i=0; i<10; i++) lst.push_back(i);
cout << "Size = " << lst.size() << endl;
cout << "Contents: ";
list<int>::iterator p = lst.begin();
while(p != lst.end()) {
cout << *p << " ";
p++;
}
cout << "\n\n";
// change contents of list
p = lst.begin();
while(p != lst.end()) {
*p = *p + 100;
p++;
}
cout << "Contents modified: ";
p = lst.begin();
while(p != lst.end()) {
cout << *p << " ";
p++;
```

```
}
return 0;
}
```

The output produced by this program is shown here:

```
Size = 10
Contents: 0 1 2 3 4 5 6 7 8 9
Contents modified: 100 101 102 103 104 105 106 107 108 109
```

This program creates a list of integers. First, an empty **list** object is created. Next, 10 integers are put into the list. This is accomplished using the **push_back( )** function, which puts each new value on the end of the existing list. Next, the size of the list and the list itself is displayed. The list is displayed via an iterator, using the following code:

```
list<int>::iterator p = lst.begin();
while(p != lst.end()) {
cout << *p << " ";
p++;
}
```

Here, the iterator **p** is initialized to point to the start of the list. Each time through the loop, **p** is incremented, causing it to point to the next element. The loop ends when **p** points to the end of the list. This code is essentially the same as was used to cycle through a vector using an iterator. Loops like this are common in STL code, and the fact that the same constructs can be used to access different types of containers is part of the power of the STL.