**List**

**What:**

std::list is sequential STL container that is internally implemented as doubly linked list.  
i.e. every element in the list is stored at a seperate memory location i.e. called node and it also contains a pointer to the previous and next node.

**How to create a list:**

**syntax:**

std::list <type> listName;

// to insert element at end

listName.push\_back(data);

// to insert element at front

listName.push\_front(data);

**Insert element in the std::list**

insert() is used to insert the elements in between the list.

*insert(iterator\_pos, element)*

It allocates a new node and copy the passed element to it and then inserts that node before iterator position iterator\_position and returns the position of the newly added element.

**How to iterate (or) traverse through the vector:**

#include<list>

#include<iostream>

int main()

{

std::list<int> l;

l.push\_back(30);

l.push\_back(40);

l.push\_front(20);

l.push\_front(21);

std::list<int>::iterator it = l.begin();

while(it != l.end()){

std::cout << \*it << " ";

it++;

}

std::cout << "\n";

}

**How to erase the element from a list**

**syntax**:

erase(iterator\_position,elem)

It erases the element at the passed node and changes the left and right pointers of previous and next nodes. Also returns iterator of next node element. This opeartion takes O(1) time, as only swapping of pointers is required not like vector and deque, where other elements need to be shifted.

**Removal of element based on condition**

|  |  |
| --- | --- |
|  | //Lets remove all elements with value greater than 3.  listOfNumbers.remove\_if([](int elem)  {  if(elem > 3)  return true;  else  return false;  }); |

**Difference between list and vector:**

Both vector and list are sequential containers of C++ Standard Template Library. But there are many differences between them because of their internal implementation i.e.

List stores elements at non contiguous memory location i.e. it internally uses a doubly linked list i.e.

Whereas, vector stores elements at contiguous memory locations like an array i.e.

### 1. Insertion and Deletion

Insertion and Deletion in List is very efficient as compared to vector because to insert an element in list at start, end or middle, internally just a couple of pointers are swapped.

Whereas, in vector insertion and deletion at start or middle will make all elements to shift by one. Also, if there is insufficient contiguous memory in vector at the time of insertion, then a new contiguous memory will be allocated and all elements will be copied there.

So, insertion and deletion in list is much efficient than vector in c++.

2. Random Access:

As List is internally implemented as doubly linked list, therefore no random access is possible in List. It means, to access 15th element in list we need to iterate through first 14 elements in list one by one.

Whereas, vector stores elements at contiguous memory locations like an array. Therefore, in vector random access is possible i.e. we can directly access the 15th element in vector using operator [] i.e.

### 3. Iterator Invalidation

Deleting or Inserting an element in List does not invalidate any iterator because during insertion and deletion no element is moved from its position only a couple pointers are changed.

Whereas, in vector insertion and deletion can invalidate the iterators. For more details about vector Iterator Invalidation

**Different way to initialize a list**

creating a empty list:

std::list<int> listName;

**program**

#include <iostream>

#include <list>

int main() {

// Create an empty list of ints

std::list<int> listOfInts;

// Push back 10 elements in the list

for (int i = 0; i < 10; i++)

listOfInts.push\_back(i);

// Iterate over the list and display numbers

for (int val : listOfInts)

std::cout << val << ",";

std::cout << std::endl;

return 0;

}

creating and initializing list with fill constructor:

std::list<int> listOfInts(5, 119);

// This list will contain 5 elements and value it contains are 119

creating and initializing list with c++11 initializer list:

std::list<int> listOfInts({2,8,7,5,3,1,4});

This will initialize the list with the above entries and we dont need to use push back functions.

Creating and initializing list with array:

int arr[] = { 2, 8, 7, 5, 3, 1, 4 };

// Create a list and initialize it with vector

std::list<int> listOfInts(arr, arr + sizeof(arr) / sizeof(int));

**Note:**

To print the element in the list we can use the following

std::copy(lsit.begin(), list.end(), ostream\_iterator<type>(std::cout, “ ”));

**How to erase elements from a list using iterator:**

1. Move the iterator to the element position using loop or increment operator, then

it = list.erase(it);

Erase based on the range,

2. Erase of the data based on the range,

list.erase(it\_pos, end\_pos);

Refer:02\_listErase.cpp

**Remove elements using iterator:**

std::list provides a member function erase() that accepts an iterator and deletes the element pointed by that Iterator. But it makes that iterator invalid i.e. we cannot use that iterator because that is already deleted and all its links has become invalid. Therefore, std::list::erase() returns the iterator to the next of last deleted element. So, we can use this to continue our iteration.

it = list.erase(it);

Here the it will contain the position of the erased element.

Refer: 03\_listIter.cpp

**Remove element based on external criterion**

remove()

std::list provides two member functions for removing elements based on value i.e. std::list::remove and std::list::remove\_if.

Syntax:

list.remove(data);

// This will remove all the occurance of the data within the list

Refer: 04\_listRemove.cpp

remove\_if()

This is used to remove element based on the condition, and we dont use loop to iterate through it, we go with function pointer, function object and lambda function.

Syntax:

list.remove\_if(function\_ptr / fun\_obj, lamd\_fun);

**How to get element by index in a list:**

List does not have access to [] operator because std::list internally stores element in double linked list. So to access an element at nth location we need to iterate one by one from the beginning to nth  element.

But STL provides an algorithm **std::next** and **std::advance**

**How to find a element in list:**

1. Using iterator

iterator it;

it = std::find(listData.begin(), listData.end(), DATA);

*REFER: 07\_listfind.cpp*

**Different ways to iterate over a list of objects:**

*1. Using iterator and for loop*

std::list<type> ::iterator it;

for(it = list.begin() ; it != list.end() ; it++){

// Operations to be performed

}

*2. Using for\_each loop and lambda function*

std::for\_each( list.begin(), list.end(), [] (parameter){

// Function statement

};

*3. Using iterator with range based for loop*

for(const Type & t, listName){

// Operations to be performed

}

*4. Using reverse iterator*

std::list<type> ::reverse\_iterator it;

for(it = list.rbegin() ; it != list.rend() ; it++){

// Operations to be performed

}

**How to sort data in list using custom comparator or lambda function:**