Course Title: **Algorithm Lab**

Course Code: **CIS132L**

Assignment TOPIC: **STL in C++**

Submitted to-

Md. Faruk Hosen Sir

Lecturer

Department of Computing and Information

Submitted by-

Nusrat Zahan Eva

ID:0242320012091046

Batch: 18th-(B)

Semester: Spring 2024

Department of Computing and Information System

STL

STL, Standard template library, a set of C++ templates which provides pre-built implements of common data structures and algorithm.

✓ Why do we use STL?

In competitive programming STL is like the golden key, by using which we can not only save times but also code efficiently without making effort of writing repetitive codes.

STL have generic codes and pre-built functions of various data structures & algorithm. So using STL we can just get our work done by only calling the pre-built functions. We even use templates and re-use them when needed. So we save not only time but also effort of writing repetitive codes.

Thus by using STL we get our code done earlier and get the upperhand in limited time of Competitive programming.

Some of the primary components of STL are :-

i) Containers
ii) Algorithm
iii) Iterator

Containers : (Stores Data)

These are pre-built structures which store data (contain data). Every container has it's own characteristics. There are containers like---

Sequence Container : In which data are stored in linear manner . Some of these are Vector, List, Dequeue .

Derived Container : In which data are stored in some specialized manner . Some of these are Stack, Queue, Priority Queue.

Associative Container : In which data are stored in a manner so that we can get direct access of data . Some of these are Map, Multi-Map, Set , Multi-Set. Here operations like searching, insertion, deletion etc are relatively faster. But Random Access is slower.

Algorithm : (Process Data)

These are pre-built functions for operation like sort, search, count etc. It is kinda set of procedures that manipulates data by operating on them.

Iterators : (Points to Data instructed as Algorithm)

Actually iterators are objects points to the elements in container. It is handled like pointer but isn't pointer. Basically it connects container and algorithm. Iterator operates on the pointed elements of container based on algorithm (following a fit sequence of instructions).

✓ How STL works?

Let's store some data in a container like this and we want to sort them.

13	7	10	3

Here we can simply call STL function to complete the task.

If we do so then, iteration will point to the elements of the container and will operate on them following sorting algorithm.

Here is how STL works.

Vector

Vector is a sequence container in STL which behaves like a Dynamic Array as when we insert or delete elements it's automatically resize itself. This container store similar kind of data type .(contiguous memory allocation).

We declare vector using this syntax - vector <data type> vector_name.

We can also use auto which supports all data type but this doesn't support in Vector.

For example- vector <int> v.

If we use #include<iostream> as header file then we can only get access to standard input/output library of c++. So we have to include all c++ header file individually. Like to use vector we have to use #include<vector>.

art here | Vec_push_back.cpp

```
1  #include<iostream>
2  #include<vector>
3  using namespace std;
4  int main()
5  {
```

But if we use #include<bits/stdc++.h> then it includes all the header file of c++ already so we wouldn't need to include other header file individually.

```
here  Vec_push_back.cpp
1  #include<bits/stdc++.h>//every header file in C++ Available.
2  using namespace std;
3  int main()
4  {
5      vector<int>v;
```

Now let's do some operations on vector using built-in function of STL. Some operations are shown below~~~

push_back( ):

This function is used to put data in vector container like this-

Vector_name.push_back(value);

```
t here  Vec_push_back.cpp
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      vector<int>v;
6      v.push_back(1); //v[0]
7      v.push_back(2); //v[1]
8      v.push_back(3); //v[2]
9      v.push_back(4); //v[3]
10     v.push_back(5); //v[4]
11 }
```

As vector sequence container it stores data in linear manner like array .

Printing Vector's Value:

We can print vector simply like array like this -

```
art here  Vec_push_back.cpp
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      vector<int>v;
6      v.push_back(1); //v[0]
7      v.push_back(2); //v[1]
8      v.push_back(3); //v[2]
9      cout<<" "<< v[0]<<endl;
10     cout<<" "<< v[1]<<endl;
11     cout<<" "<< v[2]<<endl;
12 }
13
```

```
C:\Users\LENOVO\Desktop\Algo\Vec_push...
1
2
3

Process returned 0 (0x0)   execution time : 2.575 s
Press any key to continue.
```

But here if we give index number out of range we will get a garbage value-

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    v.push_back(1); //v[0]
    v.push_back(2); //v[1]
    cout<<" "<<v[0]<<endl;
    cout<<" "<<v[77]<<endl;

}
```

```
C:\Users\LENOVO\Desktop\Algo\Vec_push...
1
0

Process returned 0 (0x0)   execution time : 2.548 s
Press any key to continue.
```

We can also print vector using v.at( ) function like this-

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    v.push_back(1); //v[0]
    v.push_back(2); //v[1]
    v.push_back(3); //v[2]
    cout<<" "<<v.at(0)<<endl;
    cout<<" "<<v.at(1)<<endl;
    cout<<" "<<v.at(2)<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Vec_push...
1
2
3

Process returned 0 (0x0)   execution time : 4.664 s
Press any key to continue.
```

But using this function if we give index out of range it will show us out of range-

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    v.push_back(1); //v[0]
    v.push_back(2); //v[1]
    cout<<" "<<v.at(0)<<endl;
   cout<<" "<<v.at(77)<<endl;
}
```

```
Select C:\Users\LENOVO\Desktop\Algo\Vec_push_back.exe
1
terminate called after throwing an instance of 'std::out_of_range'
  what():  vector::_M_range_check: __n (which is 77) >= this->size() (which is 2)

Process returned 3 (0x3)   execution time : 8.027 s
Press any key to continue.
```

size( ): This function v.size( ) is see the size (total element number) of vector –

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    v.push_back(1); //v[0]
    v.push_back(2); //v[1]
    v.push_back(3); //v[2]
    v.push_back(4); //v[3]
    v.push_back(5); //v[4]
    cout<<"Size = "<<v.size()<<endl;

}
```

```
C:\Users\LENOVO\Desktop\Algo\Vec_pu...
Size = 5

Process returned 0 (0x0)   execution time : 2.179 s
Press any key to continue.
```

front( ) & back( ): This function v.front( ) is used to print front value stored in vector & This function v.back( ) is used to print last value stored in vector.

```
art here   Vec_push_back.cpp
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      vector<int>v;
6      int i;
7      v.push_back(3); //v[0]
8      v.push_back(0); //v[1]
9      v.push_back(8); //v[2]
10     v.push_back(9); //v[3]
11     v.push_back(5); //v[4]
12     // cout<<"Size = "<<v.size()<<endl;
13     for(i=0; i<v.size(); i++)
14     {
15         cout<<v.at(i)<<" ";
16     }
17
18     cout<<endl<<"Front Value = "<<v.front()<<endl;
19     cout<<"Back Value = "<<v.back()<<endl;
20 }
```

```
C:\Users\LENOVO\Desktop\Algo\Vec_pus...
3 0 8 9 5
Front Value = 3
Back Value = 5

Process returned 0 (0x0)   execution time : 2.511 s
Press any key to continue.
```

clear( ): This function v.clear() is used to clear all the data in vector.

```
rt here   Vector_2.cpp
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      vector<int>v;
6      int i;
7      v.push_back(3); //v[0]
8      v.push_back(0); //v[1]
9      v.push_back(8); //v[2]
10     v.push_back(9); //v[3]
11     v.push_back(5); //v[4]
12     // cout<<"Size = "<<v.size()<<endl;
13     for(i=0; i<v.size(); i++)
14     {
15         cout<<v.at(i)<<" ";
16     }
17     cout<<endl<<"Size Before Clearing The Vector = "<<v.size()<<endl;
18     v.clear();
19     cout<<"Size After Clearing The Vector = "<<v.size()<<endl;
20 }
21
```

```
C:\Users\LENOVO\Desktop\Algo\Vector_...
3 0 8 9 5
Size Before Clearing The Vector = 5
Size After Clearing The Vector = 0

Process returned 0 (0x0)   execution time : 2.302 s
Press any key to continue.
```

This also shows that it is like Dynamic array. Here we never declared array/vector size (like we used to do in static array) and when we are inserting and deleting values the vector is resizing on it's own.

That's how before clearing the vector we had 5 elements and after clearing it we got 0 (empty vector) element.

empty( ): This v.empty( ) function is used to check whether the vector is empty or not.

```
rt here   Vector_2.cpp
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      vector<int>v;
6      int i;
7      v.push_back(3); //v[0]
8      v.push_back(0); //v[1]
9      v.push_back(8); //v[2]
10     v.push_back(9); //v[3]
11     v.push_back(5); //v[4]
12     // cout<<"Size = "<<v.siz
13     for(i=0; i<v.size(); i++)
14     {
15         cout<<v.at(i)<<" ";
16     }
17     if(v.empty()) cout<<endl<< "Before-- " <<"It's Empty"<<endl;
18     else cout<<endl<< "Before-- " <<"It's NOT Empty"<<endl;
19     v.clear();
20     if(v.empty()) cout<<endl<< "After Clearing-- " <<"It's Empty"<<endl;
21     else cout<<endl<< "After Clearing-- " <<"It's NOT Empty"<<endl;
22
23 }
```

```
C:\Users\LENOVO\Desktop\Algo\Vector_2....
3 0 8 9 5
Before-- It's NOT Empty

After Clearing-- It's Empty

Process returned 0 (0x0)   execution time : 3.343 s
Press any key to continue.
```

Actually v.empty( ) function return a Boolean value (True or False) if it's empty which is true then if condition works otherwise else condition works, that's why we don't compare v.empty() to 0.

pop_back( ): This v. pop_back( ) function is used to delete last element .

```
rt here   Vector_2.cpp
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      vector<int>v;
6      int i;
7      v.push_back(3); //v[0]
8      v.push_back(0); //v[1]
9      v.push_back(8); //v[2]
10     v.push_back(9); //v[3]
11     v.push_back(5); //v[4]
12     cout<<"Initial Vector = ";
13     for(i=0; i<v.size(); i++)
14     {
15         cout<<v.at(i)<<" ";
16     }
17     cout<<endl;
18     v.pop_back();
19     cout<<"After Using pop_back(_) = ";
20     for(i=0; i<v.size(); i++)
21     {
22         cout<<v.at(i)<<" ";
23     }
24     cout<<endl;
25 }
```

```
C:\Users\LENOVO\Desktop\Algo\Vector...
Initial Vector = 3 0 8 9 5
After Using pop_back(_) = 3 0 8 9

Process returned 0 (0x0)   execution time : 2.798 s
Press any key to continue.
```

erase( ): This v.erase( ) function is used to delete pointed element .

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    int i;
    v.push_back(3); //v[0]
    v.push_back(0); //v[1]
    v.push_back(8); //v[2]
    v.push_back(9); //v[3]
    v.push_back(5); //v[4]
    cout<<"Initial Vector = ";
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl<< "After Erasing The Element of Index 2"<<endl<<" = ";
    v.erase(v.begin()+2);
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Vector_5.exe
Initial Vector = 3 0 8 9 5
After Erasing The Element of Index 2
 = 3 0 9 5

Process returned 0 (0x0)   execution time : 2.220 s
Press any key to continue.
```

Suppose we want to delete the value of Index 2 using this v.erase( ) function . So we will follow this syntax---

vector_name.erase(vector_name.begin()+Index_number)

Here begin( ) function points to the 1st element of the vector and we add the index of the value we want to delete with the 1st element so that like this 0+2 means v[2]=8 gets deleted.

If we want to erase from the pointed index to last index we use erase( ) like this-

```cpp
v.erase(v.begin()+2,v.end());
```

```
C:\Users\LENOVO\Desktop\Algo\Vector_6.exe
Initial Vector = 3 0 8 9 5
After Erasing The Element of Index 2
 = 3 0

Process returned 0 (0x0)   execution time : 2.361 s
Press any key to continue.
```

And if we want to delete pointed elements from the middle then we use it individually like this -

```
v.erase(v.begin()+2);
v.erase(v.begin()+2);
```

```
C:\Users\LENOVO\Desktop\Algo\Vector_...
Initial Vector = 3 0 8 9 5
After Erasing The Element of Index 2
 = 3 0 5

Process returned 0 (0x0)   execution time : 6.394 s
Press any key to continue.
```

Here we deleted v[2]=8 , after it the vector becomes 3 0 9 5 so now v[2] is 9 which was v[3] initially. So to delete 9 we pointed to v[2] as after erasing v[2]=8 v[2] becomes 9.

Or it can be done like this too using v.end( ) function

```
v.erase(v.begin()+2,v.end()-1);
```

```
C:\Users\LENOVO\Desktop\Algo\Vector_...
Initial Vector = 3 0 8 9 5
After Erasing The Element of Index 2
 = 3 0 5

Process returned 0 (0x0)   execution time : 6.394 s
Press any key to continue.
```

insert( ): This v. insert( ) function is used to insert values or elements in pointed index.

To enter value or values single time we use the function like this-

vector_name.insert(v.begin( )+ index_number, value/{value1,value2})

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    int i;
    v.push_back(3); //v[0]
    v.push_back(0); //v[1]
    v.push_back(8); //v[2]
    v.push_back(9); //v[3]
    v.push_back(5); //v[4]
    cout<<"Initial Vector = ";
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl<< "After Inserting An Element In Index 2"<<endl<<" = ";
    v.insert(v.begin()+2,700);
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Vector_7.exe
Initial Vector = 3 0 8 9 5
After Inserting An Element In Index 2
 = 3 0 700 8 9 5

Process returned 0 (0x0)   execution time : 5.420 s
Press any key to continue.
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    int i;
    v.push_back(3); //v[0]
    v.push_back(0); //v[1]
    v.push_back(8); //v[2]
    v.push_back(9); //v[3]
    v.push_back(5); //v[4]
    cout<<"Initial Vector = ";
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl<< "After Inserting Multiple Element In Index 2"<<endl<<" = ";
    v.insert(v.begin()+2,{700,200,100});
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Vector_7....
Initial Vector = 3 0 8 9 5
After Inserting Multiple Element In Index 2
 = 3 0 700 200 100 8 9 5

Process returned 0 (0x0)   execution time : 4.279 s
Press any key to continue.
```

To enter one value repeated times we use the function like this-

vector_name.insert(v.begin( ) + Index_Number, repeated_time,value);

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    int i;
    v.push_back(3); //v[0]
    v.push_back(0); //v[1]
    v.push_back(8); //v[2]
    v.push_back(9); //v[3]
    v.push_back(5); //v[4]
    cout<<"Initial Vector = ";
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl<< "After Inserting An Element 3 times In Index 0"<<endl<<" = ";
    v.insert(v.begin()+0,3,700);
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Vector_7.exe
Initial Vector = 3 0 8 9 5
After Inserting An Element 3 times In Index 0
 = 700 700 700 3 0 8 9 5

Process returned 0 (0x0)   execution time : 3.226 s
Press any key to continue.
```

swap( ): This swap ( ) function is used to swap values of two vectors (v1,v2)-

rt here | Vector_7.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v1,v2;
    int i;
    v1.push_back(1); //v1[0]
    v1.push_back(2); //v1[1]
    v1.push_back(3); //v1[2]
    v2.push_back(100); //v2[0]
    v2.push_back(200); //v2[1]
    v2.push_back(300); //v2[2]
    cout<<"Before Swapping = "<<endl;
    cout<<"v1 = ";
    for(i=0; i<v1.size(); i++)
    {
        cout<<v1.at(i)<<" ";
    }
    cout<<endl;
    cout<<"v2 = ";
    for(i=0; i<v2.size(); i++)
    {
        cout<<v2.at(i)<<" ";
    }
    cout<<endl;
    swap(v1,v2);
    cout<<"After Swapping = "<<endl;
    cout<<"v1 = ";
    for(i=0; i<v1.size(); i++)
    {
        cout<<v1.at(i)<<" ";
    }
    cout<<endl;
    cout<<"v2 = ";
    for(i=0; i<v2.size(); i++)
    {
        cout<<v2.at(i)<<" ";
    }
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\Vector_7....

```
Before Swapping =
v1 = 1 2 3
v2 = 100 200 300
After Swapping =
v1 = 100 200 300
v2 = 1 2 3

Process returned 0 (0x0)   execution time : 3.176 s
Press any key to continue.
```

sort( ): This v.sort ( ) function is used to sort values in vectors - (in ascending order always)

sort(v.begin( ), v.end( )) //sorting all the values from 1st to last

art here | *Vector_8.cpp | Vector_9.cpp | Vector_10.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    int i;
    v.push_back(3); //v[0]
    v.push_back(0); //v[1]
    v.push_back(9); //v[2]
    v.push_back(8); //v[3]
    v.push_back(5); //v[4]
    cout<<"Before Sorting = "<<endl;
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl;
    cout<<endl;
    sort(v.begin(),v.end());
    cout<<"After Sorting = "<<endl;
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl;
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\Vector_...

```
Before Sorting =
3 0 9 8 5

After Sorting =
0 3 5 8 9


Process returned 0 (0x0)   execution time : 2.358 s
Press any key to continue.
```

reverse( ): This v. reverse ( ) function is used to reverse the values in vector-

sort(v.begin( ), v.end( ))

here | Vector_10.cpp | *Untitled11 | *Untitled12

```cpp
int main()
{
    vector<int>v;
    int i;
    v.push_back(3); //v[0]
    v.push_back(0); //v[1]
    v.push_back(9); //v[2]
    v.push_back(8); //v[3]
    v.push_back(5); //v[4]
    cout<<"Before Reversing = "<<endl;
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl;
    cout<<endl;
    reverse(v.begin(),v.end());
    cout<<"After Reversing = "<<endl;
    for(i=0; i<v.size(); i++)
    {
        cout<<v.at(i)<<" ";
    }
    cout<<endl;
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\Vector...

```
Before Reversing =
3 0 9 8 5

After Reversing =
5 8 9 0 3


Process returned 0 (0x0)   execution time : 2.432 s
Press any key to continue.
```

iteration: It works like pointer . Iterator points to the values in vector container. As it operates on the elements by directly pointing them so it reduces time complexity –

we declare iterator like this – vector<data_type>::iterator iterator_name

& print like pointer-

here | Vector_10.cpp | *Untitled11 | *Untitled12

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<int>v;
    int i;
    v.push_back(3); //v[0]
    v.push_back(0); //v[1]
    v.push_back(9); //v[2]
    v.push_back(8); //v[3]
    v.push_back(5); //v[4]
    vector<int>::iterator vit;
    vit=v.begin()+2;
    //keeping v[2]=9 in vit iterator
    cout<<"  "<<*vit<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\Vector_...

```
 9


Process returned 0 (0x0)   execution time : 2.686 s
Press any key to continue.
```

Stack

Stack is a container that stores data in linear motion and for output follows LIFO (last in first out) principle. Here Data insertion and deletion is done from the one end known as the top of the stack.

We declare stack using this syntax - stack <data type> stack_name.

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    stack<int>th;
}
```

Now let's do some operations on stack using built-in function of STL. Some operations are shown below~~~

push( ): This th.push( ) function is used to input data in stack container like this-

stack_name.push(value);

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    stack<int>th;
    int i;
    th.push(7);//top=0--th[0]=7
    th.push(15);//top=1--th[1]=15
    th.push(13);//top=2--th[2]=13
    th.push(3);//top=3--th[3]=3
    th.push(8);//top=4--th[4]=8

}
```

empty( ): This th.empty( ) function is used to check whether the stack is empty or not.

Here we already put some data so it's not empty we know . Now let's test it with the function-

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    stack<int>th;
    int i;
    th.push(7);//top=0--th[0]=7
    th.push(15);//top=1--th[1]=15
    th.push(13);//top=2--th[2]=13
    th.push(3);//top=3--th[3]=3
    th.push(8);//top=4--th[4]=8

     if(th.empty()) cout<<"It's Empty"<<endl;
    else cout<<"It's NOT Empty"<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Stack_1.e...
It's NOT Empty

Process returned 0 (0x0)   execution time : 2.822 s
Press any key to continue.
```

but Now see this-

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    stack<int>th;
    int i;
   // th.push(7);//top=0--th[0]=7
   // th.push(15);//top=1--th[1]=15
   // th.push(13);//top=2--th[2]=13
   // th.push(3);//top=3--th[3]=3
   // th.push(8);//top=4--th[4]=8

     if(th.empty()) cout<<"It's Empty"<<endl;
    else cout<<"It's NOT Empty"<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Stack_1....
It's Empty

Process returned 0 (0x0)   execution time : 2.581 s
Press any key to continue.
```

we didn't put any value in stack so we are getting out that " It's Empty".

Printing stack's Value:

We print stack like this---

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    stack<int>th;
    int i;
    th.push(7);//top=0--th[0]=7
    th.push(15);//top=1--th[1]=15
    th.push(13);//top=2--th[2]=13
    th.push(3);//top=3--th[3]=3
    th.push(8);//top=4--th[4]=8
    cout<<"The Stack is = " << endl<<" ";
    while (!th.empty())
    {
        cout << th.top() << " ";
        th.pop();
    }
    /*th.empty=True top=4 th[4]=8 th.pop top=3
    th.empty=True top=3 th[3]=3 th.pop top=2
    th.empty=True top=2 th[2]=13 th.pop top=1
    th.empty=True top=1 th[1]=15 th.pop top=0
    th.empty=True top=0 th[0]=7 th.pop top=-1
    Means stack Empty th.empty=False*/
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Stack_3.exe
The Stack is =
 8 3 13 15 7

Process returned 0 (0x0)   execution time : 2.400 s
Press any key to continue.
```

In this process we print top value and pop(delete) it then print another top value till our stack is NOT empty. That's why we use empty function in a loop to check it again & again to see whether the stack is empty and if it is then The loop will break.

As stack's input follow LIFO (last in first out) principle so we got reverse sequence of our input

Here last input was 8 which became first output then after deleting it out last input becomes 3 which become our next output. Like this we printed our stack !!! // condition can be while(th.size( ) >0)

size( ): This function th.size( ) is see the size (total element number) of stack –

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    stack<int>th;
    int i;
    th.push(7);//top=0--th[0]=7
    th.push(15);//top=1--th[1]=15
    th.push(13);//top=2--th[2]=13
    th.push(3);//top=3--th[3]=3
    th.push(8);//top=4--th[4]=8

    cout<<"The Size of The Stack is = "<<th.size()<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Stack_2.e...
The Size of The Stack is = 5

Process returned 0 (0x0)   execution time : 5.268 s
Press any key to continue.
```

top( ):

we know that data insertion and deletion is done from the one end known as the top of the stack. And as it follows last in first out rule. Then the last input will be our top.

Using th.top( ) function we can directly see what is our top value like this-
--

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    stack<int>th;
    int i;
    th.push(7);//top=0--th[0]=7
    th.push(15);//top=1--th[1]=15
    th.push(13);//top=2--th[2]=13
    th.push(3);//top=3--th[3]=3
    th.push(8);//top=4--th[4]=8
    cout<<"In The Stack Top Value is = " << th.top()<<endl<<" ";
    cout<<endl;
}
```

```
In The Stack Top Value is = 8


Process returned 0 (0x0)   execution time : 3.325 s
Press any key to continue.
```

pop( ):

This function is used to delete top. As stack follows LIFO principle so Top is the Last Input & when we use pop it gets deleted then the top becomes the 2nd input from the last. Like this---

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    stack<int>th;
    int i;
    th.push(7);//top=0--th[0]=7
    th.push(15);//top=1--th[1]=15
    th.push(13);//top=2--th[2]=13
    th.push(3);//top=3--th[3]=3
    th.push(8);//top=4--th[4]=8
    cout<<"Initially The Top Value is = " << th.top()<<endl<<" ";
    th.pop();
    cout<<"After Popping Now Top Value is = " << th.top()<<endl<<" ";
    cout<<endl;
}
```

```
Initially The Top Value is = 8
 After Popping Now Top Value is = 3


Process returned 0 (0x0)   execution time : 5.506 s
Press any key to continue.
```

Here the Top was 8 then we used pop() so Top became the 2nd input from the last which is 3.

Queue

Queue is a container that stores data in linear motion and for output follows FIFO (fisrt in first out) principle. Here Data insertion is done at one end called REAR and deletion is at another end called FRONT.

We declare queue using this syntax - queue <data type> queue_name.

```
#include<bits/stdc++.h>
int main()
{
    queue<int>kv;
}
```

Now let's do some operations on queue using built-in function of STL. Some operations are shown below~~~

push( ): This kv.push( ) function is used to input data in queue container like this-

```
#include<bits/stdc++.h>
int main()
{
    queue<int>kv;
    kv.push(10);
    kv.push(20);
    kv.push(30);
    kv.push(40);
    kv.push(50);
}
```

size( ): This kv.size( ) function is used to is see the size (total element number) of queue.

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    queue<int>kv;
    kv.push(10);//front=0 rear=0 - kv[0]=10
    kv.push(20);//front=0 rear=1 - kv[1]=20
    kv.push(30);//front=0 rear=2 - kv[2]=30
    kv.push(40);//front=0 rear=3 - kv[3]=40
    kv.push(50);//front=0 rear=4 - kv[4]=50

    cout<<" The Size Of Queue Is = "<<kv.size()<< endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Queue_1....
 The Size Of Queue Is = 5

Process returned 0 (0x0)   execution time : 5.346 s
Press any key to continue.
```

empty( ): This kv.empty( ) function is used to check whether the queue is empty or not.

Here we already put some data so it's not empty we know . Now let's test it with the function-

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    queue<int>kv;
    kv.push(10);//front=0 rear=0 - kv[0]=10
    kv.push(20);//front=0 rear=1 - kv[1]=20
    kv.push(30);//front=0 rear=2 - kv[2]=30
    kv.push(40);//front=0 rear=3 - kv[3]=40
    kv.push(50);//front=0 rear=4 - kv[4]=50
//    cout<<" The Size Of Queue Is = "<<kv.size()<< endl;
    if(kv.empty()) cout<<endl<<" It's EMPTY!!! " << endl;
    else cout<<endl<<" It's NOT EMPTY!!! " << endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Queue_1.exe

 It's NOT EMPTY!!!

Process returned 0 (0x0)   execution time : 5.105 s
Press any key to continue.
```

but Now Let's see here-

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    queue<int>kv;
    //kv.push(10);//front
   // kv.push(20);//front
  //  kv.push(30);//front
  //  kv.push(40);//front
   // kv.push(50);//front
//    cout<<" The Size Of Queue Is = "<<kv.size()<< endl;
    if(kv.empty()) cout<<endl<<" It's EMPTY!!! " << endl;
    else cout<<endl<<" It's NOT EMPTY!!! " << endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Queue_...

 It's EMPTY!!!

Process returned 0 (0x0)   execution time : 2.818 s
Press any key to continue.
```

we didn't put any value in queue so we are getting --- " It's EMPTY!!! "

front( ) & back( ):

Queue is a container that follows FIFO (first in first out) principle. It means the 1st(oldest) input will be the 1st output like taking movies ticket from the line. So front( ) refers to the oldest element of the queue & back( ) refers to the newest element of the queue ---

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    queue<int>kv;
    kv.push(10);//front=0 rear=0 - kv[0]=10
    kv.push(20);//front=0 rear=1 - kv[1]=20
    kv.push(30);//front=0 rear=2 - kv[2]=30
    kv.push(40);//front=0 rear=3 - kv[3]=40
    kv.push(50);//front=0 rear=4 - kv[4]=50
    cout<<" The Front Of The Queue Is = "<<kv.front()<<endl;
    cout<<" The Back Of The Queue Is = "<<kv.back()<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Queue_2....
 The Front Of The Queue Is = 10
 The Back Of The Queue Is = 50

Process returned 0 (0x0)   execution time : 8.172 s
Press any key to continue.
```

Here the oldest was 10 which is Top & the newest was 50 which is the back.

pop( ): This pop( ) function is used to delete value of queue. As queue follows FIFO (first in first out) principle, the oldest input will be the latest output. So if we use pop( ) to delete value of queue the oldest value will be deleted. Here we used pop( )---

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    queue<int>kv;
    kv.push(10);//front=0 rear=0 - kv[0]=10
    kv.push(20);//front=0 rear=1 - kv[1]=20
    kv.push(30);//front=0 rear=2 - kv[2]=30
    kv.push(40);//front=0 rear=3 - kv[3]=40
    kv.push(50);//front=0 rear=4 - kv[4]=50
    cout<<"Initially ~~~"<<endl<<"The Front Of The Queue Is = "<<kv.front()<< endl;
    kv.pop();
    cout<<"After Popping~~~"<<endl<<"The Front Of The Queue Is = "<<kv.front()<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Queue_...
Initially ~~~
The Front Of The Queue Is = 10
After Popping~~~
The Front Of The Queue Is = 20

Process returned 0 (0x0)   execution time : 5.356 s
Press any key to continue.
```

Initially the front was the oldest value of queue which is 10 then we used pop( ) so front =10 was deleted and the front become 2nd oldest value which is 20.

Printing queue's Value: We print queue like this~~~

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    queue<int>kv;
    kv.push(10);//front=0 rear=0 - kv[0]=10
    kv.push(20);//front=0 rear=1 - kv[1]=20
    kv.push(30);//front=0 rear=2 - kv[2]=30
    kv.push(40);//front=0 rear=3 - kv[3]=40
    kv.push(50);//front=0 rear=4 - kv[4]=50
    cout<<"The Queue Is = "<< endl << " ";
    while(kv.size()>0)
    {
        cout<<kv.front()<< " ";
        kv.pop();
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Queue_5....
The Queue Is =
 10 20 30 40 50

Process returned 0 (0x0)   execution time : 2.411 s
Press any key to continue.
```

In this process we print front value and pop(delete) it then print another front value till our queue's element is greater 0 this means till we have elements in queue. That's why we use size function in a loop to check element number and when size will be 0 means there will be no element left then the loop will break.

Here oldest input was 10 which became first output then after deleting it out the 2nd oldest 20 became our next output. Like this we printed our queue!!! // condition can be while(!kv.empty( ))

emplace( ): This kv.emplace( ) function is same as kv.push( ) function which is used to input data in queue container like this-

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    queue<int>kv;
    kv.emplace(10);//front=0 rear=0 - kv[0]=10
    kv.emplace(20);//front=0 rear=1 - kv[1]=20
    kv.emplace(30);//front=0 rear=2 - kv[2]=30
    kv.emplace(40);//front=0 rear=3 - kv[3]=40
    kv.emplace(50);//front=0 rear=4 - kv[4]=50
    cout<<"The Queue Is = "<< endl << " ";
    while(kv.size()>0)
    {
        cout<<kv.front()<< " ";
        kv.pop();
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Queue_5....
The Queue Is =
 10 20 30 40 50

Process returned 0 (0x0)   execution time : 2.959 s
Press any key to continue.
```

swap( ): This kv.swap ( ) function is used to swap the values of two queues ---

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    queue<int>kv,kt;
    kv.emplace(100);//front=0 rear=0 - kv[0]=100
    kv.emplace(200);//front=0 rear=1 - kv[1]=200
    kv.emplace(300);//front=0 rear=2 - kv[2]=300
    kt.emplace(1);//front=0 rear=0 - kt[0]=1
    kt.emplace(2);//front=0 rear=1 - kt[1]=2
    kt.emplace(3);//front=0 rear=2 - kt[2]=3
    cout<<"Before Swapping ="<< endl ;
    cout<<"The kv Queue Is = ";
    queue<int>temp1=kv;
//used temp1 queue to save that old queue to print later
    while(temp1.size()>0)
    {
        cout<<temp1.front()<< " ";
        temp1.pop();
    }
    cout<<endl;
    cout<<"The kt Queue Is = ";
    queue<int>temp2=kt;
    while(temp2.size()>0)
    {
        cout<<temp2.front()<< " ";
        temp2.pop();
    }
    cout<<endl;
    kt.swap(kv);
    cout<<endl<<"After Swapping ="<< endl ;
    cout<<"The kv Queue Is = ";
    while(kv.size()>0)
    {
        cout<<kv.front()<< " ";
        kv.pop();
    }
    cout<<endl;
    cout<<"The kt Queue Is = ";
    while(kt.size()>0)
    {
        cout<<kt.front()<< " ";
        kt.pop();
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\Queue_...
Before Swapping =
The kv Queue Is = 100 200 300
The kt Queue Is = 1 2 3

After Swapping =
The kv Queue Is = 1 2 3
The kt Queue Is = 100 200 300

Process returned 0 (0x0)   execution time : 2.284 s
Press any key to continue.
```

List

List is container in STL which allows non-contiguous memory allocation. So operations like deletion, insertion is easier in list. The inserted elements are stored in nodes of list. Node is made of values and pointers pointed to previous or next node or both. It also behaves like dynamic memory which resizes itself while being operated.

We declare list using this syntax - list <data type> list_name.

Like this~~~

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;

}
```

Now let's do some operations on list using built-in function of STL. Some operations are shown below~~~

push_back( ):

This function is used to put data in list container like this-

List_name.push_back(value);

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
}
```

Using this function we push one element after another or behind their back. (here output 1 2 3)

Printing List's Value:

We print list using iteration usually---

Iterator is declared like this-

List_name<data_type>::iterator iterator_name;

& is printed like this-

here thv.begin( ) points to the 1st element of list and thv.end( ) points to end after last element .

here | List_1.cpp | List_2.cpp | List_3.cpp | List_4.cpp | List_5.cpp | *Untitled7

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    list<int>::iterator it;
    cout<<"This Is The List = "<<endl;
    for(it=thv.begin();it!=thv.end();it++)
    {
        cout<<*it<<" ";
    }
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\List_1.ex...

```
This Is The List =
1 2 3

Process returned 0 (0x0)   execution time : 2.671 s
Press any key to continue.
```

But we can also print list using auto iterator like this-

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    list<int>::iterator it;
    cout<<"This Is The List = "<<endl;
    /*for(it=thv.begin()
    {
        cout<<*it<<" ";
    }*/
    for(auto it:thv)
    {
        cout<<it<< " ";
    }
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\List_1....

```
This Is The List =
1 2 3

Process returned 0 (0x0)   execution time : 5.286 s

Press any key to continue.
```

push_front( ):

This function is used to put data in list container like this-

List_name.push_front(value);

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    thv.push_front(7);
    thv.push_front(8);
    thv.push_front(9);
    list<int>::iterator it;
    cout<<"This Is The List = "<<endl;
    /*for(it=thv.begin(
    {
        cout<<*it<<" ";
    }*/
    for(auto it:thv)
    {
        cout<<it<< " ";
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_1.ex...
This Is The List =
9 8 7 1 2 3

Process returned 0 (0x0)   execution time : 5.442 s
Press any key to continue.
```

Using this function we push one element before another or in front of another element.

front( ) & back( ): This function List_name.front(value) is used to print front value stored in list & This function List_name.back(value) is used to print last value stored in list.

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    thv.push_back(7);
    thv.push_back(8);
    cout<<"This Is The List = "<<endl;
    for(auto it:thv)
    {
        cout<<it<< " ";
    }
    cout<<endl;
    cout<<"Front Value of The List Is = "<<thv.front()<<endl;
    cout<<"Back Value of The List Is = "<<thv.back()<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_2.exe
This Is The List =
1 2 3 7 8
Front Value of The List Is = 1
Back Value of The List Is = 8

Process returned 0 (0x0)   execution time : 2.388 s
Press any key to continue.
```

pop_front( ) & pop_back( ): This function List_name.pop_front(value) is used to delete value of list from the front and This function List_name.pop_back(value) is used to delete value of list from the back like this---

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    thv.push_back(7);
    thv.push_back(8);
    cout<<"Initially:-"<<endl<<"This Is The List = "<<endl;
    for(auto it:thv) cout<<it<< " ";
    cout<<endl;
    cout<<"Front Value of The List Is = "<<thv.front()<<endl;
    cout<<"Back Value of The List Is = "<<thv.back()<<endl;
    thv.pop_back();
    cout<<"After Applying pop.back(_) = "<<thv.back()<<endl;
    thv.pop_front();
    cout<<"After Applying pop.front(_) = "<<thv.front()<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_2.exe
Initially:-
This Is The List =
1 2 3 7 8
Front Value of The List Is = 1
Back Value of The List Is = 8
After Applying pop.back(_) = 7
After Applying pop.front(_) = 2

Process returned 0 (0x0)   execution time : 5.694 s
Press any key to continue.
```

Here from initial the list from was 1 and back was 8. After applying pop.back( ) (deleting back value of list) we got new back 7 (which is 2nd from the back). After applying pop.front( ) (deleting front value of list) we got new front 2 (which is 2nd from the front).

size( ): This function list_name.size(value) is used to see the size (total element number) of list -

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    thv.push_back(7);
    thv.push_back(8);
    cout<<"This Is The List = "<<endl;
    for(auto it:thv) cout<<it<< " ";
    cout<<endl;
    cout<< "so, The Size Of The List Is = "<<thv.size()<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_3.exe
This Is The List =
1 2 3 7 8
so, The Size Of The List Is = 5

Process returned 0 (0x0)   execution time : 5.348 s
Press any key to continue.
```

clear( ): This function list_name.clear(value) is used to clear all the data of list -

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    thv.push_back(7);
    thv.push_back(8);
    cout<<"This Is The List = "<<endl;
    for(auto it:thv) cout<<it<< " ";
    cout<<endl;
    cout<< "Initially, The Size Of The List Is = "<<thv.size()<<endl;
    thv.clear();
    cout<< "After Clearing The List The Size Of It = "<<thv.size()<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_3.exe
This Is The List =
1 2 3 7 8
Initially, The Size Of The List Is = 5
After Clearing The List The Size Of It = 0

Process returned 0 (0x0)   execution time : 2.497 s
Press any key to continue.
```

0 element means there is no element is the List.

empty( ): This list_name.empty(value) function is used to check whether the stack is empty or not.

Here we already put some data so it's not empty we know. Now let's test it with the function-

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    thv.push_back(7);
    thv.push_back(8);
    if(thv.empty()) cout<<"The List Is Empty"<<endl;
    else cout<<"The list Is NOT Empty"<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_4.e...
The list Is NOT Empty

Process returned 0 (0x0)   execution time : 2.461 s
Press any key to continue.
```

but Now Let's see here-

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
  //  thv.push_back(1);
  //  thv.push_back(2);
  //  thv.push_back(3);
  //  thv.push_back(7);
  //  thv.push_back(8);
    if(thv.empty()) cout<<"The List Is Empty"<<endl;
    else cout<<"The list Is NOT Empty"<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_4.e...
The List Is Empty

Process returned 0 (0x0)   execution time : 5.222 s
Press any key to continue.
```

we didn't put any value in queue so we are getting --- " The list is empty"

insert( ): This list_name. insert( ) function is used to insert values or elements in pointed index.

As we use pointers like iterator to point values in List, we have to use iterator to insert the value in the List. 1st we have to declare a iterator. then using that iterator we have to point the 1st value of list using thv.begin( ) function. Then we have to use advance(iterator, index) function to update the iterator that will point to the new position_index. then using list_name.insert(iterator, value) function we will insert the value we want to insert…like this-

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    thv.push_back(7);
    thv.push_back(8);
    cout<<"Initial list = "<<endl;
    for(auto v:thv)
    {
        cout<<v<<" ";
    }
    cout<<endl;
    //iterator declaration
    list<int>::iterator me;
    me=thv.begin();
    advance(me,3);
    // here 3 is the 3rd position
    thv.insert(me,100);
    cout<<"After entering 100 in 3rd index of List = "<<endl;
    for(auto v:thv)
    {
        cout<<v<<" ";
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_5.ex...
Initial list =
1 2 3 7 8
After entering 100 in 3rd index of List =
1 2 3 100 7 8

Process returned 0 (0x0)   execution time : 2.979 s
Press any key to continue.
```

To insert same values in multiple times we use this syntax list_name.insert(iterator, times, value) like this---

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    thv.push_back(7);
    thv.push_back(8);
    cout<<"Initial list = "<<endl;
    for(auto v:thv)
    {
        cout<<v<<" ";
    }
    cout<<endl;
    //iterator declaration
    list<int>::iterator me;
    me=thv.begin();
    advance(me,3);
    // here 3 is the 3rd position
    thv.insert(me,2,100);
    cout<<"After entering 100 in 3rd index of List 2 times= "<<endl;
    for(auto v:thv)
    {
        cout<<v<<" ";
    }
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_5.e...
Initial list =
1 2 3 7 8
After entering 100 in 3rd index of List 2 times=
1 2 3 100 100 7 8

Process returned 0 (0x0)   execution time : 9.473 s
Press any key to continue.
```

If we want to insert in 1st position= 0 index of list then we don't need to use advance function as we are not in need of updating the position of begin( ) value.

erase( ): This list_name. erase( ) function is used to erase values or elements in pointed index.

Like insert( ) function we have to use iterator to insert the value in the List. 1st we have to declare a iterator. then using that iterator we have to point the 1st value of list using thv.begin( ) function. Then we have to use advance(iterator, index) function to update the iterator that will point to the new position_index. then using list_name.erase (iterator) function we will erase the value we want to erase…like this-

(for single value)

here | List_6.cpp | List_7.CPP

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv;
    thv.push_back(1);
    thv.push_back(2);
    thv.push_back(3);
    thv.push_back(7);
    thv.push_back(8);
    cout<<"Initial list = "<<endl;
    for(auto v:thv)
    {
        cout<<v<<" ";
    }
    cout<<endl;
    //iterator declaration
    list<int>::iterator me;
    me=thv.begin();
    advance(me,3);
    // here 3 is the 3rd position
    thv.erase(me);
    cout<<"After erasing 3rd index of List= "<<endl;
    for(auto v:thv)
    {
        cout<<v<<" ";
    }
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\List_6.e...

```
Initial list =
1 2 3 7 8
After erasing 3rd index of List=
1 2 3 8

Process returned 0 (0x0)   execution time : 9.100 s
Press any key to continue.
```

(for multiple value)

We take two iterator to point from which index to which index we want to erase and follow this syntax list_name.erase (iterator1, iterator2) like this-

here List_6.cpp List_7.CPP

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv={1,2,3,7,8};
    cout<<"Initial list = "<<endl;
    for(auto v:thv)
    {
        cout<<v<<" ";
    }
    cout<<endl;
    //iterator declaration
    list<int>::iterator me,me1;
    me1=thv.begin();
    advance(me1,3);
    me=thv.begin();
    // here 3 is the 3rd position
    thv.erase(me,me1);
    //erasing 3 elements from the front
    cout<<"After erasing the 1st 3 Values of List= "<<endl;
    for(auto v:thv)
    {
        cout<<v<<" ";
    }
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\List_6.e...

```
Initial list =
1 2 3 7 8
After erasing the 1st 3 Values of List=
7 8

Process returned 0 (0x0)   execution time : 3.771 s
Press any key to continue.
```

remove( ): We use list_name.remove (value) to remove specific values from the list. Even if that specific value is multiple time in that list this function will remove the---

here List_7.CPP List_9.cpp List_8.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv={1,2,3,7,8,7,7};
    cout<<"Initial list = "<<endl;
    for(auto v:thv) cout<<v<<" ";
    cout<<endl;
    thv.remove(7);
    cout<<"After removing 7 from the List= "<<endl;
    for(auto v:thv) cout<<v<<" ";
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\List_7.exe

```
Initial list =
1 2 3 7 8 7 7
After removing 7 from the List=
1 2 3 8

Process returned 0 (0x0)   execution time : 4.217 s
Press any key to continue.
```

reverse( ): This list_name.reverse( ) function is used to reverse the values of list –

here List_7.CPP List_9.cpp List_8.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv={1,2,3,7,8,7,7};
    cout<<"Initial list = "<<endl;
    for(auto v:thv) cout<<v<<" ";
    cout<<endl;
    thv.remove(7);
    cout<<"After removing 7 from the List = "<<endl;
    for(auto v:thv) cout<<v<<" ";
    cout<<endl;
    thv.reverse();
    cout<<"After Reversing The List = "<<endl;
    for(auto v:thv) cout<<v<<" ";
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\List_7.exe

```
Initial list =
1 2 3 7 8 7 7
After removing 7 from the List =
1 2 3 8
After Reversing The List =
8 3 2 1

Process returned 0 (0x0)   execution time : 6.772 s
Press any key to continue.
```

sort( ): This list_name.sort( ) function is used to sort values in list – (in ascending order always)

here List_8.cpp List_9.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv={1,0,5,22,2,100,0};
    cout<<"Initial list = "<<endl;
    for(auto v:thv) cout<<v<<" ";
    cout<<endl;
    thv.sort();
    cout<<"After Sorting The List = "<<endl;
    for(auto v:thv) cout<<v<<" ";
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\List_8.exe

```
Initial list =
1 0 5 22 2 100 0
After Sorting The List =
0 0 1 2 5 22 100

Process returned 0 (0x0)   execution time : 7.750 s
Press any key to continue.
```

unique( ): This list_name.unique( ) function is used to replace all the duplicate constructive elements with single element---

here List_9.cpp *Untitled11

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv={5,6,9,9,8,8,8,7,2,1,9,9,9,9};
    cout<<"Initial list = "<<endl;
    for(auto v:thv) cout<<v<<" ";
    cout<<endl;
    thv.unique();
    cout<<"After applying unique() to The List = "<<endl;
    for(auto v:thv) cout<<v<<" ";
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\List_9.e...

```
Initial list =
5 6 9 9 8 8 8 7 2 1 9 9 9 9
After applying unique() to The List =
5 6 9 8 7 2 1 9

Process returned 0 (0x0)   execution time : 3.450 s
Press any key to continue.
```

swap( ): This list_name1.swap(list_name2) function is used to swap the elements of two list_name1 & list_name2 list---

t here List_10.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv1={50,60,70,80,90};
    list<int>thv2={5000,6000,7000,8000,9000};
    cout<<"Initially :- "<<endl<<"thv1 = ";
    for(auto v:thv1) cout<<v<<" ";
    cout<<endl<<"thv2 = ";
    for(auto v:thv2) cout<<v<<" ";
    cout<<endl;
    thv1.swap(thv2);
    cout<<"After :- "<<endl<<"thv1 = ";
    for(auto v:thv1) cout<<v<<" ";
    cout<<endl<<"thv2 = ";
    for(auto v:thv2) cout<<v<<" ";
    cout<<endl;
}
```

C:\Users\LENOVO\Desktop\Algo\List_10.exe

```
Initially :-
thv1 = 50 60 70 80 90
thv2 = 5000 6000 7000 8000 9000
After :-
thv1 = 5000 6000 7000 8000 9000
thv2 = 50 60 70 80 90

Process returned 0 (0x0)   execution time : 3.778 s
Press any key to continue.
```

merge( ): This list_name1.swap(list_name2) function is used to swap the elements of two list_name1 & list_name2 list---

```
List_11.cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    list<int>thv1={50,60,70,80,90};
    list<int>thv2={5000,6000,7000,8000,9000};
    cout<<"Initially :- "<<endl<<"thv1 = ";
    for(auto v:thv1) cout<<v<<" ";
    cout<<endl<<"thv2 = ";
    for(auto v:thv2) cout<<v<<" ";
    cout<<endl;
    thv1.merge(thv2);
    cout<<"After Merging :- "<<endl<<"thv1 = ";
    for(auto v:thv1) cout<<v<<" ";
    cout<<endl<<"thv2 = ";
    for(auto v:thv2) cout<<v<<" ";
    cout<<endl;
}
```

```
C:\Users\LENOVO\Desktop\Algo\List_11.exe
Initially :-
thv1 = 50 60 70 80 90
thv2 = 5000 6000 7000 8000 9000
After Merging :-
thv1 = 50 60 70 80 90 5000 6000 7000 8000 9000
thv2 =

Process returned 0 (0x0)   execution time : 2.716 s
Press any key to continue.
```

the merged list remains in list 1 while list 2 becomes empty.