STL cont.

Today’s Agenda

Today we are going to cover -

- Iterators
- Container - List.
- Algorithms

Let’s Get Started-

Recap

C++ Standard Template Library consists of three well-structured components −

1. Containers
2. Algorithms
3. Iterators

Iterators

Iterators are used to point to the containers in STL, because of iterators it is possible for an algorithm to manipulate different types of data structures/Containers.

Defining an Iterator in STL

Syntax for defining an iterator is :

container_type <parameter_list>::iterator iterator_name;

Iterators

```
#include<vector>

using namespace std;

int main()
{
   vector<int>::iterator i;
   /* create an iterator named i to a vector of integers */

   vector<string>::iterator j;
   /* create an iterator named j to a vector of strings */

   list<int>::iterator k;
   /* create an iterator named k to a vector of integers */
 }
```

Iterators

Iterators can be used to traverse the container, and we can de-reference the iterator to get the value of the element it is pointing to.

```
#include<iostream>
#include<vector>
int main()
{
   vector<int> v(10);
   /* creates an vector v : 0,0,0,0,0,0,0,0,0,0  */
   vector<int>::iterator i;
   for(i = v.begin(); i!= v.end(); i++)
   cout << *i <<"  ";
   /* in the above for loop iterator i iterates though the
   vector v and *operator is used of printing the element
   pointed by it.  */
}
```

Operations on Iterators

Following are the operations that can be used with Iterators to perform various actions.

Advance

distance

next

prev

begin

end

Operations on Iterators: advance()

Advance: It will increment the iterator i by the value of the distance. If the value of distance is negative, then iterator will be decremented.

SYNTAX: advance(iterator i ,int distance)

Consider following program:

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
   vector<int>  v {1,2,3,4,5,6,7,8,9,10};    // create a vector of 10 0's
   vector<int>::iterator i;  // defines an iterator i to the vector of integers
```

Operations on Iterators: advance()

```
    i = v.begin();
    cout<<" Value at i " <<*i<<endl;
    /* i now points to the beginning of the vector v */

    advance(i,5);
    cout<<" Value at i " <<*i<<endl;
    /* i now points to the fifth element form the
    beginning of the vector v */

    advance(i,-1);
    cout<<" Value at i " <<*i<<endl;
    /* i  now points to the fourth element from the
    beginning of the vector */
}
```

Operations on Iterators: distance()

distance() Operation : It will return the number of elements or we can say distance between the first and the last iterator.
SYNTAX: distance(iterator first, iterator last)

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
   vector<int>  v(10) ;    // create a vector of 10 0's
   vector<int>::iterator i, j;  // defines iterators i,j to the vector of integers
   i = v.begin();     /* i now points to the beginning of the vector v */
   j = v.end();     /* j now points to the end() of the vector v */
   cout << distance(i,j) << endl;
   /* prints 10 , */
}
```

Operations on Iterators

next() Operation: It will return the nth iterator to i, i.e iterator pointing to the nth element from the element pointed by i.
SYNTAX: next(iterator i ,int n)

prev() Operation :It will return the nth predecessor to i, i.e iterator pointing to the nth predecessor element from the element pointed by i.
SYNTAX: prev(iterator i, int n)

begin() Operation: This method returns an iterator to the start of the given container.
SYNTAX: begin()

end() Operation: This method returns an iterator to the end of the given container.
SYNTAX: end()

Operations on Iterators : begin(), end()

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{

   vector<int>  v{1,2,3,4,5,6,7,8,9,10};    // create a vector of 10 0's
   vector<int>::iterator i;  // defines an iterator i to the vector of integers

   for( i = v.begin(); i != v.end(); i++)
   {
        cout << *i <<" ";   // for printing the vector
   }

}
```

Operations on Iterators : prev(), next()

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{

   vector<int>  v{1,2,3,4,5,6,7,8,9,10};    // create a vector of 10 0's
   vector<int>::iterator i;  // defines an iterator i to the vector of integers
   i=v.begin();
   cout<<"Currently at " <<*i<<endl; //prints 1
   cout<<"Now at " << *next(i ,5)<<endl;  //prints 6
   i=v.end()-1; //v.end() gives end of the vector. -1 will take you to last element in vector
   cout<<"Currently at " <<*i<<endl; //prints 10
   cout<<"Now at " << *prev(i ,7)<<endl;  //prints 3
}
```

MCQ

Which pair of functions are not available with iterators?

1. Front, Back
2. Next,Prev
3. Advance, Distance
4. Begin , end

MCQ

Which pair of functions are not available with iterators?

1. Front, Back
2. Next,Prev
3. Advance, Distance
4. Begin , end

MCQ

Which of the following is false in case of iterators

1. Distance function will return number of elements between first and last iterator
2. End will always take you to the last element in the container (vector, list)
3. Iterators help you to traverse through the list or vector
4. Iterators can help you to de-reference the container elements

MCQ

Which of the following is false in case of iterators

1. Distance function will return number of elements between first and last iterator
2. End will always take you to the last element in the container (vector, list)
3. Iterators help you to traverse through the list or vector
4. Iterators can help you to de-reference the container elements

Assignment

Create a vector of 10 integers
Apply following functions on it and observe the output by printing the contents.

1. begin
2. end
3. next
4. prev
5. distance
6. advance

Container - List

Array and Vector are contiguous containers, i.e they store their data on continuous memory

The the insert operation at the middle of vector/array is very costly (in terms of number of operation and process time) because we have to shift all the elements, linked list overcome this problem.

Linked list can be implemented by using the list container.

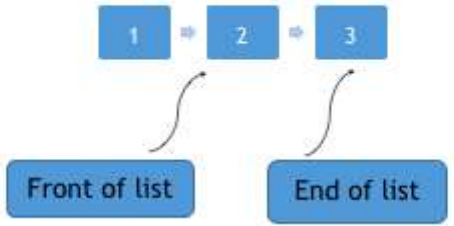
List

```
include <iostream>
#include <list>
using namespace std;
int main()
{
   list<int> l;
/* Creates a new empty linked list l */

 list<int> myList{1,2,3};
   /* creates list with 1,2,3 in it */

   list<int> myNewList = 10;
   /*  create list myNewList of integer
       and copies value of 1 into it*/
}
```

Similar to vector and array, lists can also be initialized with parameters,

```
list<int> l{1,2,3};
```

The above code will create list as :



Functions in List

empty function :This method returns true if the list is empty else returns false.

size function : This method can be used to find the number of elements present in the list.

front and back function : front() is used to get the first element of the list from the start while back() is used to get the first element of the list from the back.

reverse function :This method can be used to reverse a list completely.

push_back and push_front functions :
push_back(element) method is used to push elements into a list from the back.
push_front(element) method is used to push elements into a list from the front.

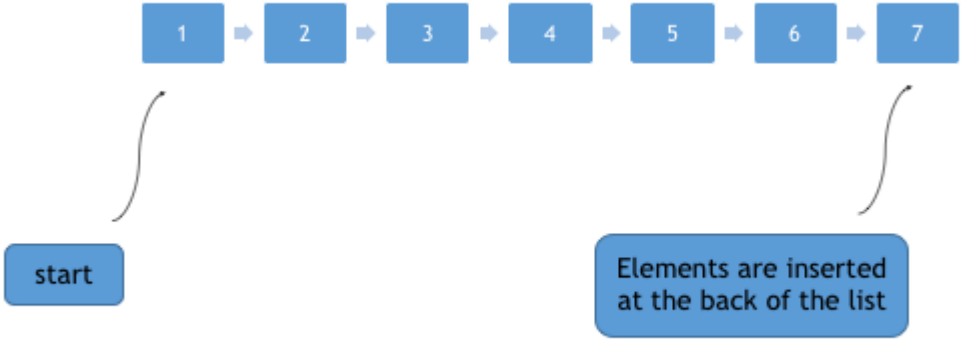
pop_front() , pop_back(): pop_front() removes first element from the start of the list. While pop_back() removes first element from the end of the list.

Practice question –push_back, push_front

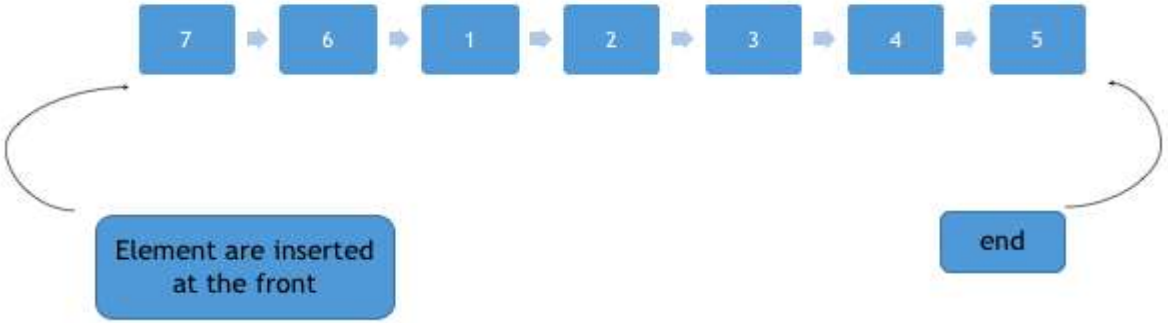
```
#include <iostream>
#include <list>
using namespace std;
int main()
{
   list<int> l{1,2,3,4,5};
   l.push_back(6);
   l.push_back(7);
   /* now the list becomes 1,2,3,4,5,6,7 */
   l.push_front(8);
   l.push_front(9);
   /* now the list becomes 9,8,1,2,3,4,5,6,7 */

}
```

push_back and push_front

push_back(element) method is used to push elements into a list from the back.



push_front(element) method is used to push elements into a list from the front.



Practice question –pop_back , pop_front

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
   list<int> l{1,2,3,4,5};
   l.pop_back()();
   /* now the list becomes 1,2,3,4 */
   l.pop_front()();
   /* now the list becomes 2,3,4 */
}
```

Practice question –size, clear, reverse

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
    list<int> l{1,2,3,4,5};
    cout<<"size of list = "<<l.size()<<endl; //size is 5
    cout<<"Front element in list =  "<<l.front()<<endl; //returns 1
    cout<<"Back element in list =  "<<l.back()<<endl; //returns 5
    l.clear(); //clears the list
    cout<<"size of list = "<<l.size()<<endl; //size becomes 0
    l.reverse(); /* now the list becomes 5,4,3,2,1 */
   list<int>::iterator it;    //you can't print the list  using at(), this function not available with list
   for(it=l.begin(); it!=l.end();it++)
        cout<<*it <<" ";
}
```

Practice question –insert

insert function: This method, as the name suggests, inserts an element at specific position, in a list.

There are 3 variations of insert(), they are as follows :

insert(iterator, element) : inserts element in the list before the position pointed by the iterator.

insert(iterator, count, element) : inserts element in the list before the position pointed by the iterator, count number of times.

insert(iterator, start_iterator, end_iterator): insert the element pointed by start_iterator to the element pointed by end_iterator before the position pointed by iterator

Practice question –insert

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
   list<int> l = {1,2,3,4,5};

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

   for(it=l.begin(); it!=l.end();it++)
      cout<<*it <<" ";
```

```
 it=l.begin() ;

 l.insert (it, 100);

// insert 100 before 1 position

   /* now the list is 100 1 2 3 4 5 */

   cout<<endl<< "The revised list is "<<endl;

   for(it=l.begin(); it!=l.end();it++)
      cout<<*it <<" ";
```

Practice question –insert

```
list<int> new_l = {10,20,30,40};   // new list

cout<<endl<<"The new list is "<<endl;

for(it=new_l.begin(); it!=new_l.end();it++)
    cout<<*it <<" ";

new_l.insert (new_l.begin(), l.begin(), l.end());

/* insert elements from beginning of list l to end of list l  before 1 position in list new_l */
/* now the list new_l is 100 1 2 3 4 5 10 20 30 40 */

cout<<endl<<"The revised list after insert is "<<endl;
for(it=new_l.begin(); it!=new_l.end();it++)
    cout<<*it <<" ";
```

Practice question –insert

```
    l.insert(l.begin(), 5, 10);  // insert 10 before beginning 5 times

/* now l is 10 10 10 10 10 100 1 2 3 4 5 */

    cout<<endl<<"The last operation of insert 10 gives you "<<endl;

    for(it=l.begin(); it!=l.end();it++)
        cout<<*it <<" ";

}
```

Practice question –sort

sort function: sort() method sorts the given list. It does not create new sorted list but changes the position of elements within an existing list to sort it.

This method has two variations :

sort() : sorts the elements of the list in ascending order, the element of the list should by numeric for this function.

sort(compare_function) : This type of sort() is used when we have to alter the method of sorting. Its very helpful for the elements that are not numeric. We can define how we want to sort the list elements in compare_funtion. For example, list of strings can be sorted by the length of the string, it can also be used for sorting in descending order.

Practice question –sort

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
   list<int> list1 = {2,4,5,6,1,3};
   list<int>::iterator it;
   cout<<"Before sorting the list "<<endl;
   for (it=list1.begin();it!=list1.end();it++)
      cout<<*it <<" ";
   list1.sort();
   /* list1 is now 1 2 3 4 5 6 */
   cout<<endl<<"After sorting the list "<<endl;
   for (it=list1.begin();it!=list1.end();it++)
      cout<<*it <<" ";
 }
```

Practice question –sort

```
#include <iostream>
#include <list>
using namespace std;
bool compare_function( string& s1 , string& s2
)
{
   return ( s1.length() > s2.length() );
}
int main()
{
   list<string> list2 = {"h", "hhh", "hh"};

   list<string>::iterator it1;

   cout<<endl<<"Before sorting the list "<<endl;
```

```
 for (it1=list2.begin();it1!=list2.end();it1++)
       cout<<*it1 <<" ";

list2.sort(compare_function);
  /* list2 is now h hh hhh */

cout<<endl<<"After sorting the list "<<endl;

for (it1=list2.begin();it1!=list2.end();it1++)
       cout<<*it1 <<" ";

}
```

Practice question - merge

merge function:
Merges two sorted list. It is mandatory that both the list should be sorted first. merge() merges the two list such that each element is placed at its proper position in the resulting list. Syntax for merge is list1.merge(list2).

The list that is passed as parameter does not get deleted and the list which calls the merge() becomes the merged list

Practice question - merge

merge function:
Merges two sorted list. It is mandatory that both the list should be sorted first.

merge() merges the two list such that each element is placed at its proper position in the resulting list.

Syntax for merge is list1.merge(list2).

The list that is passed as parameter does not get deleted and the list which calls the merge() becomes the merged list

Practice question - merge

```
#include <iostream>
#include <list>
using namespace std;
int main ()
{
   list<int> list1 = {1,3,5,7,9};
   list<int> list2 = {2,4,6,8,10};
/* both the lists are sorted. In case they are not ,
   first they should be sorted by sort function() */
    list1.merge(list2);
 /* list list1 is now 1,2,3,4,5,6,7,8,9,10  */
    cout << list1.size() << endl;    // prints 10
   list<int>::iterator it;
   for(it=list1.begin();it!=list1.end();it++)
      cout<<*it <<" ";
  }
```

MCQ

Which of the following is correct syntax for list creation in C++?

list<int> l

list<int> l{ 1,2 3}

list<int> myNewList = 10;

1. 1 ,2 only
2. 2 ,3 only
3. 1,2 3
4. 1,3

MCQ

Which of the following is correct syntax for list creation in C++?

list<int> l

list<int> l{ 1,2 3}

list<int> myNewList = 10;

1. 1 ,2 only
2. 2 ,3 only
3. 1,2 3
4. 1,3

MCQ

Which of the following function is not available in list container in C++?

1. empty
2. At
3. Front
4. size

MCQ

Which of the following function is not available in list container in C++?

1. empty
2. At
3. Front
4. size

MCQ

Which of the following is not correct in list container in C++?

1. Insert function inserts the element before the given position
2. Reverse function will print the reversed contents
3. Sort does not create new lists but sort within an existing list
4. Merge function does not delete the list passed as parameter but merges it the list which calls merge

MCQ

Which of the following is not correct in list container in C++?

1. Insert function inserts the element before the given position
2. Reverse function will print the reversed contents
3. Sort does not create new lists but sort within an existing list
4. Merge function does not delete the list passed as parameter but merges it the list which calls merge

Assignment

Create a list of 10 integers

Apply following functions on it and observe the output by printing the contents.

1. Empty
2. Size
3. Front
4. Back
5. Push_back
6. Push_front
7. Pop_back
8. Pop_front
9. Insert
10. Reverse
11. clear
12. Sort
13. Merge (create a new list and merge with original)

STL Algorithms

STL provide different types of algorithms that can be implemented upon any of the container with the help of iterators.

Thus now we don’t have to define complex algorithm instead we just use the built in functions provided by the algorithm library in STL.

Algorithm functions provided by algorithm library works on the iterators, not on the containers. Thus one algorithm function can be used on any type of container.

Use of algorithms from STL saves time, effort, code and are very reliable.

There are many algorithm in algorithm library such as sorting algorithm, searching algorithm, numeric algorithm etc. Out of which we will study only sorting algorithm.

Sorting Algorithm

We will be studying about two methods under Sorting Algorithms, namely:

sort

is_sorted

Note: earlier sort() function that we learnt was working with containers. This sorting algorithm will work on int, string ,range of elements. You cannot directly apply sort function like list.sort() here.

Sorting Algorithm

sort method: This function of the STL, sorts the contents of the given range.

There are two version of sort() :
sort(start_iterator, end_iterator) : sorts the range defined by iterators start_iterator and end_iterator in ascending order.

sort(start_iterator, end_iterator, compare_function) : this also sorts the given range but you can define how the sorting should be done by compare_function.

Sorting Algorithm- example 1

```
#include<iostream>
#include<algorithm> //always include while using algorithm
#include<vector>
using namespace std;

int main()
{
   int arr[5] = {1,5,8,4,2};

   sort(arr , arr+5);    // sorts arr[0] to arr[4] in ascending order
   /* now the arr is 1,2,4,5,8  */

    for (int i=0;i<5;i++)
       cout<<arr[i]<<" ";
}
```

Sorting Algorithm- example 2

```
    vector<int> v1;
    v1.push_back(8);
    v1.push_back(4);
    v1.push_back(5);
    v1.push_back(1);
/* now the vector v1 is 8,4,5,1 */
    vector<int>::iterator i, j;
    i = v1.begin();   // i now points to beginning of the vector v1
    j = v1.end();     // j now points to end of the vector v1
    sort(i,j);      //sort(v1.begin() , v1.end() ) can also be used
    /* now the vector v1 is 1,4,5,8 */
    cout<<endl;
    for(i = v1.begin(); i != v1.end(); i++)
   {
        cout << *i <<" ";   // for printing the vector
    }
```

Sorting Algorithm- example 3

```
bool compare_function(int i, int j)
{
   return i > j;    // return 1 if i>j else 0
}
int main() {
    int a2[] = { 4,3,6,5,6,8,4,3,6 };
    sort(a2,a2+9,compare_function);  // sorts a2 in descending order
    /* here we have used compare_function which uses operator(>),
    that result into sorting in descending order */
   /* compare_function is also used to sort non-numeric elements such as*/
    cout<<endl;
    for( int i=0;i<9; i++)
    {
         cout << a2[i] <<" ";   // for printing an vector
    }
 }
```

Sorting Algorithm- example 4

```
bool compare_string(string i, string j)
{
  return (i.size() < j.size());
}
int main() {
    string s[]={"a" , "abc", "ab" , "abcde"};

    sort(s,s+4,compare_string);
    /* now s is "a","ab","abc","abcde"  */
    cout<<endl;
    for( int i=0;i<4; i++)
    {
         cout << s[i] <<" ";   // for printing an array
    }
 }
```

Sorting Algorithm- is_sorted()

is_sorted method : This function of the STL, returns true if the given range is sorted.

There are two version of is_sorted() :

is_sorted(start_iterator, end_iterator) : Checks the range defined by iterators start_iterator and end_iterator in ascending order.

is_sorted(start_iterator, end_iterator, compare_function) : It also checks the given range but you can define how the sorting must be done.

Sorting Algorithm- is_sorted()

is_sorted method : This function of the STL, returns true if the given range is sorted.

There are two version of is_sorted() :

is_sorted(start_iterator, end_iterator) : Checks the range defined by iterators start_iterator and end_iterator in ascending order.

is_sorted(start_iterator, end_iterator, compare_function) : It also checks the given range but you can define how the sorting must be done.

Sorting Algorithm- is_sorted()

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
int main()
{
  int a[5] = {1,5,8,4,2};
  cout<<is_sorted(a, a+5);
  /* prints 0 , Boolean false  */
 vector<int> v1;
 v1.push_back(8);
  v1.push_back(4);
  v1.push_back(5);
  v1.push_back(1);
```

```
/* now the vector v1 is 8,4,5,1 */
cout<<is_sorted(v1.begin() , v1.end() );
 /* prints 0 */
sort(v1.begin() , v1.end() );
/* sorts the vector v1 */
cout<<is_sorted(v1.begin() , v1.end());
 /* prints 1 , as vector v1 is sorted */

}
```

MCQ

which of the following is true in case of Sorting algorithm in STL?

1. Sort(i,j) will sort the elements within range i to j where i and j are called iterators.
2. is_sorted(i,j) will return boolean value if the list within the given range is sorted or not.

A. 1 is true
B. 2 is true
C. Both are true
D. Both are false.

MCQ

which of the following is true in case of Sorting algorithm in STL?

1. Sort(i,j) will sort the elements within range i to j where i and j are called iterators.
2. is_sorted(i,j) will return boolean value if the list within the given range is sorted or not.

A. 1 is true
B. 2 is true
C. Both are true
D. Both are false.

MCQ

which of the following is true in case of Sorting algorithm in STL?

1. Sort function will always sort the elements in ascending order by default
2. Sort function can be modified to get descending order of elements.

A. 1 is true
B. 2 is true
C. Both are true
D. Both are false.

MCQ

which of the following is true in case of Sorting algorithm in STL?

1. Sort function will always sort the elements in ascending order by default
2. Sort function can be modified to get descending order of elements.

A. 1 is true
B. 2 is true
C. Both are true
D. Both are false.

Assignment

Create an vector/array of 10 strings , write your own compare function to sort an array. Use is_sorted function to check the sorted status. Print the sorted array before and after sorting.

QNA Time

Any Questions??

Thank You!

See you guys in next class.