CSE 1203

Object Oriented Programming [C++]

Chapter 4: Advanced Topics-2

STL in C++ (Vector)

Vector is a dynamic array
SYNTAX for creating a vector is: `vector< object_type > vector_name;`

Member Functions

`push_back()` is used for inserting an element at the end of the vector.
`insert(itr, element`) method inserts the element in vector before the position pointed by iterator itr.
`pop_back()` is used to remove the last element from the vector. It reduces the size of the vector by one.
`erase(itr_pos)` removes the element pointed by the iterator **itr_pos**.
`reserve(value)` method allocates capacity
`swap ()` method interchanges value of two vectors.
`clear()` method clears the whole vector, removes all the elements from the vector but do not delete the vector.
`size ()` This method returns the size of the vector.
`empty()` method returns true if the vector is empty else returns false.
`resize(n)` method resizes the vector to **n** elements.
`capacity()` method returns the number of elements that can be inserted in the vector. The sizes becomes double if it is full.
`shrink_to_fit()` – Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.
`at()` method works same in case of vector as it works for array.
`front()` retuns the element at the front of the vector (i.e. leftmost element).
`back()` returns the element at the back of the vector (i.e. rightmost element).

STL in C++ (Vector)

Member Functions (contd)

begin() – Returns an iterator pointing to the first element in the vector
end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector
rbegin() – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
rend() – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)
cbegin() – Returns a constant iterator pointing to the first element in the vector.
cend() – Returns a constant iterator pointing to the theoretical element that follows the last element in the vector.
crbegin() – Returns a constant reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
crend() – Returns a constant reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)

```
vector<int>bx={10,20,30};
vector<int>::const_iterator p=bx.cbegin()+1;
    *p=90;
    cout<<"p="<<*p<<endl;
```

Produces Error as p is constant pointer so *p=90 is illegal

STL in C++ (Iterator)

Iterators are used to point at the memory addresses of STL containers. They are primarily used in sequences of numbers, characters etc. They reduce the complexity and execution time of the program.

Operations of iterators :-

1. begin() :- This function is used to return the **beginning position** of the container.

2. end() :- This function is used to return the ***after* end position** of the container.

3. advance() :- This function is used to **increment the iterator position** till the specified number mentioned in its arguments.

```
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main()
{
    vector<int> ar = { 1, 2, 3, 4, 5 };
    // Declaring iterator to a vector
    vector<int>::iterator ptr;
    // Displaying vector elements using begin()
and end()
    cout << "The vector elements are : ";
    for (ptr = ar.begin(); ptr < ar.end(); ptr++)
        cout << *ptr << " ";
// Using advance() to increment iterator position
    // points to 4
    advance(ptr, 3);

    // Displaying iterator position
    cout << "The position of iterator after
advancing is : ";
    cout << *ptr << " ";

    return 0;
}
```

Output

```
The vector elements are : 1 2 3 4 5
 The position of iterator after advancing is : 4
```

STL in C++ (Iterator)

4. next() :- This function **returns the new iterator** that the iterator would point after **advancing the positions** mentioned in its arguments.
5. prev() :- This function **returns the new iterator** that the iterator would point **after decrementing the positions** mentioned in its arguments.

```
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main()
{
    vector<int> ar = { 1, 2, 3, 4, 5 };
    // Declaring iterators to a vector
    vector<int>::iterator ptr = ar.begin();
    vector<int>::iterator ftr = ar.end();

    // Using next() to return new iterator
    // points to 4
    auto it = next(ptr, 3);
    // Using prev() to return new iterator
    // points to 3
    auto it1 = prev(ftr, 3);
    // Displaying iterator position
    cout << "The position of new iterator using next() is
: ";
    cout << *it << " ";
    cout << endl;
    // Displaying iterator position
    cout << "The position of new iterator using
prev()  is : ";
    cout << *it1 << " ";
    cout << endl;
    return 0;
}
```

Output

```
The position of new iterator using next() is : 4
The position of new iterator using prev() is : 3
```

STL in C++ (Iterator)

6. inserter() :- This function is used to **insert the elements at any position** in the container. It accepts **2 arguments, the container and iterator to position where the elements have to be inserted**.

```
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main()
{
    vector<int> ar = { 1, 2, 3, 4, 5 };
    vector<int> ar1 = {10, 20, 30};

    // Declaring iterator to a vector
    vector<int>::iterator ptr = ar.begin();

    // Using advance to set position
    advance(ptr, 3);

    // copying 1 vector elements in other using
inserter()
    // inserts ar1 after 3rd position in ar
    copy(ar1.begin(), ar1.end(), inserter(ar,ptr));

    // Displaying new vector elements
    cout << "The new vector after inserting elements is :
";
    for (int &x : ar)
        cout << x << " ";

    return 0;
}
```

Output

```
The new vector after inserting elements is : 1 2 3 10 20 30 4 5
```

STL in C++ (vector Exmple)

```
#include<bits/stdc++.h>
#include<array>
using namespace std ;
void Display(vector<int> &p){
 for(int i=0;i<p.size();i++)
    cout<<p[i]<<" ";
 cout<<endl;
}
int main()
{
    vector<int>v{10,20,30,40,50};
    vector<int>::iterator it;
    vector<int>k;
    cout<<"The vector: ";
    for(auto x:v)
        cout<<x<<" ";
    cout<<endl;
    v.push_back(60);
    k.pop_back();
    it=v.begin();
    v.insert(it,566);
    v.insert(it+2,5,15);
    cout<<endl;
```

```
  cout<<"The vector: ";
      for(auto x:v)
          cout<<x<<" ";

      k.reserve(31); //without researve
capacity is always doubled
      for(int i=1;i<=32;i++){
           k.push_back(i);
           cout<<" size="<<k.size()<<"
capacity="<<k.capacity()<<endl;
      }

      cout<<"k vector:";
      Display(k);
  }
```

STL in C++ (stack class)

empty() – Returns whether the stack is empty
size() – Returns the size of the stack
top() – Returns a reference to the top most element of the stack
push(g) – Adds the element ‘g’ at the top of the stack
pop() – Deletes the top most element of the stack

```
#include <iostream>
#include<bits/stdc++.h>
#include<stack>
#include <algorithm>
//#include<utility>
using namespace std;

int main()
{
  stack<int>st;
  st.push(10);
  st.push(20);
  st.push(30);
  st.push(40);
  st.pop();
  cout<<"front="<<st.top()<<endl;
  cout<<"size="<<st.size()<<endl;
}
```

STL in C++ (queue class)

empty() Returns whether the queue is empty. It return true if the queue is empty otherwise returns false.
size() Returns the size of the queue.
swap() Exchange the contents of two queues but the queues must be of the same data type, although sizes may differ.
front() Returns a reference to the first element of the queue.
back() Returns a reference to the last element of the queue.
push(g) Adds the element ‘g’ at the end of the queue.
pop() Deletes the first element of the queue.

```
#include <iostream>
#include<bits/stdc++.h>
#include<queue>
#include <algorithm>
using namespace std;

int main()
{
  queue<int>qu;
  qu.push(10);
  qu.push(20);
  qu.push(30);
  qu.push(40);
  qu.pop();
  cout<<"front="<<qu.front()<<endl;
  cout<<"back="<<qu.back()<<endl;
  cout<<"size="<<qu.size()<<endl;
}
```

STL in C++ (deque class)

insert()	Inserts an element. And returns an iterator that points to the first of the newly inserted elements.
rbegin()	Returns a reverse iterator which points to the last element of the deque (i.e., its reverse beginning).
rend()	Returns a reverse iterator which points to the position before the beginning of the deque (which is considered its reverse end).
cbegin()	Returns a constant iterator pointing to the first element of the container, that is, the iterator cannot be used to modify, only traverse the deque.
max_size()	Returns the maximum number of elements that a deque container can hold.
push_front()	It is used to push elements into a deque from the front.
push_back()	This function is used to push elements into a deque from the back.
pop_front() and pop_back()	pop_front() function is used to pop or remove elements from a deque from the front. pop_back() function is used to pop or remove elements from a deque from the back.
front() and back()	front() function is used to reference the first element of the deque container. back() function is used to reference the last element of the deque container.
clear() and erase()	clear() function is used to remove all the elements of the deque container, thus making its size 0. erase() function is used to remove elements from a container from the specified position or range.
empty() and size()	empty() function is used to check if the deque container is empty or not. size() function is used to return the size of the deque container or the number of elements in the deque container.

```
#include <iostream>
#include<bits/stdc++.h>
#include<queue>
#include <algorithm>
//#include<utility>
using namespace std;

int main()
{
  deque<int>qu;
  deque<int>::iterator it;
  qu.push_back(10);
  qu.push_back(20);
  qu.push_back(30);
  qu.push_back(40);
  //qu.pop_back();
  cout<<"front="<<qu.front()<<endl;
  cout<<"back="<<qu.back()<<endl;
  for(it = qu.begin();it!=qu.end();it++)
    {
        cout << *it << " ";
    }
}
```

STL in C++ (list class)

front()	Returns the value of the first element in the list.
back()	Returns the value of the last element in the list.
push_front(g)	Adds a new element ‘g’ at the beginning of the list.
push_back(g)	Adds a new element ‘g’ at the end of the list.
pop_front()	Removes the first element of the list, and reduces size of the list by 1.
pop_back()	Removes the last element of the list, and reduces size of the list by 1.
begin()	begin() function returns an iterator pointing to the first element of the list.
end()	end() function returns an iterator pointing to the theoretical last element which follows the last element.
rbegin() and rend()	rbegin() returns a reverse iterator which points to the last element of the list. rend() returns a reverse iterator which points to the position before the beginning of the list.
cbegin() and cend()	cbegin() returns a constant random access iterator which points to the beginning of the list. cend() returns a constant random access iterator which points to the end of the list.
crbegin() and crend()	crbegin() returns a constant reverse iterator which points to the last element of the list i.e reversed beginning of container. crend() returns a constant reverse iterator which points to the theoretical element preceding the first element in the list i.e. the reverse end of the list.
empty()	Returns whether the list is empty(1) or not(0).
insert()	Inserts new elements in the list before the element at a specified position.

erase()	Removes a single element or a range of elements from the list.
assign()	Assigns new elements to list by replacing current elements and resizes the list.
remove()	Removes all the elements from the list, which are equal to given element.
remove_if()	Used to remove all the values from the list that correspond true to the predicate or condition given as parameter to the function.
reverse()	Reverses the list.
size()	Returns the number of elements in the list.
resize()	Used to resize a list container.
sort()	Sorts the list in increasing order.
max_size()	Returns the maximum number of elements a list container can hold.
unique()	Removes all duplicate consecutive elements from the list.
emplace_front() and ::emplace_back()	emplace_front() function is used to insert a new element into the list container, the new element is added to the beginning of the list. emplace_back() function is used to insert a new element into the list container, the new element is added to the end of the list.
list::clear()	clear() function is used to remove all the elements of the list container, thus making it size 0.
list::operator=	This operator is used to assign new contents to the container by replacing the existing contents.
list::swap()	This function is used to swap the contents of one list with another list of same type and size.
list splice()	Used to transfer elements from one list to another.

STL in C++ (deque class)

```
#include <iostream>
#include<bits/stdc++.h>
#include<stack>
#include <algorithm>
#include <iterator>
//#include<utility>
using namespace std;

// a predicate implemented as a function:
bool isOdd(int value)
  { return (value%2)==1; }

//Display list in forward direction
void Display(list<int> g){
  list<int>::iterator it;
  for(it=g.begin();it!=g.end();it++){
    cout<<" "<<*it;
  }
 cout<<endl;
}
//Display list in reverse direction
void DisplayR(list<int> g){
  list<int>::reverse_iterator it;
  for(it=g.rbegin();it!=g.rend();it++){
    cout<<" "<<*it;
  }
 cout<<endl;
}
```

```
int main()
{
  list<int>ar;
  list<int>::iterator p,q;
  ar.push_back(30);
  ar.push_back(21);
  ar.push_back(21);
  ar.push_front(21);
  DisplayR(ar);
  //ar.pop_front();
  Display(ar);
  //ar.pop_back();
  DisplayR(ar);
  cout<<"front="<<ar.front()<<endl;
  cout<<"back="<<ar.back()<<endl;

  //insert before first element
  p=ar.begin();
  ar.insert(p,11);
  Display(ar);

 //insert after last element
  p=ar.end();
  ar.insert(p,27);
  Display(ar);
```

STL in C++ (list class)

```
//insert before a node 30
  p=find(ar.begin(),ar.end(),30);
  ar.insert(p,22);
  if(p==ar.end())
    cout<<"Not found"<<endl;
  else
    cout<<"Found address="<<&p<<endl;
  Display(ar);

  //insert after a node 30
  p=find(ar.begin(),ar.end(),30);
  advance(p,1);
  ar.insert(p,26);
  if(p==ar.end())
    cout<<"Not found"<<endl;
  else
    cout<<"Found address="<<&p<<endl;
  Display(ar);

  //delete 2nd element
  p=ar.begin();
  ar.erase(++p);
  Display(ar);

  //delete a specific element 30
  p=find(ar.begin(),ar.end(),30);
  ar.erase(p);
  Display(ar);
```

```
//delete elements within the range
  //here delete first four elements
  p=ar.begin();
  q=ar.begin();
  advance(q,1); //error write q=q+4
  ar.erase(p,q);
  Display(ar);

  //remove elements of value=26
  ar.remove(28);
  Display(ar);

  //remove elements with condition
  cout<<"Remove if"<<endl;
  //ar.remove_if(isOdd);
  Display(ar);

  //count an element
  int mc;
  Display(ar);
  mc = count (ar.begin(), ar.end(), 22);
  cout<<"Count="<<mc<<endl;

//count elements with condition
  Display(ar);
  mc=count_if(ar.begin(), ar.end(),isOdd);
  cout<<"Count odd="<<mc<<endl;
```

STL in C++ (list class)

```
//assign elements
  cout<<"Assign"<<endl;
  list<int> br;
  br.assign (7,100); // 7 ints with value
100
  Display(br);
  br.assign (ar.begin(),ar.end()); // a
copy of first
   Display(br);
  int myints[]={1776,7};
  br.assign (myints,myints+2);
  Display(br);            // assigning
from array

  //Delete consecutive unique values
  cout<<"Unique value deletion"<<endl;
  ar.sort();
  ar.unique();
  Display(ar);
}
```

STL Library: List

THANK YOU