Preview

- List STL Container
- List Container Functions
  - splice()
  - remove()
  - remove_if()
  - unique()
  - sort()
  - merge()
- List STL with Structured Data Type

1

List Container

- Lists are a kind of sequence containers.
- List containers are implemented as doubly-linked lists; Doubly linked lists can store each of the elements they contain in different and unrelated storage locations.
- lists containers perform generally better in inserting, extracting and moving elements in any position within the container

2

List Container

- The main drawback of lists compared to these other sequence containers is that they lack direct access to the elements by their position.
- Why?
  - Each element of list are stored in unrelated memory locations
- Need extra space to keep the linking information associated to each element

3

List Member Functions

(constructor)	Construct list (public member function)
(destructor)	List destructor (public member function)
operator=	Copy container content (public member function)
Iterators:	
begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
Capacity:	
empty	Test whether container is empty (public member function)
size	Return size (public member function)
max_size	Return maximum size (public member function)
resize	Change size (public member function)
Element access:	
front	Access first element (public member function)
back	Access last element (public member function)

4

List Member Functions

Element access:	
front	Access first element (public member function)
back	Access last element (public member function)
Modifiers:	
assign	Assign new content to container (public member function)
push_front	Insert element at beginning (public member function)
pop_front	Delete first element (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
Operations:	
splice	Move elements from list to list (public member function)
remove	Remove elements with specific value (public member function)
remove_if	Remove elements fulfilling condition (public member function template)
unique	Remove duplicate values (member function)
merge	Merge sorted lists (public member function)
sort	Sort elements in container (public member function)
reverse	Reverse the order of elements (public member function)
Allocator:	
get_allocator	Get allocator (public member function)

5

List Member Functions

splice(towhere, from)
iterator list

splice (towhere, from, it)
iterator list iterator

splice(towhere, from, it, it2)
iterator list iterator iterator

6

```
// list1.cpp
// splicing listsMoves elements from list x into the list container at the specified
// position, effectively inserting the specified elements into the container and removing
// them from x.
#include <iostream>
#include <list>
using namespace std;

int main ()
{
  list<int> mylist1, mylist2;
  list<int>::iterator it, it1;

  // set some initial values: 1, 2, 3, 4
  for (int i=1; i<=4; i++)
     mylist1.push_back(i);
  // set some initial values: 10, 20, 30
  for (int i=1; i<=3; i++)
     mylist2.push_back(i*10);

  it = mylist1.begin();
  it ++; // points to 2

  // insert content of mylist2 before pointed by it.
  // mylist1 becomes 1, 10, 20, 30, 2, 3, 4
  // mylist2 become empty. it still points to the 5th element (2)
  mylist1.splice (it, mylist2);
  cout << "mylist1 contains:";
  for (it1= mylist1.begin(); it1 != mylist1.end(); it1++)
             cout<< *it1 <<" ";
  cout<<endl;
  cout << "mylist2 contains:";
  for (it1= mylist2.begin(); it1 != mylist2.end(); it1++)
             cout<< *it1 <<" ";
  cout<<endl;

  return 0;
}
```

splice(towhere, from)
iterrator list

7

```
// list2.cpp
// splicing listsMoves elements from list x into the list container at the specified position,
// effectively inserting the specified elements into the container and removing them from x.
#include <iostream>
#include <list>
using namespace std;

int main ()
{
  list<int> mylist1, mylist2;
  list<int>::iterator it, it1;

  // set some initial values: 1, 2, 3, 4, 5, 6
  for (int i=1; i<=6; i++)
     mylist1.push_back(i);

   it = mylist1.begin();
  advance(it,3); // it points now to 4
  // one element pointed by it in mylist1 insert as a first element of list2
  // after splice, mylist1 becomes 1, 2, 3, 5, 6
  // mylist2 becomes 4, iterator it become invalid since element pointed by it is removed
  mylist2.splice (mylist2.begin(),mylist1, it);
  cout << "mylist1 contains: ";
  for (it1= mylist1.begin(); it1 != mylist1.end(); it1++)
             cout<< *it1 <<" ";
          cout<<endl;
  cout << "mylist2 contains: "; // mylist2 becomes 4.
  for (it1= mylist2.begin(); it1 != mylist2.end(); it1++)
             cout<< *it1 <<" ";
  cout<<endl;
  return 0;
}
```

splice (towhere, from, it)
iterator list iterator

8

```
// list3.cpp
// splicing listsMoves elements from list x into the list container at the specified position,
// effectively inserting the specified elements into the container and removing them from x.
#include <iostream>
#include <list>
using namespace std;

int main ()
{
  list<int> mylist1;
  list<int>::iterator it, it1;

  // set some initial values: 1, 2, 3, 4, 5, 6
  for (int i=1; i<=6; i++)
     mylist1.push_back(i);

  it = mylist1.begin();
  advance(it,3);                  // "it" points now to 4

  // mylist1 become 4, 5, 6, 1, 2, 3
  mylist1.splice ( mylist1.begin(), mylist1, it, mylist1.end());

  cout << "mylist1 contains:";
  for (it=mylist1.begin(); it!=mylist1.end(); it++)
     cout << " " << *it;
  return 0;
}
```

splice(towhere, from, it, it2)
iterator list iterator iterator

9

List Member Functions

```
// list4.cpp
// list member function remove():
// Removes from the list all the elements with a specific value
#include <iostream>
#include <list>
using namespace std;

int main ()
{
  int myints[]= {17,89,7,14,89};
  int rm;
  list<int> mylist (myints,myints+5);
  list<int>::iterator it;
  cout <<"initial mylist contains: ";
  for (it=mylist.begin(); it!=mylist.end(); ++it)
     cout << " " << *it;
  cout << endl;

  cout <<"element you want delete from mylist: ";
  cin >> rm;

  mylist.remove(rm);
  cout << "mylist contains after remove(" << rm << "):";

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

  return 0;
}
```


10

List Member Functions

```
// list5.cpp
// list member function remove()& sort():
// Removes from the list all the elements with a specific value
#include <iostream>
#include <list>
#include <string>
using namespace std;
int main ()
{
  string mlist[]= {"Michael","July","Scott","Bill","Sam"};
  list<string> mylist (mlist,mlist+5); // initiate list
  list<string>::iterator it; //iterator for a list
  cout<<" mylist initial elements are: ";
  for (it=mylist.begin(); it!=mylist.end(); ++it)
     cout << " " << *it;
  cout << endl;
  char tmp[10];
  cout << "a element wish to remove from the list: ";
  cin >>tmp;
  mylist.remove(tmp);
  cout << "mylist contains after remove "<<tmp<<": ";
  for (it=mylist.begin(); it!=mylist.end(); ++it)
     cout << " " << *it;
  cout << endl;
  mylist.sort();
  cout << "mylist contains after sorted "<<tmp<<": ";
  for (it=mylist.begin(); it!=mylist.end(); ++it)
     cout << " " << *it;
  cout << endl;
  return 0;
}
```


11

```
// list6.cpp
// list::remove_if(): Removes from the list all the elements
// for which Predicate pred returns true.
#include <iostream>
#include <list>
using namespace std;
template < class T >
void printList(list< T >&);
// a predicate chek single digit
bool single_digit (const int& value) { return (value<10); }
// a predicate check odd number
bool odd(const int &value) {return (value%2);}

int main ()
{
  int myints[]= {15,36,7,17,20,39,4,1};
  list<int> mylist (myints,myints+8);   // 15 36 7 17 20 39 4 1
  cout <<"initial mylists elements are"<<endl;
  printList(mylist);

  cout <<"after remove single digit numbers, mylists elements are: "<<endl;
  mylist.remove_if (single_digit);       // 15 36 17 20 39
  printList(mylist);
  cout <<"after remove odd numbers, mylists elements are: "<<endl;
  mylist.remove_if (odd);           // 36 20
  printList(mylist);
  return 0;
}
template < class T >
void printList(list< T >& L)
{
          list<int>::iterator it;
          for (it=L.begin(); it!=L.end(); ++it)
                    cout << " " << *it;
          cout << endl<<endl;
}
```


12

List STL with Structured Data Type

```
// list8.cpp
// a list STL with structured data type
#include <iostream>
#include <list>
using namespace std;
struct Student {
          char LastName[20];                //last name
          char FirstName[20];               // First Name
          int IDNumber;                     // Student ID
          Student();                        // Constructor
          bool operator == (const Student);
          bool operator > (const Student);
          bool operator < (const Student);
          friend ostream& operator << (ostream &stream, const Student &student);
};
// Student Constructor
Student::Student()
{
          cout << "What is the student's Last Name: ";
          cin >> LastName;
          cout << "What is the student's First Name: ";
          cin >> FirstName;
          cout << "What is the student's ID#: ";
          cin >> IDNumber;
}
```


13

```
// == Operator Overloader: checks if the numbers are equal
bool Student::operator == (Student x)
{
          return (x.IDNumber == IDNumber);
}
// > Operator Overloader: checks if the number is greater than
bool Student::operator > (Student x)
{
          return (IDNumber > x.IDNumber);
}
// < Operator Overloader: checks if the number is less than
bool Student::operator < (Student x)
{
          return (IDNumber < x.IDNumber);
}
// << Operator Overloader: displays the structures information
ostream& operator << (ostream &stream, const Student &student)
{
          stream << "ID#: " << student.IDNumber <<" - " << student.LastName << ", " <<
student.FirstName << endl;
          return(stream);
}
int main ()
{
  // vector for Student structure
  list <Student> StdList;
  list<Student>::iterator it;
  for (int i= 1; i <3; i++)
  {
            Student *ptr;
            ptr = new Student;
            StdList.push_back(*ptr);
  }
  for (it=StdList.begin(); it != StdList.end(); it++)
        cout << *it <<endl;
  return 0;
}
```


14

List STL with Structured Data Type

```
// list9.cpp
// a list STL with structured data type
#include <iostream>
#include <list>
using namespace std;
struct Student {
          char LastName[20];      //last name
          char FirstName[20];     // First Name
          int IDNumber;           // Student ID
          Student();              // Constructor
          Student(int);           // it is Null constructor for  declare
          bool operator == (const Student);
          bool operator > (const Student);
          bool operator < (const Student);
          friend ostream& operator << (ostream &stream, const Student &student);
};
// Student Constructor
Student::Student()
{
          cout << "What is the student's Last Name: ";
          cin >> LastName;
          cout << "What is the student's First Name: ";
          cin >> FirstName;
          cout << "What is the student's ID#: ";
          cin >> IDNumber;
}
Student::Student(int x)
{;}
```


15

```
// == Operator Overloader: checks if the numbers are equal
bool Student::operator == (Student x)
{
          return (x.IDNumber == IDNumber);
}
// > Operator Overloader: checks if the number is greater than
bool Student::operator > (Student x)
{
          return (IDNumber > x.IDNumber);
}
// < Operator Overloader: checks if the number is less than
bool Student::operator < (Student x)
{
          return (IDNumber < x.IDNumber);
}
// << Operator Overloader: displays the structures information
ostream& operator << (ostream &stream, const Student &student)
{
          stream << "ID#: " << student.IDNumber <<" - " << student.LastName << ", " << student.FirstName
<< endl;
          return(stream);
}
```

16

```
int main ()
{
  // vector for Student structure
  list <Student> StdList;
  list<Student>::iterator it;
  for (int i= 1; i <4; i++)
  {
            Student *ptr;
            ptr = new Student;
            StdList.push_back(*ptr);
  }
  StdList.sort();
  for (it=StdList.begin(); it != StdList.end(); it++)
            cout << *it <<endl;

  Student tmp(0); // declare
  int ID;
  cout << "Student ID for remove: "<<endl;
  cin >> tmp.IDNumber;
  StdList.remove(tmp); // remove operator in the list use overloaded operator == from Student
  for (it=StdList.begin(); it != StdList.end(); it++)
            cout << *it <<endl;
  return 0;
}
```

17