Preview

- More <vector> member functions
  - front(), back()
  - at()
  - insert();
- <list> Container
- What is list container
- <list> Container functions


```
// vector.cpp Testing Standard Library vector class template
#include <iostream>
#include <vector>
using namespace std;

int main()
{
   const int SIZE = 6;
   vector< int > v, w;

   cout << "The initial size of v is: " << v.size() <<endl;
   cout << "The initial capacity of v is: " << v.capacity()<<endl;

   for (int i = 1; i <= SIZE; i++)
   {
            v.push_back(i);
            cout <<"Now, the capacity of v becomes " << v.capacity() <<endl;
            cout <<"Now, the size of v becomes " << v.size()<<endl <<endl;
   }
          // using iterator for display element of vector
   cout << "Contents of vector v using iterator notation: ";
   vector <int>::iterator p3;
          for (p3=v.begin(); p3 != v.end(); p3++)
             cout << *p3 << ' ';
          cout <<endl <<endl;
           // using reversed iterator for display element of vector
   cout << "Reversed contents of vector v using reverse_iterator notation: ";
   vector< int >::reverse_iterator p2;
   for ( p2 = v.rbegin(); p2 != v.rend(); ++p2 )
      cout << *p2 << ' ';
   cout <<endl;

   w = v; // use overloaded operator in vector STL;
   cout << "Contents of vector w using iterator notation: ";
   for (p3=w.begin(); p3 != w.end(); p3++)
            cout << *p3 << ' ';

   cout << endl;
   return 0;
}
```



```
// vector1.cpp Testing Standard Library vector class template
#include <iostream>
#include <vector>
using namespace std;

template < class T >
void printVec(vector< T > &vec );

template < class T >
void printVecRev(vector< T > &vec );

int main()
{
   const int SIZE = 6;
   vector< int > v, w;

   cout << "The initial size of v is: " << v.size() <<endl;
   cout << "The initial capacity of v is: " << v.capacity()<<endl;

   for (int i = 1; i <= SIZE; i++)
   {
            v.push_back(i);
            cout <<"Now, the capacity of v becomes " << v.capacity() <<endl;
            cout <<"Now, the size of v becomes " << v.size()<<endl <<endl;
   }
          // using iterator for display element of vector
   cout << "Contents of vector v using iterator notation: ";
   printVec(v);
          // using reversed iterator for display element of vector
   cout << "Reversed contents of vector v using reverse_iterator notation: ";
   printVecRev(v);

   w = v; // use overloaded operator in vector STL;
   cout << "Contents of vector w using iterator notation: ";
   printVec(w);
   return 0;
}
```


Vector Container Member Function

```
template < class T >
void printVec(vector< T > &vec )
{
   for ( i = 0; p != vec.end(); p++ )
      cout << *p << ' ';
   cout <<endl;
}

template < class T >
void printVecRev(vector< T > &v )
{
  vector< T >::reverse_iterator p2;

   for ( p2 = v.rbegin(); p2 != v.rend(); ++p2 )
      cout << *p2 << ' ';
   cout << endl;
}
```


Vector Container Member Function

```
// vector2.cppcomparing size, capacity and max_size
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
  vector<int> myvector;

  // set some content in the vector:
  for (int i=0; i<100; i++) myvector.push_back(i);

  cout << "size: " << (int) myvector.size() << endl;
  cout << "capacity: " << (int) myvector.capacity() << endl;
  cout << "max_size: " << (int) myvector.max_size() << endl;
  return 0;
}
```



```
// vector3.cpp vector container member functions resize(), push_back()
#include <iostream>
#include <vector>
using namespace std;

template <class T>
void printVec(vector <T> &);

int main ()
{
  vector<int> myvector;

  // set some initial content:
  for (int i=1;i<10;i++)
            myvector.push_back(i);

  printVec(myvector);
  myvector.resize(5);
  printVec(myvector);

  myvector.resize(8,100);
  printVec(myvector);

  myvector.resize(12);
  printVec(myvector);

  return 0;
}
// function for print vector
template <class T>
void printVec(vector <T> &v)
{
          for (int i=0; i<v.size(); i++)
            cout << " " <<v[i];
          cout <<endl;
}
```

Vector Container Member Function

```
// vector4.cpp front(): access front
//              back(): access back
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
  vector<int> myvector;

  myvector.push_back(77);
  myvector.push_back(16);

  // now front equals 77, and back 16

  myvector.front() -= myvector.back();

  cout << "myvector.front() is now " << myvector.front() << endl;

  return 0;
}
```



```
// vector5.cpp insert() inserting into a vector
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
  vector<int> myvector (3,100);
  vector<int>::iterator it;

  it = myvector.begin();
  it = myvector.insert ( it , 200 );

  myvector.insert (it,2,300);

  // "it" no longer valid, get a new one:
  it = myvector.begin();

  vector<int> anothervector (2,400);
  myvector.insert (it+2,anothervector.begin(),anothervector.end());

  int myarray [] = { 501,502,503 };
  myvector.insert (myvector.begin(), myarray, myarray+3);

  cout << "myvector contains:";
  for (it=myvector.begin(); it<myvector.end(); it++)
    cout << " " << *it;
  cout << endl;

  return 0;
}
```


Vector Member Function

(front(), back())

```
// vector4.cpp front(): access front
//              back(): access back
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
  vector<int> myvector;

  myvector.push_back(77);
  myvector.push_back(16);

  // now front equals 77, and back 16

  myvector.front() -= myvector.back();

  cout << "myvector.front() is now " << myvector.front() << endl;

  return 0;
}
```



```
//  vector5.cpp insert() inserting into a vector
#include <iostream>
#include <vector>
using namespace std;
template < class T >
void printVec(vector< T >&);

int main ()
{
  vector<int> myvector (3,100);
  vector<int>::iterator it;

  it = myvector.begin();
  it = myvector.insert ( it , 200 );
  cout<< "content of myvector: ";
  printVec(myvector);

  myvector.insert (it,2,300);
  cout<< "content of myvector: ";
  printVec(myvector);
  // "it" no longer valid, get a new one:
  it = myvector.begin();

  vector<int> anothervector (2,400);
  myvector.insert (it+2,anothervector.begin(),anothervector.end());
  cout<< "content of myvector: ";
  printVec(myvector);

  int myarray [] = { 501,502,503 };
  myvector.insert (myvector.begin(), myarray, myarray+3);
  cout<< "content of myvector: ";
  printVec(myvector);


  return 0;
}
template < class T >
void printVec(vector< T > &vec )
{
   for (int i=0; i <vec.size(); i++ )
      cout << vec.at(i) << ' ';
   cout <<endl;
}
```


Vector for Structured Data Type

```
// vector6.cpp vector container member functions with structured data type
#include <iostream>
#include <vector>
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;
}
```



```
// == 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
          vector<Student> StdVec;

  for (int i= 1; i <3; i++)
  {
          Student *ptr;
          ptr = new Student;
          StdVec.push_back(*ptr);
  }

  for (int i =1; i < 3; i++)
          cout << StdVec[i-1];


}
```

Vector Member Function

(at( ))

```
// vector7.cpp
// vector member function at()
// Returns a reference to the element at position n in the vector
// can be used for modify or get a value of element with index i
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
  vector<int> myvector (10);   // 10 zero-initialized ints
  unsigned int i;

  // assign some values:
  for (i=0; i<myvector.size(); i++)
    myvector.at(i)=i;

  cout << "myvector contains:";
  for (i=0; i<myvector.size(); i++)
    cout << " " << myvector.at(i); // at(i) is a content of vector in index i

  cout << endl;

  return 0;
}
```



```
// vector8.cpp
// vector member function assign(): Assigns new content to the vector object,
// dropping all the elements contained in the vector before the call and
// replacing them by those specified by the parameters
#include <iostream>
#include <vector>
using namespace std;
template < class T >
void printVec(vector< T >&);
int main ()
{
  vector<int> first;
  vector<int> second;
  vector<int> third;
  // a repetition 7 times of value 100
  first.assign (7,100); // first:100 100 100 100 100 100 100
  printVec(first);
  vector<int>::iterator it;
  it=first.begin()+1; // it point to second value of vector first

  // assign the 5 central values of  vector first to vector second: 100 100 100 100 100
  second.assign (it,first.end()-1);
  printVec(second);

  int myints[] = {1,2,3,4,5,6,7};
  // assigning from array 2, 3, 4, 5  to vector third
  third.assign (myints+1,myints+5);
  printVec(third);
  return 0;
}

template < class T >
void printVec(vector< T > &vec )
{
   for (int i=0; i <vec.size(); i++ )
      cout << vec.at(i) << ' ';
   cout <<endl;
}
```



```
// vector9.cpp
// vector member function erase(): Removes from the vector container either
// a single element (position) or a range of elements
#include <iostream>
#include <vector>
using namespace std;
template < class T >
void printVec(vector< T >&);
int main ()
{
  vector<unsigned int> myvector;

  // push values 1 2 3 4 5 6 7 8 9 10 to myvector
  for (int i=1; i<=10; i++)
            myvector.push_back(i);

  // erase the 6th element myvector become 1 2 3 4 5 7 8 9 10
  myvector.erase (myvector.begin()+5);
  printVec(myvector);

  // erase the first 3 elements: myvector become 4 5 7 8 9 10
  myvector.erase (myvector.begin(),myvector.begin()+3);
  printVec(myvector);
  return 0;
}

template < class T >
void printVec(vector< T > &vec )
{
   for (int i=0; i <vec.size(); i++ )
      cout << vec.at(i) << ' ';
   cout <<endl;
}
```



```
// vector10.cpp
// vector member function switch(): Exchanges the content of the vector by
// the content of vec, which is another vector of the same type. Sizes may differ.
#include <iostream>
#include <vector>
using namespace std;
template < class T >
void printVec(vector< T >&);
int main ()
{
  vector<int> first (3,1);   // first 1 1 1
  vector<int> second (5,2);  // second 2 2 2 2 2

  cout <<"content of the first vector: ";
  printVec(first);
  cout <<"content of the second vector: ";
  printVec(second);
  cout <<endl;

  //swap content of first and second vector
  first.swap(second);
  cout <<"After swap "<<endl;
  cout <<"content of the first vector: ";
  printVec(first); // first 2 2 2 2 2
  cout <<"content of the second vector: ";
  printVec(second); // second 1 1 1
  return 0;
}
template < class T >
void printVec(vector< T > &vec )
{
   for (int i=0; i <vec.size(); i++ )
      cout << vec.at(i) << ' ';
   cout <<endl;
}
```



```
// vector11.cpp
// vector member function clear(): All the elements of the vector are dropped,
// leaving the container with a size of 0.
#include <iostream>
#include <vector>
using namespace std;
template < class T >
void printVec(vector< T >&);

int main ()
{
  vector<int> myvector;
  for (int i =0; i <5; i++) //myvector initiate with 0 1 2 3 4
            myvector.push_back(i);

  cout << "myvector contains:";
  printVec(myvector);

  //clear content of myvector
  myvector.clear();
  cout<<"content of myvector is cleared"<<endl;
  cout<<"Now content of myvector are:"<<endl;
  printVec(myvector);
  for (int i =0; i <6; i++) //myvector assign with new values 0 10 20 30 40 50
            myvector.push_back(i*10);
  cout << "myvector contains:";
  printVec(myvector);
  cout << endl;
  return 0;
}

template < class T >
void printVec(vector< T > &vec )
{
   for (int i=0; i <vec.size(); i++ )
      cout << vec.at(i) << ' ';
   cout <<endl;
}
```

```
// vector12.cpp :vector member function get_allocater:
// Returns the allocator object used to constuct the vector
#include <iostream>
#include <vector>
using namespace std;
int main ()
{
  vector<int> myvector;
  vector<char> chvector;
  int * p;
  char *cp;
  unsigned int i;
  // allocate an array of integer 5 elements using vector's allocator:
  p=myvector.get_allocator().allocate(5);
  cout <<"contant of a integer array after allocated: ";
  for (i=0; i<5; i++)
     cout << " " << p[i];
  cout << endl;
  // assign some values to array
  for (i=0; i<5; i++)
            p[i]=i;
  cout << "The allocated integer array contains:";
  for (i=0; i<5; i++) //print after allocated
     cout << " " << p[i];
  cout << endl;
 // allocate an array of integer 5 elements using vector's allocator:
  cp =chvector.get_allocator().allocate(4);
  cout <<"contant of a charector array after allocated: ";
  for (i=0; i<5; i++) //print after allocated
     cout << " " << cp[i];
  cout << endl;

  // assign some values to array
  for (i=0; i<4; i++)
            cp[i]='a';
  cout << "The allocated array contains:";
  for (i=0; i<4; i++) cout << " " << cp[i];
  cout << endl;
  // need deallocate array since it is created with pointer
  myvector.get_allocator().deallocate(p,5);
  chvector.get_allocator().deallocate(cp,4);
  return 0;
}
```

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

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

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)

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)


```cpp
// 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
  mylist1.splice (it, mylist2);
  cout << "mylist1 contains:";
  for (it1= mylist1.begin(); it1 != mylist1.end(); it1++)
              cout<< *it1 <<" ";
  cout<<endl;
 // mylist2 become empty. it still points to the 5th element (2)
  cout << "mylist2 contains:";
  for (it1= mylist2.begin(); it1 != mylist2.end(); it1++)
              cout<< *it1 <<" ";
  cout<<endl;

  return 0;
}
```



```cpp
// 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 1 element.
  // iterator it become invalid since element pointed by it is removed
  mylist2.splice (mylist2.begin(),mylist1, it);
 //mylist1 fourth elemement 4 is splice to 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;
}
```

```cpp
// 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, 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

  // 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;

}
```