Object Oriented Programming (CS1143)

Week 14

Department of Computer Science

Capital University of Science and Technology (CUST)

Outline

- Friend Functions and Friend Classes
- Standard Template Library

Friend Functions and Classes

- Private members of a class cannot be accessed from outside the class.
- Occasionally, it is convenient to allow some trusted functions and classes to access a class's private members.
- C++ enables you to use the friend keyword to define friend functions and friend classes so that these trusted functions and classes can access another class's private members.

Example: Friend Class

```
1   #include <iostream>
2   using namespace std;
3
4   class Date
5   {
6       public:
7           Date(int y, int m, int d)
8           {
9               year = y;
10              month = m;
11              day = d;
12          }
13
14      friend class AccessDate;
15
16      private:
17          int year;
18          int month;
19          int day;
20  };
```

```
22  class AccessDate
23  {
24      public:
25          static void p()
26          {
27              Date birthDate(2010, 3, 4);
28              birthDate.year = 2000;
29              cout << birthDate.year << endl;
30          }
31  };
```

```
33  int main()
34  {
35      AccessDate::p();
36
37      return 0;
38  }
```

Description

- The AccessDate class is defined in lines 22–31.
- A Date object is created in the class.
- Since AccessDate is a friend class of the Date class, the private data in a Date object can be accessed in the AccessDate class.
- The main function invokes the static function AccessDate::p() in line 35

Example: Friend Function

```
1   #include <iostream>
2   using namespace std;
3
4   class Date
5   {
6       public:
7           Date(int year, int month, int day)
8           {
9               this->year = year;
10              this->month = month;
11              this->day = day;
12          }
13
14          friend void p();
15
16      private:
17          int year;
18          int month;
19          int day;
20  };
```

```
22  void p()
23  {
24      Date date(2010, 5, 9);
25      date.year = 2000;
26      cout << date.year << endl;
27  }
28
29  int main()
30  {
31      p();
32
33      return 0;
34  }
```

Description

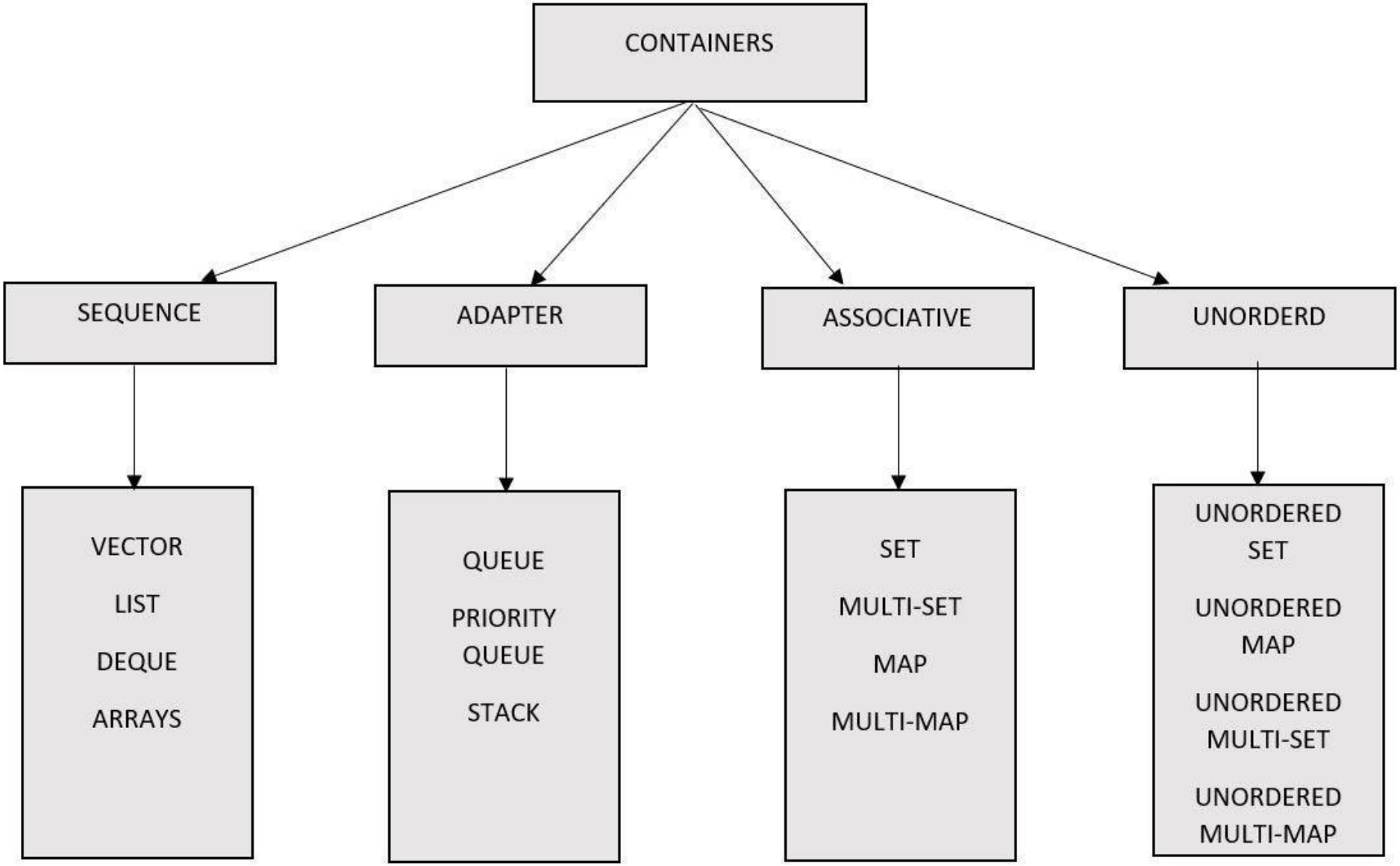
- The program defines the Date class and mentions function p() as a friend.
- Function p is not a member of the Date class but can access the private data in Date .
- In function p , a Date object is created and the private data member year is modified and retrieved.

Outline

- Friend Functions
- Friend Classes
- Standard Template Library

Standard Template Library

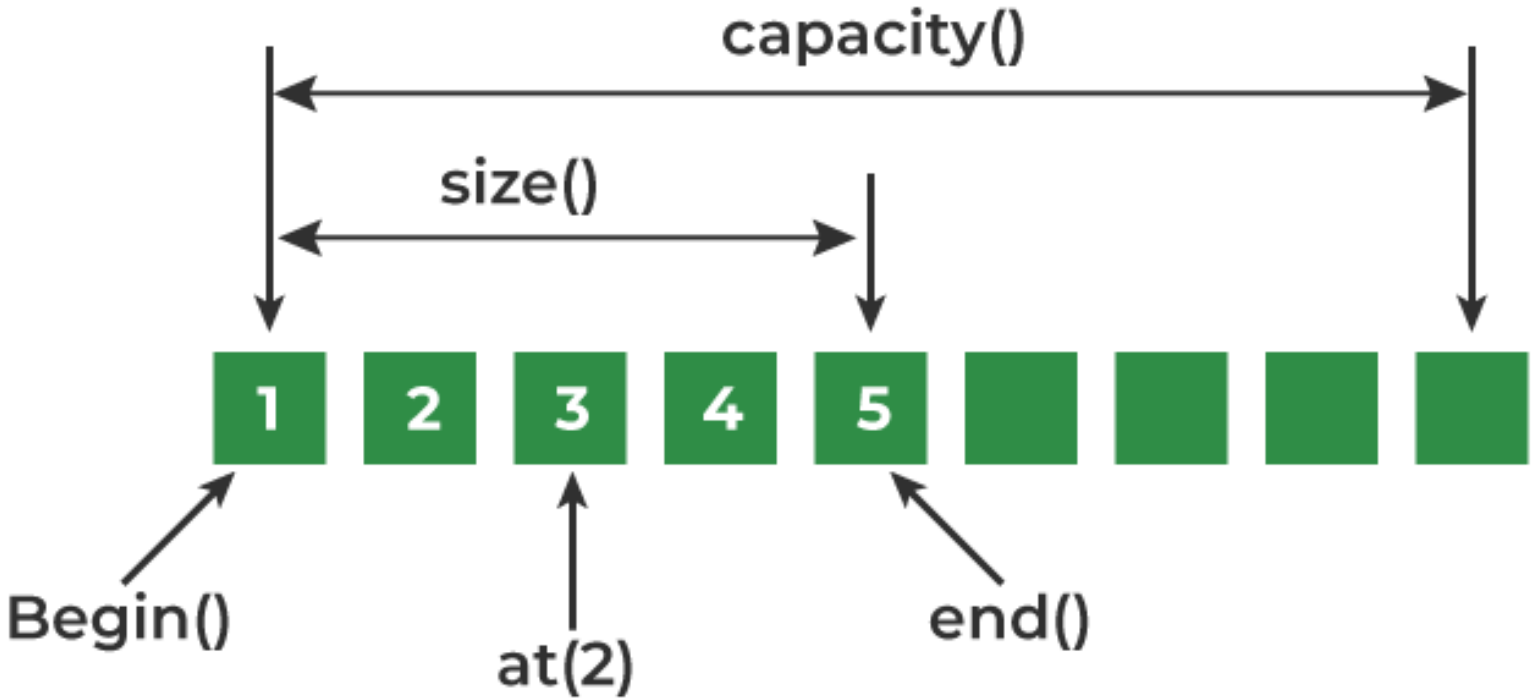
- Standard Template Library (STL) is a software library for the C++ programming language
- It has 3 main components
- Containers
  - The containers are objects that store data
- Iterators
  - Iterators are pointer-like entities used to access the individual elements in a container. Iterators are moved sequentially from one element to another element.
- Algorithms
  - A large number of algorithms to perform activities such as searching and sorting are provided in the STL


CONTAINERS
SEQUENCE
ADAPTER
ASSOCIATIVE
UNORDERD
VECTOR
LIST
DEQUE
ARRAYS
QUEUE
PRIORITY QUEUE
STACK
SET
MULTI-SET
MAP
MULTI-MAP
UNORDERED SET
UNORDERED MAP
UNORDERED MULTI-SET
UNORDERED MULTI-MAP

Containers and Iterators

The vector Class

- The vector class, defined in the <vector> header, implements a sequence container that provides fast random access to any element and fast insertion and deletion at the back (end).
- The vector class is implemented as an array allocated in heap memory but with additional features.
- Like an array, it has an indexing mechanism to access each element.
- Unlike an array, a vector resizes itself whenever more elements are needed.


capacity()
size()
1
2
3
4
5
Begin()
at(2)
end()

Useful Functions (called using object vec)

```
vec.size();                    // Returns the current size
vec.max_size();                // Returns the maximum size
vec.resize(n, value);          // Resizes the vector
vec.empty();                   // Returns true if vector is empty
vec.capacity();                // Returns potential size
vec.reserve(n);                // Reserves more memory locations
```

```
vec.front();                   // Access the first element
vec.back();                    // Access the last element
vec [i];                       // Access the element at index i
vec.at(i);                     // Access the element at index i
```

```
vec.push_back(value);          // Insert value at the back
vec.insert(pos, value)         // Insert value before pos
vec.insert(pos, n, value);     // insert n copies of value before pos
```

```
vec.pop_back();                // Erase the back (last) element
vec.erase(pos);                // Erase the element before pos
vec.erase(first, second);      // Erase elements in the range [first, last)
vec.clear();                   // Erase all elements
```

```cpp
#include <vector>
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    // Constructing a vector of 10 elements and two iterators
    vector <int> vec(10);
    vector <int> :: iterator iter;
    vector <int> :: reverse_iterator rIter;

    // Changing the value of elements
    for (int i = 0; i < 10; i++)
    {
        vec.at(i) = i * i;
    }
    // Printing the elements using the forward iterator
    cout << "Regular navigation: ";
    for (iter = vec.begin() ; iter != vec.end() ; iter++)
    {
        cout << setw(4) << *iter;
    }
    cout << endl;
    // Printing the elements using reverse iterator
    cout << "Reverse navigation: ";
    for (rIter = vec.rbegin() ; rIter != vec.rend() ; rIter++)
    {
        cout << setw(4) << *rIter;
    }
    cout << endl;
    return 0;
}
```

D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Programs\W14-P3.exe

```
Regular navigation:    0   1   4   9  16  25  36  49  64  81
Reverse navigation:   81  64  49  36  25  16   9   4   1   0

--------------------------------
```

Example insert()

```
1   #include <vector>
2   #include <iostream>
3   using namespace std;
4   int main()
5   {
6       // Constructing a vector of 10 elements and two iterators
7       vector <int> vec(10);
8       cout<<"Size: "<<vec.size()<<endl;
9       cout<<"Max Size: "<<vec.max_size()<<endl;
10      cout<<"Capacity: "<<vec.capacity()<<endl;
11
12      // Changing the value of elements
13      for (int i = 0; i < 10; i++)
14          vec.at(i) = i;
15
16      //printing
17      for (int i = 0; i < 10; i++)
18          cout<<vec.at(i)<<" ";
19      cout<<endl<<endl;
20
21      vector <int> :: iterator iter;
22      iter=vec.begin();
23      iter=iter+4;
24      vec.insert(iter, 33);
25
26      cout<<"New Size: "<<vec.size()<<endl;
27      cout<<"New Max Size: "<<vec.max_size()<<endl;
28      cout<<"New Capacity: "<<vec.capacity()<<endl;
29
30      //printing
31      for (int i = 0; i < vec.size(); i++)
32          cout<<vec.at(i)<<" ";
33      cout<<endl;
34      return 0;
35  }
```

D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Progra

```
Size: 10
Max Size: 4611686018427387903
Capacity: 10
0 1 2 3 4 5 6 7 8 9

New Size: 11
New Max Size: 4611686018427387903
New Capacity: 20
0 1 2 3 33 4 5 6 7 8 9
```

Example: Vector of Objects

```
1   #include <iostream>
2   #include <vector>
3   using namespace std;
4
5   class Student
6   {
7       public:
8           int id;
9
10      public:
11          Student(int x)
12          {
13              id = x;
14          }
15  };
16
17  int main()
18  {
19      vector<Student> v;
20
21      Student s1(1001);
22      Student s2(1002);
23      v.push_back(s1);
24      v.push_back(s2);
25
26      cout<<v.at(0).id<<endl;
27
28      return 0;
29  }
```

The list Class

- The list class, defined in the <list> header file, is a sequence container with fast insertion and deletion at any point.
- This means that we can insert or delete easily at any point in a list.
- On the other hand, a list does not support random access for retrieving or changing the value of an element using the index operator or the at() member function

```
1   #include <list>
2   #include <iostream>
3   using namespace std;
4   int main()
5   {
6       // Instantiation of a list object and declaration of a variable
7       list <int> lst;
8       int value;
9       // Inputting five integers and store them in the list
10      for (int i = 0; i < 5; i++)
11      {
12          cout << "Enter an integer: ";
13          cin >> value;
14          lst.push_back(value);
15      }
16      // Printing the list in forward direction
17      cout << "Print the list in forward direction. " << endl;
18      list <int> :: iterator iter1;
19      for(iter1 = lst.begin(); iter1 != lst.end(); iter1++)
20      {
21          cout << *iter1 << " " ;
22      }
23      cout << endl;
24      // Printing the list in backward direction
25      cout << "Print the list in reverse direction. " << endl;
26      list <int> :: reverse_iterator iter2;
27      for (iter2 = lst.rbegin(); iter2 != lst.rend(); iter2++)
28      {
29          cout << *iter2 << " " ;
30      }
31      return 0;
32  }
```

D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Programs\W14-P4.exe

```
Enter an integer: 5
Enter an integer: 6
Enter an integer: 7
Enter an integer: 2
Enter an integer: 9
Print the list in forward direction.
5 6 7 2 9
Print the list in reverse direction.
9 2 7 6 5
--------------------------------
```

Container Adapters

- The Standard Template Library also defines three container adapters that have a smaller interface for easier use.
- The container adapters defined in the library are stack, queue, and priority_queue.
- We cannot apply the algorithms defined in the library to container adapters because they lack iterators; they do not provide the member functions, such as begin and end, to create iterators.

Stack Example

```
1   #include <stack>
2   #include <iostream>
3   using namespace std;
4   int main()
5   {
6       // Instantiation of a stack
7       stack <char> stk;
8       // Creation of two strings and a declaration of a variable
9       string converter("0123456789ABCDEF");
10      string hexadecimal;
11      int decimal;
12      // Inputting a decimal number
13      cout << "Enter a positive integer: ";
14      cin >> decimal;
15
16      // Creation of hexadecimal characters and push them into stack
17      while (decimal != 0)
18      {
19          stk.push(converter [decimal % 16]);
20          decimal = decimal / 16;
21      }
22
23      // Popping characters from stack and pushing into hex string
24      while (!stk.empty())
25      {
26          hexadecimal.push_back(stk.top());
27          stk.pop();
28      }
29      cout << "The hexadecimal number : " << hexadecimal;
30      return 0;
31  }
```

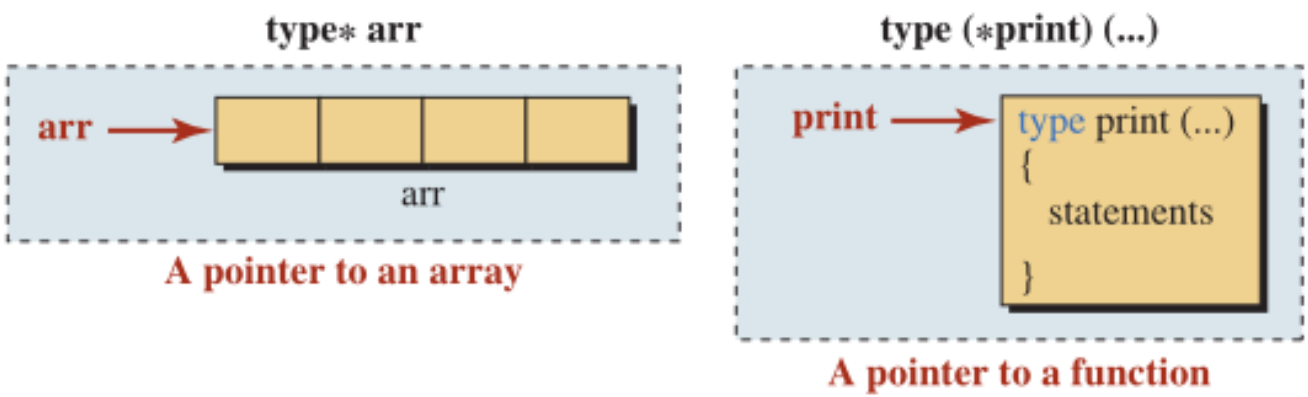
D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Progr

```
Enter a positive integer: 16
The hexadecimal number : 10
--------------------------------
```

Algorithms

Pointer to Functions

- We know that the definition of a function is stored in memory.
- Every entity that is stored in memory has an address.
- In fact, the name of a function is a pointer to the first byte of memory where the function is stored, just as the name of an array is a pointer to the first element of an array.



Calling a function that receives a function *

```
1   #include <iostream>
2   using namespace std;
3
4   // Definition of print function
5   void print (int value)
6   {
7       cout << value << endl;
8   }
9
10  // Definition of fun function
11  void fun (int x, void(*f)(int))
12  {
13      f(x);
14  }
15
16  int main()
17  {
18      fun(24, print); // Calling function fun
19      fun(88, print); // Calling function fun
20      return 0;
21  }
```

Description

- print() is a regular function that prints the value passed to it as a parameter
- fun() has two parameters, an integer and a pointer to a function.
- When name of a function is sent to fun(), the pointer to this function will be stored in f.
- It then calls the function represented as f and passes it the value x.
- In the main, we call fun() with a value and the name of the function we want to send to it. We send “print”.

Using STL algorithm for_each

- for_each applies a function to a range of items in a container.
- The algorithm applies the function defined as the third parameter to the range [first, last)
- The algorithm defines that the third parameter must be a pointer to a function.
- We pass the name print and then we define a function named print with one argument.
- The print function takes the value of its argument from the iterator

```cpp
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;

// Definition of the print function
void print(int value)
{
    cout << value << " ";
}

int main()
{
    // Instantiation of a vector object and storing three values
    vector <int> vec;
    vec.push_back(24);
    vec.push_back(42);
    vec.push_back(73);

    // Using a print function to print the value of each element
    for_each(vec.begin(), vec.end(), print);
    return 0;
}
```

D:\Work Umair\4_C

```
24 42 73
----------
```

Sorting Algorithm from STL

```
1   #include <vector>
2   #include <algorithm>
3   #include <iostream>
4   using namespace std;
5
6   // Definition of print function
7   void print(int value)
8   {
9       cout << value << " ";
10  }
```

```
12  int main()
13  {
14      // Instantiation of a vector object
15      vector <int> vec ;
16      // Pushing six elements into the vector and print them
17      vec.push_back(17);
18      vec.push_back(10);
19      vec.push_back(13);
20      vec.push_back(18);
21      vec.push_back(15);
22      vec.push_back(11);
23      cout << "Original vector" << endl;
24      for_each(vec.begin(), vec.end(), print);
25      cout << endl << endl;
26
27      // Sorting the vector in ascending order and print it
28      cout << "Vector after sorting in ascending order" << endl;
29      sort(vec.begin(), vec.end());
30      for_each(vec.begin(), vec.end(), print);
31      cout << endl << endl;
32
33      // Sorting the vector in descending order and print it
34      cout << "Vector after sorting in descending order" << endl;
35      sort(vec.begin(), vec.end(), greater <int>());
36      for_each(vec.begin(), vec.end(), print);
37      cout << endl << endl;
38      return 0;
39  }
```

D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Programs\W14-P8.exe

```
Original vector
17 10 13 18 15 11

Vector after sorting in ascending order
10 11 13 15 17 18

Vector after sorting in descending order
18 17 15 13 11 10
```

About STL

- This was just a brief overview of some useful containers and algorithms from STL
- It is a big library and contains many useful features.
- A detailed discussion of STL is beyond the scope of this course.

This is all for Week 14