Standard Template Library

STL

- While C++ is a powerful language in its own right, recent additions have added even more capabilities
- Of these, perhaps the most influential is the ***Standard Template Library (STL)***
- It provides three generic entities: containers, iterators, and algorithms, and a set of classes that overload the function operator called function objects
- It also has a ready-made set of common classes for C++, such as containers and associative arrays
- These can be used with any built-in type and with any user-defined type that supports some elementary operations

STL

Containers

- A container is a data structure that is typically designed to hold objects of the same type
- Although the number of possible organizations of this data is unlimited, only a few are practical and implemented in the STL
- Containers are implemented as template classes whose methods specify operations on the data in the structures as well as the structures themselves
- A number of these methods are common to all containers, while some are more specific to the container they are defined in
- The data stored in containers can be of any type and must supply some basic methods and operations
- This is especially necessary if pointers are involved

STL

Iterators

- An iterator is an object that accesses the elements of a container
- The STL implements five types of iterators
  - Input iterators, which can read a sequence of values
  - Output iterators, which can write a sequence of values
  - Forward iterators, which can be read, written to, or moved forward
  - Bidirectional iterators, which behave like forward iterators but can also move backwards
  - Random iterators that can move freely in any direction at one time
- Essentially, an iterator is a generalized pointer, and can be dereferenced and manipulated like a pointer
- These capabilities make iterators a major feature that allows the generality of the STL

Vectors in the STL

- A vector is one of the simplest containers in the STL
- The elements of vectors are stored contiguously in memory and the entire structure is treated like a dynamic array
- Like dynamic arrays, vectors exhibit low memory utilization, good locality of reference, and good data cache utilization
- Vectors allow random access, so elements can be referenced using indices
- The vector’s structure is able to efficiently allocate the memory needed for data storage
- This is useful for storing lists whose length may not be known prior to setting up the list but where removal is rare
- Vectors are incorporated into code by including the vector library:

```
#include <vector>
```

Lists in the STL

- The STL implements lists as generic doubly linked lists with head and tail pointers
- The class is available in a program through the directive #include <list>
- How to iterate over this container → `#include <iterator>`

```
void showlist(list<int> g)
{
   list<int>::iterator it;
   for (it = g.begin(); it != g.end(); ++it)
      cout << '\t' << *it;
   cout << '\n';
}
```

Lists in the STL

```
list<int> mylist;
```

Some useful Lists methods:

```
front()

back()

push_front(int i)

push_back(int i)

pop_back(int i)
```