# OBJECT ORIENTED PROGRAMMING

Lab 8

# OBJECT ORIENTED PROGRAMMING WITH C++

- Chapter Objectives
  - Namespaces
  - static class members
  - Standart Template Library
  - The Container Classes

# OBJECT ORIENTED PROGRAMMING WITH C++

- Namespaces
  - Namespaces are a relatively recent addition to C++.
  - Their purpose is to localize the names of identifiers to avoid name collisions.
  - In the C++ programming environment, there has been an explosion of variable,function, and class names.
  - Prior to the invention of namespaces, all of these names competed for slots in the global namespace, and many conflicts arose.
  - Name collisions were compounded when two or more third-party libraries were used by the same program.
  - In this case, it was possible-even likely-that a name defined by one library would conflict with the same name defined by another library.

# OBJECT ORIENTED PROGRAMMING WITH C++

- Namespaces
  - The namespace keyword allows you to partition the global namespace by creating a declarative region.
  - In essence, a namespace defines a scope.
  - The general form of namespace is shown here:

    ```
    namespace name
    {
    // declarations
    }
    ```

  - Anything defined within a namespace statement is within the scope of that namespace.

# OBJECT ORIENTED PROGRAMMING WITH C++

- static class members
  - When you declare a member variable as static, you cause only one copy of that variable to exist-no matter how many objects of that class are created.
  - Each object simply shares that one variable.
  - Remember, for a normal member variable, each time an object is created, a new copy of that class are created.
  - Each object simply shares that one variable.

# OBJECT ORIENTED PROGRAMMING WITH C++

- static class members
  - Also, the same static variable will be used by any classes derived from the class that contains the static member.
  - Although it might seem odd when you first think about it, a static member variable exists before any object of its class is created.
  - In essence, a static class member is a global variable that simply has its scope restricted to the class in which it is declared.
  - In fact, as you will see in one of the following examples, it is actually possible to access a static member variable independent of any object.

# OBJECT ORIENTED PROGRAMMING WITH C++

- static class members
  - When you declare a static data member within a class, you are not defining it.
  - Instead, you must provide a definition for it elsewhere, outside the class.
  - To do this, you redeclare the static variable, using the scope resolution operator to identify which class it belongs to.
  - All static member variables are initialized to 0 by default.
  - However, you can give a static class variable an initial value of your choosing, if you like.
  - Keep in mind that the principal reason static member variables are supported by C++ is to <u>avoid</u> the need for global variables.

# OBJECT ORIENTED PROGRAMMING WITH C++

- static class members
  - A member function declared as static can access only other static members of its class. (Of course, a static member function can access non-static global data and functions.)
  - A static member function does not have a this pointer.
  - Virtual static member functions are not allowed.
  - Also, static member functions cannot be declared as const or volatile.
  - A static member function can be invoked by an object of its class, or it can be called independent of any object, via the class name and the scope resolution operator.

# OBJECT ORIENTED PROGRAMMING WITH C++

- Standart Template Library
  - Standard Template Library is large and its syntax is, at times, rather intimidating, it is constructed and what elements it employs.
  - Therefore, before looking at any code examples, an overview of the STL is warranted.
  - At the core of the Standard Template Library are three foundational items: containers, algorithms, and iterators.
  - These items work in conjunction with one another to provide off-the-shelf solutions to a variety of programming problems.

# OBJECT ORIENTED PROGRAMMING WITH C++

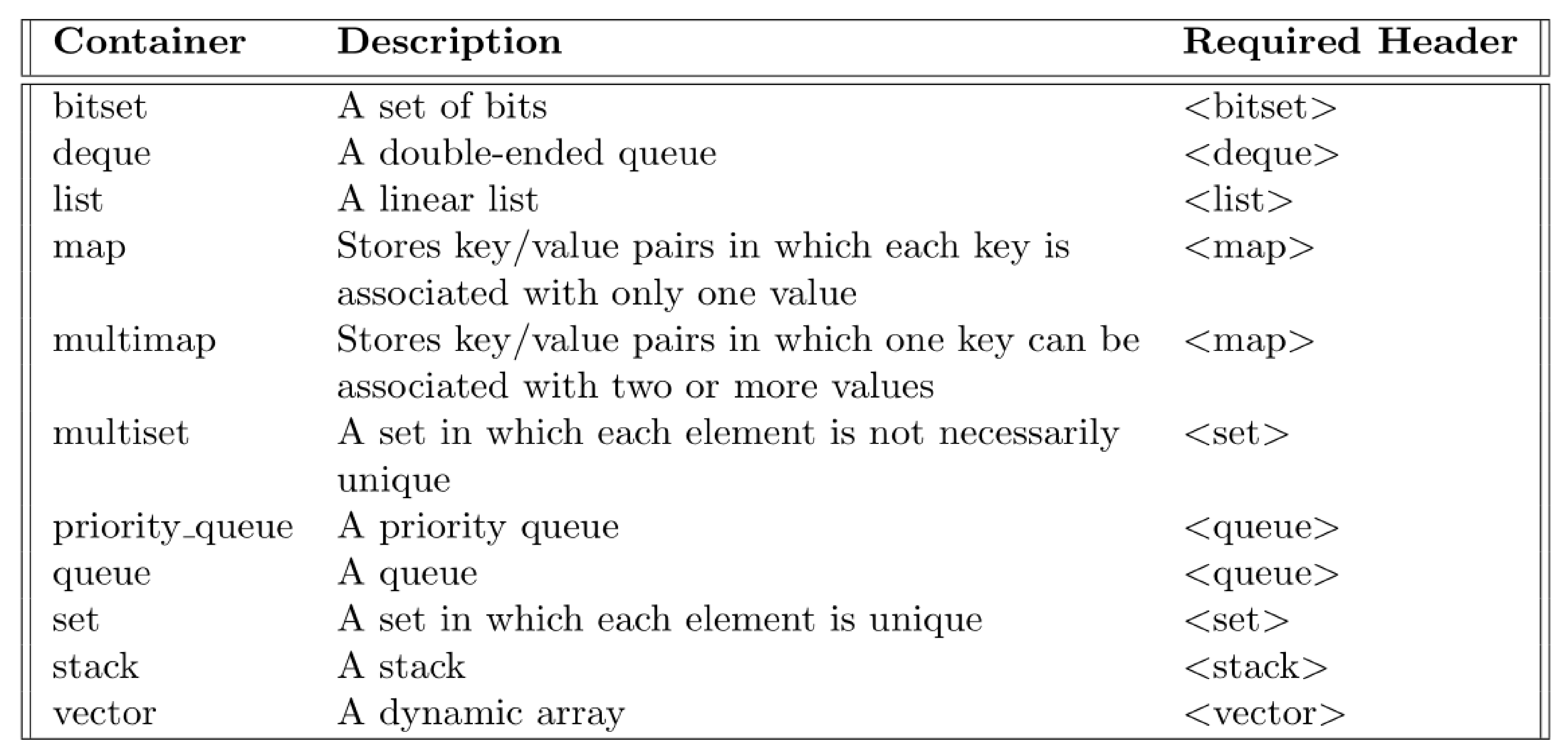
- Standart Template Library
  - Containers are objects that hold other objects. There are several different types of containers.
  - For example, the vector class defines a dynamic array, queue creates a queue, and list provides a linear list.
  - In addition to the basic containers, the STL also defines associative containers, which allow efficient retrieval of values based on keys.
  - For example, the map class defines a map that provides access to values with unique keys, Thus, a map stores a key/value pair and allows a value to be retrieved when its key is given.
  - Each container class defines a set of functions that can be applied to the container. For example, a list container includes functions that insert, delete, and merge elements.
  - A stack includes functions that push and pop values.

# OBJECT ORIENTED PROGRAMMING WITH C++

- Standart Template Library
  - Algorithms act on containers. Some of the services algorithms perform are initializing, sorting, searching, and transforming the contents of containers.
  - Many algorithms operate on a sequence, which is a linear list of elements within a container.
  - Iterators are objects that are, more or less, pointers. They give you the ability to cycle through the contents of a container in much the same way that you would use a pointer to cycle through an array.

# OBJECT ORIENTED PROGRAMMING WITH C++

- The Container Classes



| Container | Description | Required Header |
|---|---|---|
| bitset | A set of bits | <bitset> |
| deque | A double-ended queue | <deque> |
| list | A linear list | <list> |
| map | Stores key/value pairs in which each key is associated with only one value | <map> |
| multimap | Stores key/value pairs in which one key can be associated with two or more values | <map> |
| multiset | A set in which each element is not necessarily unique | <set> |
| priority_queue | A priority queue | <queue> |
| queue | A queue | <queue> |
| set | A set in which each element is unique | <set> |
| stack | A stack | <stack> |
| vector | A dynamic array | <vector> |