Operator Overloading.

Const Member Functions cannot modify data members of a class, they can only access it.

External functions cannot access the private members of a class.

Derived class Member Functions cannot access the Private members of a class.

Derived class Member Functions can only access the public and protected members of a class.

External functions can only access public members of a class.

Own member functions can access all types of members of the class (public, protected, private).

Objects of Abstract Classes are not meant to be instantiated.

3 types of Access Specifiers in C++:
1. Public
2. Protected
3. Private

An Abstract Class is one which has at least 1 Pure Virtual Function.

Syntax of a Pure Virtual Function:

```
virtual void show() = 0;              // pure virtual function
```

A Pure Virtual Function of an Abstract Class must always be overidden in its derived classes.

Virtual Destructors

Virtual Base Classes

The concept of virtual base classes is only applicable to the case of Multiple Inheritance to avoid ambiguity.

Friend Function

The concept of data hiding and data encapsulation mean that non – member functions should not be able to access an objects non – members functions.

1> `Friend Functions` can be used to act as a bridge between the objects of 2 different classes.

i.e. friend functions are applicable in situations in which a function requires access to private and protected members of objects of two different classes.

2> A minor point: Remember that a class can't be referred to until it has been declared. Class beta is referred to in the declaration of the function frifunc() in class alpha , so beta must be declared before alpha . Hence the declaration
class beta;
at the beginning of the program.

3> Friend Functions must be used sparingly as they go against the basic philosophy of 'Data Hiding' in Object Oriented Programming (OOPs).

4> Friend Functions are also used in situations when we requires a more natural and intuitive functional notation for operations on objects of different classes which cannot be provided by normal member function.

Friend Classes

In a friend class of a given class all the member functions of the friend class are friend functions of the given class.

And therefore all the member functions of the friend class have access to the private members of the given class.

Static Functions

Static Functions are not replicated for every single object of a class. A single copy of the static function is maintained for a particular class of objects.

This type of functions are useful in situations where we need to keep a record of the number of objects of a particular class.

These types of functions are required when a single function needs to keep track of some information about the objects of a particular class.

We get to know that local objects are stored in a stack by observing the order of constructor and destructor invocation.

Static Data Members

1. Static Data Members of a class must always be initialized outside the class.

```
int X::id = 0
  Where, X → class
         id → static data member of X.
```

2.

Assignment Operator Overloading

istream (>>) Operator Overloading

Syntax: ***istream& operator >> (istream& out, <classtype> x);***

```
out >> Name >> x.b >> x.a >> … etc.;
return out;                          // return the istream object.
```

ostream (<<) Operator Overloading

Syntax: ***ostream& operator << (ostream& out, <classtype> x);***

```
out << Name << " " << age << … etc.;
return out;                         // return the ostream object.
```

Copy Constructor

Difference b/w no – arg , one-arg , copy constructor and overloaded assignment operator.

Suppose there is an `Alpha` class.

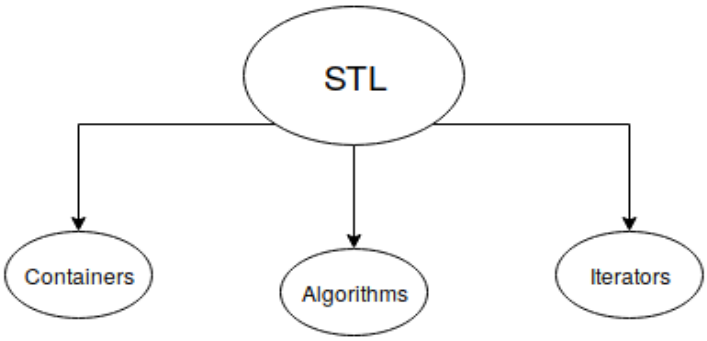
Then,

```
Alpha a;                        // Invokes the No - Arg Constructor.
Alpha a(5);                     // Invokes the One - Arg Constructor.
Alpha a(a1);                    // Invokes the Copy - Constructor.
Alpha a = a1;                   // Alternative form of copy
                                // initialization.

Alpha a;
a = a1;                         // Invokes the default assignment
                                // operator if it is not overloaded.

                                // Else invokes the overloaded
                                // assignment operator.
```

The Standard Template Library (STL)

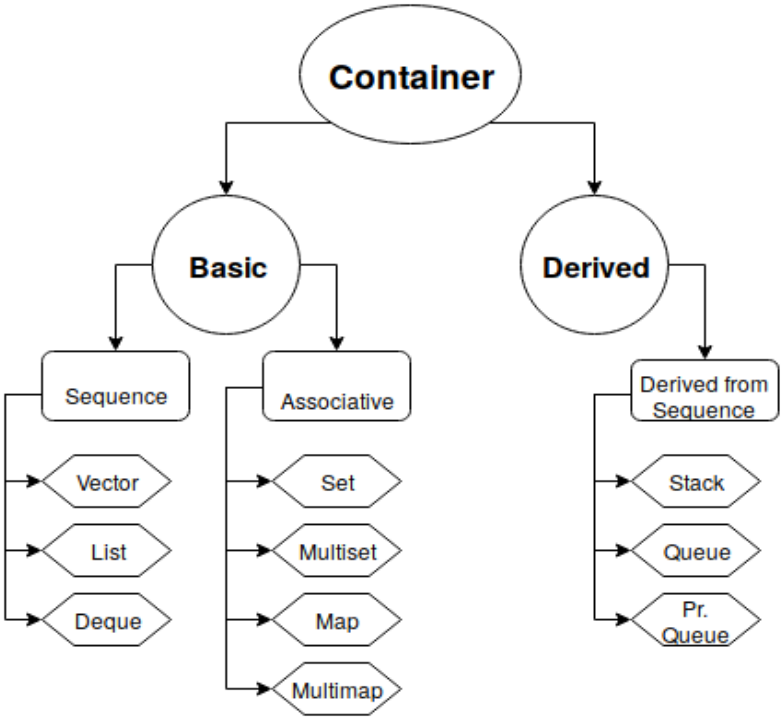


Iterators are a key part of the STL because they connect algorithms with containers.

Containers

1. A container is a way to store data, whether the data consists of built-in types such as **int** and **float** , or of **class objects**.

2. There are **seven basic** kinds of **containers** and **three containers derived** from the basic kind.



Sequence Containers

A Container in which data objects are stored in a linear fashion, i.e. which can be visualized as a line containing the objects one are the other and where the Objects can be referred to by their location in the line is called a **Sequence Container.** In a sequence container each element (except the 1st and last elements) are preceded and followed by objects.

According to the above definition an **Array** is also a **Sequence Container** built into the **C/C++** language.

Comparison of Sequence Containers

Container	Characteristic	Advantages / Disadvantages
Array	Fixed Size	1. Quick Random Access (via Index) 2. Slow to remove and add elements at the middle. 3. Size cannot be changed at Runtime.
Vector	Relocatable, Expandable Array	1. Quick Random Access (via Index) 2. Slow to insert or erase at the middle. 3. Fast to insert or erase from the end.
List	Doubly Linked List	1. Quick to insert or delete at any location. 2. Quick access to the ends. 3. Slow Random Access.

Deque	**D**ouble **E**nded **QUE**ue Combination of Queue and Stack. Similar to a vector but can be accessed at either end.	1. Quick Random Access(via Index) 2. Slow to insert or erase at the middle. 3. Quick access to the ends.

Associative Container

An associative container is not sequential; instead it uses keys to access data. The keys, typically numbers or stings, are used automatically by the container to arrange the stored elements in a specific order. It's like an ordinary English dictionary, in which you access data by looking up words arranged in alphabetical order.

Basically the Sequence Containers are a special case of Associative Containers in which the position of an object acts as the key for the object.

Member Function Common to all Containers

Name	**Purpose**
`size()`	Return the number of items in the Container.
`empty()`	Returns true if a Container is empty.
`max_size()`	
`begin()`	Returns an iterator to the start of a container for iterating forwards.
`end()`	Returns an iterator pointing to the theoretical element after the last element.
`rbegin()`	Returns a reverse iterator to the end of the container for iterating backwards.
`rend()`	

Important Properties of the C++ STL Sets:

1. Sets are Associative Containers, i.e. they store values referenced by a key value and not their absolute position.
2. Elements in a set are unique.
3. Sets in C++ are similar to sets in python.
4.

Important Properties of the C++ STL Maps:

1. Maps are also **Associative Containers** and are similar to **dictionary** in **python.**

Algorithms

STL Algorithms are functions which perform certain common manipulation operations on the STL containers. These functions are not member functions / friend functions of these STL container classes but are generalized template functions which can operate on different containers and even user defined container classes provided they have certains basic member functions.

Some Commonly Used STL Algorithms

Name	**Purpose**
`find`	Returns 1st element equivalent to a specified value.
`sort`	Sorts the values in a container according to a specified ordering.
`equal`	Compares contents of two containers and returns true if all corres. elements are equal.
`search`	Looks for a sequence of values in one container that correspond to the same sequence in another container.
`copy`	Copies a sequence of values from one container to another. (Or to a different location in the same container.)
`swap`	Exchange the value in one location with a value in another location.
`iter_swap`	Exchanges a sequence of values in one location with a sequence of values in another location.
`fill`	Copies a value into a sequence of locations.
`count`	Count the number of elements having a specified value.
`merge`	Combine two sorted ranges of elements to form larger sorted range.
`accumulate`	Return the sum of elements in a given range.
`for_each`	Run a specified function for each given element in the array.

Important Technical Details about the STL

Ranges

The first two parameters to find() specify the range of elements to be examined. These values are specified by iterators. In this example we use normal C++ pointer values, which are a special case of iterators.
The first parameter is the iterator of (or in this case the pointer to) the first value to be examined. The ***second parameter is the iterator of the location one past the last element*** to be exam-

ined. Since there are 8 elements, this value is the first value plus 8. This is called a ***past-the-end*** value; it points to the element just past the end of the range to be examined.

The arguments to algorithms such as search() don't need to be the same type of container. The source could be in an STL vector, and the pattern in an array, for example. ***This kind of generality is a very powerful feature of the STL***.

Templates and Exceptions