

Agenda

- STL
- Nested and Local class
- ~~Streams~~
- ~~File IO~~

STL

- We can not divide template code into multiple files.
- Standard Template Library(STL) is a collection of readymade template data structure classes and algorithms.
- The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc.
- It is a library of container classes, algorithms, and iterators.
- It is a generalized library and so, its components are parameterized.
- Working knowledge of template classes is a prerequisite for working with STL.
- STL has 4 components:
 1. Containers
 2. Algorithms
 3. Function Objects
 4. Iterators

1. Container

- Containers or container classes to store objects and data.
- The C++ container library categorizes containers into four types:
 1. Sequence containers
 2. Associative containers
 3. Unordered associative containers
 4. Sequence container adapters

```
STL
|
├── Containers
│   ├── Sequence Containers
│   │   ├── vector
│   │   ├── deque
│   │   ├── list
│   │   └── forward_list
│   └── Associative Containers
│       └── set
```

```
├── multiset
├── map
├── multimap
├── Unordered Associative Containers
│   ├── unordered_set
│   ├── unordered_multiset
│   ├── unordered_map
│   └── unordered_multimap
├── Container Adapters
│   ├── stack
│   ├── queue
│   └── priority_queue
```

2. Algorithm

- They act on containers and provide means for various operations for the contents of the containers.
 - Sorting
 - Searching

3. Functions

- The STL includes classes that overload the function call operator.
- Instances of such classes are called function objects or functors.
- Functors allow the working of the associated function to be customized with the help of parameters to be passed.

4. Iterators

- As the name suggests, iterators are used for working upon a sequence of values.
- They are the major feature that allows generality in STL.
- Iterators are used to point at the memory addresses of STL containers.
- They are primarily used in sequences of numbers, characters etc.
- They reduce the complexity and execution time of the program.

Sequence Containers

- Sequence containers are used for data structures that store objects of the same type in a linear manner.
- The STL Sequence Container types are:
 - vector: A dynamic array that can grow and shrink in size. It provides fast random access to elements and efficient insertion and deletion at the end.
 - deque: A double-ended queue that supports efficient insertion and deletion at both ends. It provides similar functionality to vector but may have better performance for inserting and deleting elements at the beginning.
 - list: A doubly-linked list that allows for efficient insertion and deletion of elements at any position. It does not provide random access to elements and has slower traversal compared to vector and deque.

- `forward_list`: A singly-linked list that provides similar functionality to `list` but with reduced memory overhead. It allows for efficient insertion and deletion at the beginning and after an element.

Associative Containers

- Associative containers implement sorted data structures that can be quickly searched.
- `set` : collection of unique keys, sorted by keys
- `map` : collection of key-value pairs, sorted by keys, keys are unique
- `multiset` : collection of keys, sorted by keys
- `multimap` : collection of key-value pairs, sorted by keys

Container adaptors

- Container adaptors provide a different interface for sequential containers.
- `stack` : adapts a container to provide stack (LIFO data structure)
- `queue` : adapts a container to provide queue (FIFO data structure)

Vector

- Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container.
- Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators.
- In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes the array may need to be extended. Removing the last element takes only constant time because no resizing happens.
- When the vector needs to grow beyond its current capacity, it typically doubles its capacity.
- Inserting and erasing at the beginning or in the middle is linear in time.
- the iterator in the vector is a Random Access Iterator that Supports all iterator operations including arithmetic (e.g., `+`, `-`), comparison (`<`, `>`, etc.), and dereferencing (`*`, `[]`).

map

- `map` is a sorted associative container that contains key-value pairs with unique keys.
- Keys are sorted by using the comparison function `Compare`.
- Search, removal, and insertion operations have logarithmic complexity.
- Maps are usually implemented as Red-black trees
- Iterators of `map` iterate in ascending order of keys, where ascending is defined by the comparison that was used for construction.
- Iterator of `map` are bidirectional iterator that supports dereferencing (`*`, `->`) and bidirectional movement (`++`, `--`).

iterator

- An iterator in C++ is an object that enables traversal over the elements of a container (such as `std::vector`, `std::list`, `std::map`, etc.) and provides access to these elements.
- Iterators are a fundamental part of the Standard Template Library (STL) and are designed to abstract the concept of element traversal, making it possible to work with different containers in a consistent

manner.

- Key Characteristics of Iterators:

1. Traversal: Iterators allow you to move through the elements of a container, one element at a time.
2. Access: Iterators provide access to the element they point to, typically through the dereference operator (*).
3. Type-Specific: Iterators are strongly typed, meaning that an iterator for an `std::vector` will be different from an iterator for an `std::list`.

Types of Iterators:

1. Input Iterators:

- Can read elements from a container. Only allow single-pass access (i.e., you can only move forward through the container).

2. Output Iterators:

- Can write elements to a container.
- Also allow only single-pass access.

3. Forward Iterators:

- Can read and write elements.
- Support multi-pass traversal, meaning you can go through the container multiple times.

4. Bidirectional Iterators:

- Can move both forward and backward in a container.
- Support all operations of forward iterators, with the additional ability to decrement the iterator.

5. Random Access Iterators:

- Provide all the capabilities of bidirectional iterators.
- Allow direct access to any element in the container using arithmetic operations like addition and subtraction.

Common Operations on Iterators:

- Dereferencing (*): Access the element the iterator points to.
- Incrementing (++): Move the iterator to the next element.
- Decrementing (--): Move the iterator to the previous element (not supported by input or output iterators).
- Equality/Inequality (==, !=): Compare iterators to check if they point to the same position.
- Addition/Subtraction (+, -): For random access iterators, allows moving the iterator by a specific number of elements.

Local Class

- If inside a function you declare a class then such classes are called as local classes.

- Inside local class you can access static and global members but you cannot access the local members declared inside the function where the class is declared.
- Inside local classes we cannot declare static data member however defining static member functions are allowed.
- As every local variable declared in function goes on its individual stack frame, such variables cannot be accessed in the local class functions.
- static and global variables get space on data section which are designed to be shared.
- In a class static data members are designed to be accessed using classname and ::, the static members are designed to be shared however in local classes we cannot access them outside the function in which they are declared.
- Hence there is no purpose of keeping static data members inside local classes

Nested class

- A class declared inside another class is called as nested class
- Inside Nested class we can access private static members of outer class directly.
- A nested class can access all the private and public members of outer class directly on the outer class object
- Inside Nested class you can access static and global variables.
- As static and global variables are designed to be shared they are accessible inside the nested class.
- Data members get the memory only when object is created and hence the nested class cannot access outer class data non static data members directly as they do not get memory inside them.