STL

- The C++ STL (Standard Template Library) is a powerful set of C++ template classes to provide general-purpose classes and functions with templates that implement many popular and commonly used algorithms and data structures like vectors, lists, queues, and stacks.
- STL provides programmers to store the data effectively, and do manipulation in stored data. These are the general-purpose templates of classes and functions that help in implementing the basic algorithms and data structures like vector, lists, queue, stack, etc.

It is a set of C++ template classes that provide generic classes and function can be used to implement data structures and algorithms.

STL is mainly composed of :

1. Algorithms
2. Containers
3. Iterators

STL is a generic library , i.e a same container or algorithm can be operated on any data types , you don’t have to define the same algorithm for different type of elements.

example you can very easily define a linked list in a single statement by using list container of container library in STL , saving your time and effort.

example , sort algorithm will sort the elements in the given range irrespective of their data type , we don’t have to implement different sort algorithm for different data types

It is a set of C++ template classes that provide generic classes and function that can be used to implement data structures and algorithms .STL is mainly composed of :

1. Algorithms
2. Containers
3. Iterators

C++: Algorithms in STL

STL provide number of algorithms that can be used of any container, irrespective of their type.

Algorithms library contains built in functions that performs complex algorithms on the data structures.

For example: one can reverse a range with reverse() function, sort a range with sort() function, search in a range with binary_search() and so on.

Algorithm library provides abstraction, i.e you don't necessarily need to know how the the algorithm works.

C++: Containers in STL

Container library in STL provide containers that are used to create data structures like arrays, linked list, trees etc.

These container are generic, they can hold elements of any data types,

for example: vector can be used for creating dynamic arrays of char, integer, float and other types.

C++: Iterators in STL

Iterators in STL are used to point to the containers.

Iterators are used to point at the memory addresses of STL containers.

Iterators actually acts as a bridge between containers and algorithms.

Operations of iterators :-

1. begin() :- This function is used to return the **beginning position** of the container.

2. end() :- This function is used to return the ***after*** **end position** of the container.

3. advance() :- This function is used to **increment the iterator position** till the specified number mentioned in its arguments.

4. next() :- This function **returns the new iterator** that the iterator would point after **advancing the positions** mentioned in its arguments.

5. prev() :- This function **returns the new iterator** that the iterator would point **after decrementing the positions** mentioned in its arguments.

For example: sort() algorithm have two parameters, starting iterator and ending iterator, now sort() compare the elements pointed by each of these iterators and arrange them in sorted order, thus it does not matter what is the type of the container and same sort() can be used on different types of containers.

Application of STL

STL being generic library provide containers and algorithms which can be used to store and manipulate different types of data thus it

saves us from defining these data structures and algorithms from the scratch.

Because of STL, now we do not have to define our sort function every time, instead we can just use the generic container and algorithms in STL.

This saves a lot of time, code and effort during programming.

if we are reading input stream of elements and we do not know the upper bound of the number of elements, there are high chances of occurrence of index_out_bound situation or unwanted termination of the program.

initialize the array with maximum size allowed by the complier, i.e. 10^6 elements per array, but that is highly space consuming approach and there is a wastage of space if number of elements to be entered are way too less, thus this approach is never used in programming.

Solution of the above problem is dynamic arrays!

have dynamic size, i.e. their size can change during runtime.

SYNTAX for creating a vector is: vector< object_type > vector_name;

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> v ;

    v.push_back(11);
    v.push_back(1);
    v.push_back(21);
    v.push_back(6);
    v.push_back(9);

    vector<int>::iterator it;

    for(it = v.begin(); it != v.end(); it++)
    {
        cout << *it <<" ";   // for printing the vector
    }
    cout<<v.size();
    sort(v.begin(), v.end());

    cout << "Sorted \n";
     for(it = v.begin(); it != v.end(); it++)
    {
        cout << *it <<" ";   // for printing the vector
    }
    return 0;
}
```