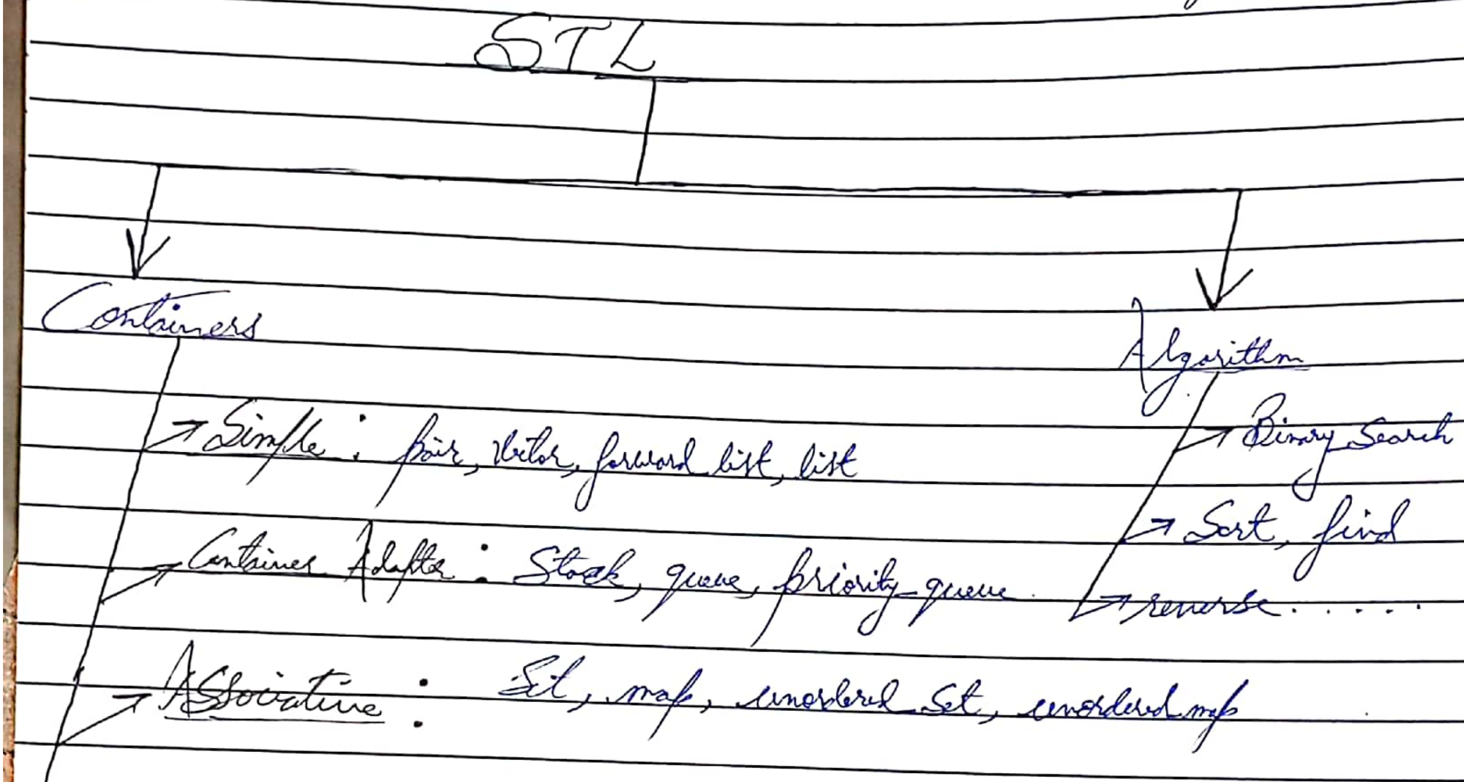
#II Standard Template Library

→ It is a Standard implementation of data structure and algorithm. Using STL, we can make our code faster and its time Saves, It is the features which make C++ unique & faster than others



STL

Containers

- Simple : pair, vector, forward list, list
- Container Adapter : Stack, queue, priority_queue
- Associative : Set, map, unordered_set, unordered_map

Algorithm

- Binary Search
- Sort, find
- reverse.....

Eg :→ Sort (arr, arr+n)

binary_search (arr, arr+n, 8).

#II Iterators

→ It is just like object (like pointer) that point to element inside the Container. It plays very critical roles in connecting algorithm to the Container.

Syntax :→ Containers_name iterator :: iterator_Variable

Eg. → vector <int>:: iterator i; Or auto i;

begin ( ) → Address of first element., end ( ) → Point Memory location beyond last element.

Simple Container [Pair]

(*) It is a Simple Container of having only two elements in it.

(*) It can be of Same type or different types, it can be assigned, copied, compared.

Syntax :→ pair (data-type 1, data-type 2) Pair_name;

Example.

```
int main ( )
{
    pair<int, int> p1 (10, 20);
    pair <int, String> p2 (10, "Geeksforlemon");
    Cout << p1. first << " " << p1. Second << endl;
    Cout << p2. first << " " << p2. Second << endl;
}.
```

(**) If we do not initilize the pair it gets default value 0 in case of int and empty String in case of string.

Comparison of Pair

```
int main ( )
{
  pair < int, int > p1 (1, 12), p2 (9, 12);
cout << (p1 == p2); It will true if both values are Same.
Cout << (p1 != p2); Both Same = 0, Both or Any Diff = 1.
Cout << (p1 > p2); Check first Value only, if Same then Check Second
Cout << (p1 < p2); Check first Value only, if Same then Check Second
}
}.
```

```
O/P => 0
       1
       0
       1.
```

Vector

→ It is a alternative of array whose Size increase automatically.

Ⓡ Advantages of Vector over Array

① Dynamic Size	④ Can be easily returned.
② Rich Library f^n	⑤ Efficient
③ No Need to pass Size	⑥ Fast

⊛ Return Vector in f^n

```
int fun()
{ vector <int> V;

    return V; //Valid
}.
```

⊛ Return array in f^n

```
int fun()
{
    int arr[100];

    return arr; // Not Valid
}.
```

#④ Vector Declaration & Traversal

Syntax ⇒ vector < data_type > vector_name;

⊛ Declaration of Vector with Size ⇒ vector <data-type> vector-name (N);

⊛ Accessing the vector element ⇒

```
vector <int> V;
for(int i = 0; i < V.size(); i++)
    cout << V[i];
          OR
for(int x : V)
    cout << x;
```

⊛ Widely Used function of Vector

① push_back ⇒ push element from back. push_back(6), push_back(arr[i])

② pop_back ⇒ pop element from back pop_back(6);

③ insert() ⇒ insert element at i^{th} index V.insert(V.begin()+i, 7).

④ erase() ⇒ remove element at i^{th} index or range: V.erase(V.begin());

⑤ Size () → Return no. of element in vector, V. Size();

⑥ empty() → Check vector empty or not, V.empty();

⑦ Sort () → Sort the vector ~~V~~. Sort (V.begin(), V.end());

⑧ reverse () ⇒ Reverse the vector in reverse (V.begin(), V.end());

⑨ binary_Search () ⇒ Check element is present or not, binary_search(V.begin(), V.end(), 7)

Forward List

→ It is used to implement singly linked list. It is very efficient terms of time complexity. It is generally used in chaining, hashing, adjacency list representation of graph.

Syntax ⇒ forward_list <int> l;

Display the element of forward list ⇒ for (int x : l)
cout << x << " ";

Time Complexity

① insert_after() ⇒ O(1)

② erase_after() ⇒ O(1)

③ push_front() ⇒ O(1)

④ reverse() ⇒ O(n)

⑤ sort() O(n log n)

⑥ remove() ⇒ O(n)

⑦ assign() ⇒ O(1)

⑧ pop_front() ⇒ O(1).

List

→ It is used to implement the double linked list. It has both next, prev element's pointer as the doubly linked list.

Syntax ⇒ list <int> l;

Traversal ⇒ for (auto itr = l.begin(); itr != l.end(); itr++)

cout << *itr << " ";

Time Complexity :

1. front() ⇒ O(1)
2. back() ⇒ O(1)
3. size() ⇒ O(1)
4. begin() ⇒ O(1)
5. end() ⇒ O(1)
6. eraser(itr) ⇒ O(1).
7. push_front() ⇒ O(1)
8. push_back() ⇒ O(1)
9. pop_front() ⇒ O(1)

11. pop_back() ⇒ O(1)
12. reverse() ⇒ O(N)
13. remove() ⇒ O(N)
14. sort() ⇒ O(N*logN)