STL means standard Templet library. It provides libraries for implimentation of datastructurs and algorithums.

It provides more time to you while you are at CP or at any interview.

STL (C++)

- Containers (implimentations of basic data structurs)
- algorithums (implimentations of algorithums)

Containers:-

- Simple: pair, vector, forward list, list
- Container adapters: stack, queue, priority-queue
- Associative: set, map, unordered set, unordered map ...

Algorithms : binary-search, find, reverse, sort ...

→ Iterators :-

iterator gives you address of an element in a container. We can use iterator to traverse through containers or do other things also.

iterators are similar to pointers but not exactly the pointers, we can do pointers arithmetic on some iterators, depending upon containers, and we can dereference them to get value.

Syntax of declaration :-

{{container name}}<datatype>:: iterator i = v.begin();

here, begin() is function which returns address of first element in container. So, v.begin() returns address of first element of vector v.

end() function returns address of element beyond last element.

ex.-

```
#include<iostream>
#include<vector>
using namespace std;

int main()
{
   vector<int> v={10,20,30,40,50};

   vector<int>::iterator i=v.begin();
   cout<<(*i)<<" ";
    i++;
   cout<<(*i)<<" ";
    i=v.end();
    i=--;
   cout<<(*i)<<" ";
    return 0;
}
```

next() function gives iterator to the next element in container if no specific argument are there.

If there is a position given in arguments it will give iterator to that position.

i = next(i) // default gives iterator to next element.

i = next(i,3) // gives iterator to 3rd element ahead.

prev() function gives iterator to the previous element in default argument and gives iterator to given numbered element as written in argument.

```
i = prev(i);
i = prev(i,2);
```

advance() function not returns iterators to any element but advance/ change the position of element as given.

advance(i,3) // changes position of iterator i 3 element ahead.

advance(i,-2) // changes position 2 element back.

Types of iterators on the basis of their ability/working:-

- Input iterators:-

  It is basic iterator only used to take input. We can't use them to give output.

- Output iterators:-

  It is also a basic iterator which is only used to ~~take~~ give output.

- Forward iterators:-

  It has higher hierarchy than input & output iterators. It has all feature of both I/O iterators and some add. feature. As name suggest it can only move in forward direction also one at a time.

• bidirectional iterators :-

It has all features of previous 3 iterators and some add. features. As name suggest it can move in forward or backword both direction.

• Random - Access iterators :-

It is most powerful iterators. They can move to any random position in containers. They are similar/equal to pointers.

List of iterator properties :-

iterator	access	Read	write	iterate	Compare
input	→	= *i		++	==, !, =
output			*i =	++	
forward	→	= *i	*i =	++	==, !, =
bi. dire.		= *i	*i =	++, --	==, !=
Random Access	→, []	= *i	*i =	++, --, +=, +, -,	==, !=, <, >, <=, >=

Date
Page

iterators associated/linked with Containers.

Types	Containers	iterators
Simple	forward list (Singly linked list)	forward
	list (doubly linked list)	bidirectional
	vector	Random
Associative	set	bidirectional
	map	"
	multimap	"
	multiset	"
	unordered_set	forward
	unordered_map	"
adapters	queue	Don't have it.
	stack	"
	priority_queue	"

⇒ Pairs:-

Pair is a standard templet class which can have two elements of any types.

It is useful in multiple case like storing co-ordinates of point in 2-D plane, storing Id and price of object at same place.

Also it has pre-defined functions to acess it's data, so, do not need to emplement our own.

Syntax:-

- pair <datatype1, datatype2> name

To use pair we have to include <utility> header file.

If we dont initialize a pair, by default value is zero, for char NULL

ways to initialize a pair.

- Pair<int, int> P1(10, 20);
- pair<int, int> P1 ;
  P1 = {10, 20};

ex.-

```
#include<iostream>
#include<utility>
using namespace std;

int main()
{
    pair<int, int> P1{10,20};
    pair<int, int> P2;
    pair<char, string> P3;

    P3={'c', "Anvj kumar"};

    cout<<P1.first<<" "<<P1.second<<endl;
    cout<<P2.first<<" "<<P2.second<<endl;
    cout<<P3.first<<" "<<P3.second<<endl;
    return 0;
}
```

output :-

```
10  20
0   0
c   Anvj kumar
```

first function used to access first element of pair and second function used to acess second element of pair.

Comparision operators on pairs:-

- ==

returns true is both elements are same, checks one by one.

- !=

checks first element then second, if any one of them don't match returns true.

- $P_1 > P_2$

checks first element, if one (P_2 first) is less than other (P_1 first) returns true, will not check for second element, unless first element is equal.

Same rule for <, <= and >=

ex-

$P_1 = \{1, 2\}$, $P_2 = \{3, 7\}$

$P_3 = \{3, 5\}$

$P_1 > P_2 \rightarrow 0$	$P_2 > P_3 \rightarrow 1$
$P_3 > P_1 \rightarrow 1$	

If we are sorting an vector of pairs then also sorting will be done on basis of first element.

Q. Sort one array according to other.

Lets we have two array.

$$X[] = \{20, 15, 10\}$$
$$Y[] = \{'x', 'A', 'c'\}$$

now, we have to sort array X[] and print it's corresponding value in order.

So, it's output will be.

C . A . X

Another ex. -

$$X[] = \{7, 2, 5, 10\}$$
$$Y[] = \{'c', 'f', 't', 'p'\}$$

Output f t c p

⇒ So what we will do is we makes pairs from each element of this array and then sort this array of pairs.

```
void sortpairs(int a[], int b[], int n)
{
    pair<int, char> P[n];
    for(int i=0; i<n; i++)
        P[i] = {a[i], b[i]};

    sort(P, P+n);

    for(int i=0; i<n; i++)
        cout<<P[i].second<<" ";

    return 0;
}
```

Copy one pair to other :-

To copy one pair to other, both pair should have same type.

```
pair <int, float> P1(7, 11.05);
pair <int, float> P2(P1);
                    // copied.
```

- make_pair() function :-

make_pair() is function used previously to initialize value to the pair but now, we can initialise value directly from object.

Syntax

```
pair_name = make_pair(value1, value2);

pair <int, float> P1;
P1 = make_pair( 7, 17.05);
```

- Swap() :-

Swap() function swaps values of two pair if both are of same type. It swaps first elements of both pairs and second element of both pairs to each other.

ex.-

```
pair <int, int> P1(10, 20);
pair <int, int> P2(30, 40);

P1.swap(P2);
```

now,

P_1 is {30, 40} and
P_2 is {10, 20}.

NOTE:- We can perform addition like arithmetic operation on elements of pairs of type int, float or double.

and we can concatnate value of pairs using '+' operator.

```
pair<string, int> P1("Greeks", 1);
pair<string, int> P2("quiz", 2);

cout<< P1.second + P2.second <<endl;

cout<< P1.first + P2.first;
```

Output

3
Greeksquiz