C++ Standard Template Library (STL)

It is a powerful collection of template classes and functions that provide efficient DSAs.

STL is organized into 4 main components.

① Containers :

These are template classes that manage collections of objects. They provide various ways to store and organize data.

Examples :

std::vector ⟶ Dynamic Array.

std::list ⟶ Doubly Linked List

std::deque ⟶ Double ended queue

std::set, std::multiset ⟶ Sorted collections of unique / non-unique elements.

std::map, std::multimap ⟶ Sorted collections of key-value pairs.

std::stack, std::queue, std::priority_queue ⟶ Adapter containers for specific data structures.

(2) Algorithms :

These are template functions that perform operations on ranges of elements within containers. They include functions for searching, sorting, manipulating, and transforming data.

Examples :

- `std::sort()` → Sorts elements in a range
- `std::find()` → Searches for a specific element
- `std::copy()` → Copies elements from one range to another.
- `std::for_each` → Applies a function to each element in a range.

(3) Iterators :

These are objects that act as pointers to elements within containers; providing a generalized way to access and traverse elements regardless of the container type. They connect algorithms to containers. Different types of iterators exists, such as input, output, forward, bidirectional, and random access iterators.

(4) Functors (Function Objects) :

These are objects that can be called like functions. They are often used with algorithms to customize their behavior, such as providing custom comparison logic for sorting or specific operation for `for_each()`.

The STL's design emphasizes generic programming, allowing algorithms and containers to work with various data types, without requiring separate implementations for each type. This is achieved through the use of C++ templates, making the STL a highly versatile and efficient tool for C++ developers.

(i) Vectors (Resizable Array)

- Exactly similar to Array, except the fact that Vectors are RESIZABLE, where Arrays are not.

```
#include <iostream>
#include <vector>   (or) #include <bits/stdc++.h>
using namespace std;
int main() {

    vector <int> nums = {1,3,2,5,4};

1   for (int i : nums)                    for (int i=0; i<5 ; i++)
3       cout << i << endl;        (or)        cout << nums[i] << endl;
2
5
4                                     (or) for (int i=0; i<5; i++)
                                               cout << nums.at[i] << endl;
```

```
1   cout << nums.front() << endl;
4   cout << nums.back() << endl;
```

```
    nums.push_back(56);
56  cout << nums.back() << endl;
```

```
X   nums.push_front(65);   Cannot add elements at Front.
```

```
    nums.pop_back();
4   cout << nums.back() << endl;
```

```
1
3
2
5
4   for (int i=0 ; i<nums.size() ; i++)
        cout << nums.at(i) << endl;
```

```
(False) 0   cout << nums.empty() << endl;
(True)  1
```

```
    return 0;
}
```

(ii) Lists

- Behaves like Array, but it is not an Array.
- The elements of the List cannot be accessed using index, cozy. Lists are implemented using LL.

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
int main() {
    list <int> nums = {2, 3, 4, 5};
    for (int i : nums)
        cout << i << endl;
```

Output:
2
3
4
5

```
    for (int i=0 ; i<4 ; i++)
        cout << nums[i] << endl;
```

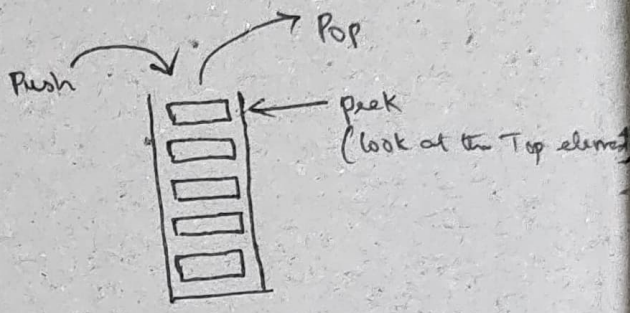
Error.
Lists cannot be indexed

```
    nums.push_front(87); ✓
    nums.push_back(56); ✓
    for (int i : nums)
        cout << i << endl;
```

Output:
87
2
3
4
5
56

```
nums.pop_front();
nums.pop_back();
for (int i : nums)          ⟶   2
    cout << i << endl;          3
                                4
                                5
return 0;
}
```

(iii) Stack (LIFO)




```
#include <bits/stdc++.h>
using namespace std;
int main() {
    stack<int> st;
    st.push(5);
    st.push(6);
```


```
    cout << st.top() << endl;   6   // peek operation
```

```
    st.pop();
```


```
    cout << st.top() << endl;   5
```

```
    st.top() = 64;
```

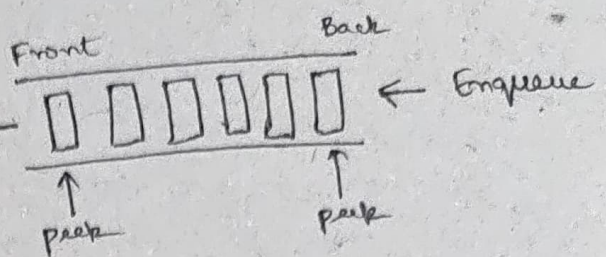

```
    // Set our Top element to 64.
    cout << st.top() << endl;   64
```

```
    cout << st.size() << endl;   1
```

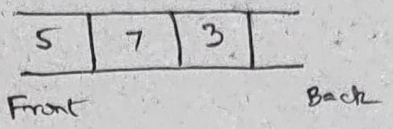
```
    cout << st.empty() << endl;   0 (False)

    return 0;
}
```

(iv) Queue (FIFO)

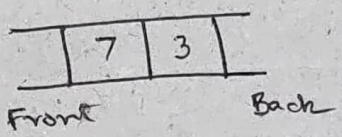



```
#include <bits/stdc++.h>
using namespace std;
int main() {
    queue <int> qu;
    qu.push(5);
    qu.push(7);
    qu.push(3);
```




```
    cout << qu.front() << endl;   5
    cout << qu.back() << endl;    3
```

```
    qu.pop();
```



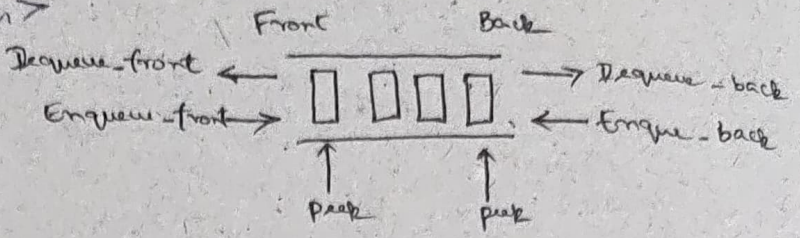

```
    cout << qu.front() << endl;   7
    cout << qu.back() << endl;    3
```

```
    cout << qu.size() << endl;    2
```

```
    cout << qu.empty() << endl;   0 (False)

    return 0;
}
```

(v) Deque (Double Ended Queue)




```
#include <bits/stdc++.h>
using namespace std;
int main() {

    deque <int> dq;
    dq.push_back(5);
    dq.push_front(7);

    cout << dq.front() << endl;   7
    cout << dq.back() << endl;    5

    ---------------------

    dq.pop_back();
    dq.pop_front();

    ---------------------

    return 0;
}
```

Front Back

7	5

Front Back

(vi) Set

- Stores only unique elements
- Automatically sorted
- Logarithmic time complexity for insertion, deletion and search
- Header file: #include<set>.
- Can Add / Remove elements, but cannot modify elements.
- No indexing.
- Set → Internally implemented using Heap.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    set<int> nums;

    nums.insert(5);
    nums.insert(7);
    nums.insert(3);

    for (int i : nums)
        cout << i << endl;

    for (int i = 0; i < 3; i++)
        cout << nums[i] << endl;

    nums.insert(3); // Duplicate insertion not allowed
    nums.erase(5);

    for (auto i : nums)
        cout << i << endl;

    cout << nums.size() << endl;

    cout << nums.empty() << endl;

    nums.clear();
    cout << nums.empty() << endl;

    return 0;
}
```

// Sort elements in descending order
set<int, greater<int>> nums;

→ 3
5
7
(Sorted order) (Ascending)

X Error.
Set cannot be indexed.

→ 3
7

2

0 (Not empty)

1 (Empty)

(vii) Map (similar to Dictionary in Python)

In C++, Map is a container that stores key-value pairs in sorted order of keys, and it ensures unique keys.

Properties:

- Each element is a pair of Key and Value.
- Keys are Unique
- Keys are automatically Sorted (ascending) by default
- Implemented as Red-Black Tree
- Average time Complexity for insertion, deletion and lookup. $O(\log n)$.
- Header: `#include<map>`

```
#include <bits/stdc++.h>
using namespace std;
int main() {

    map<string, int> age;

    age["Uvaraj"] = 36;
    age["Siva"] = 34;
    age["Anand"] = 38;

    for(auto i : age)
        cout << i << endl;
```

✗ Error

```
    for(auto i : age)
        cout << i.first << i.second << endl;
```

Anand 38
Siva 34
Uvaraj 36

```
    age["Siva"] = 35;
    for(auto i : age)
        cout << i.first << i.second << endl;
```

Anand 38
Siva 35
Uvaraj 36

```
age.insert({"Kartik", 40});
for(auto i : age)
    cout << i.first << i.second << endl;
```

Anand 38
Kartik 40
Siva 35
Uvaraj 36

```
age.erase("Uvaraj");
for(auto i : age)
    cout << i.first << i.second << endl;
```

Anand 38
Kartik 40
Siva 35

```
cout << age.size() << endl;   3
cout << age.empty() << endl;  0 → Not empty

return 0;
}
```

(viii) Unordered Map

- Stores key-value pairs using a hash table, not a balanced tree like map.

Properties:

* Unordered → No guaranteed order of keys.
* Unique keys → Each key appears only once
* Fast access → Average time complexity for insert, find and erase : O(1)
* Header : #include <unordered_map>

Same code as above, except

"unordered_map <string, int> age;"

(ix) Iterators

- Like pointers that help traverse containers.
- Used with STL containers (vector, map, etc).
- Types: begin(), end(), rbegin(), const_iterator, etc.,

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    vector<int> age = {24, 23, 21, 4, 1, 7};
✓   for (int i=0; i<age.size(); i++)
        cout << age[i] << endl;
```

→
```
24
23
21
4
1
7
```

```
    set<int> age = {24, 23, 21, 4, 1, 7};
✗   for (int i=0; i<age.size(); i++)
        cout << age[i] << endl;
```

Error. Set cannot be accessed using index.

```
✓   for (int i : age)
        cout << i << endl;
```

```
1
4
7
21
23
24
```

// Iterator

```
✓   for (auto i = age.begin(); i != age.end(); i++)
        cout << *i << endl;
```

```
1
4
7
21
23
24
```

```
✓   for (auto i = age.rbegin(); i != age.rend(); i++)
        cout << *i << endl;
```

```
24
23
21
7
4
1
```

```
    return 0;
}
```

(x) Algorithms.

- STL offers 60+ algorithms :
  sort(), reverse(), find(), count(), accumulate(), binary_search(), etc.,
- Work on ranges defined by iterators
- Designed to be generic and work with any compatible container.
- Header : #include <algorithm>.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    vector <int> age = {24, 23, 21, 4, 1, 7};
    sort(age.begin(), age.end());
    for (auto i = age.begin(); i != age.end(); i++)
        cout << *i << endl;
    cout << *find(age.begin(), age.end(), 23) << endl;
    cout << *find(age.begin(), age.end(), 6) << endl;
    return 0;
}
```

1
4
7
21
23
24

23

Garbage