**Stack**

* Stack works on LIFO (Last-in-First-Out Principle)

Declaration e.g :

stack<int> myStack;

stack<char> myStack;

Push the item to the top of the stack :

E.g.myStack.**push**(5);

myStack.**push**(‘A’);

Pop the item from the stack (no-return of pop value) :

E.g.myStack.**pop**();

Peek the item at the top of the stack (returns pop value) :

E.gmyStack.**top**();

Check Whether the Stack is empty or not :

E.g.myStack.**empty**();

* Returns true if Stack is empty else returns false

Check the size of the stack:

E.g myStack.**size**();

* Returns the number of elements present in stack

**NOTE**

* **size()** may return value of type **size\_t** that's why **type-casting** size like **(int)myStack.size()** will be a more safe choice.

**Queue**

* Queue works on FIFO (First-in-First-Out) Principle

**Linked Lists**

In C++ STL, Linked Lists can be implemented using **List** containers. There are two main types of lists: **std::list** and **std::forward\_list**.

**List**

list is a doubly-linked list that allows bidirectional traversal. It is defined in the <list> header file.

Here's an example code that demonstrates how to use an std::list:

#include <iostream>

#include <list>

int main() {

// Create an empty list

std::list<int> myList;

// Add some elements to the list

myList.push\_back(10);

myList.push\_back(20);

myList.push\_back(30);

// Insert an element at the beginning of the list

myList.push\_front(5);

// Remove the last element from the list

myList.pop\_back();

// Print the elements of the list using an iterator

std::cout << "Elements of the list: ";

for (auto it = myList.begin(); it != myList.end(); ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

The std::list container provides the following operations:

**push\_front() and push\_back()**: Adds an element to the front or back of the list, respectively.

**pop\_front() and pop\_back()**: Removes the element from the front or back of the list, respectively.

**insert():** Inserts an element at a given position in the list.

**erase()**: Removes an element from a given position in the list.

**size()**: Returns the number of elements in the list.

**empty()**: Returns whether the list is empty.

**begin() and end()**: Returns iterators to the beginning and end of the list, respectively.

**Forward List**

forward\_list is a singly-linked list that allows only forward traversal. It is defined in the <forward\_list> header file.

Here's an example code that demonstrates some of the operations that can be performed on a std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

// Create an empty forward list

std::forward\_list<int> myList;

// Add some elements to the list

myList.push\_front(30);

myList.push\_front(20);

myList.push\_front(10);

// Print the elements of the list

std::cout << "Elements of the forward list: ";

for (auto it = myList.begin(); it != myList.end(); ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

// Insert an element after the second element

auto it = myList.begin();

std::advance(it, 1); // Advance the iterator to the second element

myList.insert\_after(it, 25);

// Remove the first element

myList.pop\_front();

// Remove all elements with value 20

myList.remove(20);

// Remove consecutive duplicates

myList.unique();

// Sort the elements

myList.sort();

// Print the sorted elements of the list

std::cout << "Sorted elements of the forward list: ";

for (auto it = myList.begin(); it != myList.end(); ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Here are some of the main operations that can be performed on a std::forward\_list:

**push\_front(val)**: Inserts a new element with value val at the beginning of the list.

**pop\_front()**: Removes the first element of the list.

**insert\_after(pos, val)**: Inserts a new element with value val after the element pointed to by pos.

**erase\_after(pos)**: Removes the element after the element pointed to by pos.

**remove(val)**: Removes all elements with value val from the list.

**unique()**: Removes all consecutive duplicate elements from the list.

**sort()**: Sorts the elements of the list in ascending order.

**NOTE**

* Unlike std::list, std::forward\_list does not provide a push\_back() operation since it can only be traversed forward. However, std::forward\_list does provide a before\_begin() function which returns an iterator to the element before the first element of the list. This can be used to insert elements at the end of the list by calling insert\_after() with the before\_begin() iterator.
* Both types of lists provide similar operations, but std::list has more functionality since it supports bidirectional traversal. However, std::forward\_list is more memory-efficient since it requires only one pointer per node (compared to two pointers in std::list), and it can be faster in some cases since it has fewer memory access operations.