1. What is list?

- A `mylist` is a container in the C++ Standard Template Library (STL) that implements a **doubly-linked list**.
- It allows efficient insertion and deletion of elements from both ends and anywhere in the middle.

---

2. How to Use list?

- To use `mylist`, include the library:

```
#include <list>
```

---

3. Syntax for Declaring a List

- Declaration:

```
list<T> mylist_name;
```

    - **T**: Data type of the elements (e.g., `int`, `float`, `string`, etc.).

---

4. Why and Where to Use list?

- Use when:
    - Frequent insertion and deletion operations are required.
    - Random access is not necessary.
- Avoid when:
    - Quick access to elements by index is required.

---

5. Member Functions of list

Below are some commonly used member functions of `mylist` with their syntax, parameters, return types, and usage examples:

1. **push_back**:
    - **Syntax**: `void push_back(const T& value);`
    - **Parameter**: `value` (of type `T`) - the value to be added to the end of the mylist.
    - **Returns**: Nothing (`void`).
    - **Usage**:

```
mylist.push_back(10); // Adds 10 to the end of the list
```

2. **push_front**:
    - **Syntax**: `void push_front(const T& value);`

- **Parameter**: `value` (of type `T`) - the value to be added to the front of the mylist.
- **Returns**: Nothing (`void`).
- **Usage**:

```
mylist.push_front(5); // Adds 5 to the front of the list
```

3. **pop_back**:
    - **Syntax**: `void pop_back();`
    - **Parameters**: None.
    - **Returns**: Nothing (`void`).
    - **Usage**:

```
mylist.pop_back(); // Removes the last element
```

4. **pop_front**:
    - **Syntax**: `void pop_front();`
    - **Parameters**: None.
    - **Returns**: Nothing (`void`).
    - **Usage**:

```
mylist.pop_front(); // Removes the first element
```

5. **insert**:
    - **Syntax**: `iterator insert(iterator pos, const T& value);`
    - **Parameters**:
        - `pos` (iterator) - position before which the new element will be inserted.
        - `value` (of type `T`) - the value to be inserted.
    - **Returns**: An iterator to the inserted element (direct address).
    - **Usage**:

```
mylist.insert(it, 15); // Inserts 15 at the position pointed by
iterator it
```

6. **erase**:
    - **Syntax**: `iterator erase(iterator pos);`
    - **Parameter**: `pos` (iterator) - position of the element to be removed.
    - **Returns**: An iterator to the element following the erased element (direct address).
    - **Usage**:

```
mylist.erase(it); // Removes the element pointed by iterator it
```

7. **clear**:
    - **Syntax**: `void clear();`
    - **Parameters**: None.
    - **Returns**: Nothing (`void`).
    - **Usage**:

```
mylist.clear(); // Removes all elements from the list
```

8. **size**:
    - **Syntax**: `size_type size() const;`
    - **Parameters**: None.
    - **Returns**: The number of elements in the mylist (of type `size_type`).
    - **Usage**:
      ```
      mylist.size(); // Returns the number of elements in the list
      ```
9. **empty**:
    - **Syntax**: `bool empty() const;`
    - **Parameters**: None.
    - **Returns**: `true` if the mylist is empty, otherwise `false`.
    - **Usage**:
      ```
      mylist.empty(); // Checks if the list is empty
      ```
10. **front**:
    - **Syntax**: `T& front();`
    - **Parameters**: None.
    - **Returns**: A reference to the first element (direct element).
    - **Usage**:
      ```
      mylist.front(); // Accesses the first element of the list
      ```
11. **back**:
    - **Syntax**: `T& back();`
    - **Parameters**: None.
    - **Returns**: A reference to the last element (direct element).
    - **Usage**:
      ```
      mylist.back(); // Accesses the last element of the list
      ```

---

6. Key Differences Between list and vector

1. **Memory Allocation**:
    - `mylist`: Allocates memory for each element separately.
    - `vector`: Allocates memory in contiguous blocks.
2. **Insertion/Deletion**:
    - `mylist`: Efficient anywhere in the list.
    - `vector`: Efficient only at the end.
3. **Access Time**:
    - `mylist`: Sequential access (O(n) for random access).
    - `vector`: Direct access using indices (O(1)).
4. **Use Case**:
    - `mylist`: Use when frequent insertions and deletions are required.

- `vector`: Use when frequent random access is needed.