Using Vector in C++ Standard Template Library (STL)

1. Introduction

`std::vector` is 1 of the most fundamental sequence containers in C++ STL. It helps store elements contiguously in memory. This report will explain in details how to initialize, manipulate, and access data using this container.

2. The std::vector container

A vector is a dynamic array that can resize itself automatically when an element is inserted or deleted. It is the more widely used container in C++ for sequential data compare to regular array.

2.1/ Initialization

To use vectors in C++, we must include the `<vector>` header first:

```
#include <vector>
```

To declare a vector, use this syntax:

```
std::vector<type> <vector name>;
```

Ex:

```
std::vector<int> numb;
```

2.2/ Adding and removing elements

Vectors handle memory management dynamically. We can add elements to the end or insert them at specific positions.

`push_back(item)`: Add an item to the last end of the current vector (O(1) time complexity).

`pop_back()`: Remove the last input item (O(1) time complexity).

`insert(iterator, item)`: Insert an item at the specified position (O(n) time complexity for shifting elements and adding a new one).

Ex:

```
std::vector<int> student = {2, 5, 1, 2, 5, 0, 4, 3};

push_back(6) -> {2, 5, 1, 2, 5, 0, 4, 3, 6}

pop_back() -> {2, 5, 1, 2, 5, 0, 4}

insert(student.begin() + 3, 8) -> {2, 5, 1, 8, 2, 5, 0, 4, 3};
```

2.3/ Accessing Elements

Elements can be accessed using the index operator `[]` or `.at()`.

`v[i]`: Fast access, but return undefined behavior if i is out of bounds.

`v.at(i)`: Checks bounds and throws an exception if i is out of bound, if not then return v[i].

2.4/ Capacity and Size

`size()`: Returns the number of elements currently holding data.

`capacity()`: Returns the storage space currently allocated (usually larger than size).

`empty()`: Returns true if the current vector size is 0.

2.5/ Iteration

We can iterate through vectors using standard loops, iterators, or range-based loops.

Ex:

```
std::vector<int> vt;
```

standard loop:

```
for (int i = 0; i < vt.size(); i++) std::cout << vt[i] << ‘ ’;
```

iterators:

```
for (auto i = vt.begin(); i != vt.end(); i++) std::cout << *i << ‘ ’;
```

range-based loop:

```
for (auto i : vt) std::cout << i << ‘ ’;
```

3. Conclusion

`std::vector` serves as the default choice for dynamic sequences due to its flexibility with resizability and memory management. Understanding the use of vector ensures efficient memory usage and cleaner code.