iterators

Vector Iteration In C++, a vector is a sequence container that dynamically grows its size. It is part of the Standard Template Library (STL) and provides a dynamic array-like functionality with additional features, such as automatic resizing and memory management.

```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};

    // Using iterator to access elements
    for (std::vector<int>::iterator it = vec.begin(); it != vec.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

List

In C++, std::list is a doubly-linked list container class defined in the <list> header. It is part of the Standard Template Library (STL) and provides a way to store and manipulate a sequence of elements.

```
#include <iostream>
#include <list>

int main() {
    // Declare a list of integers
    std::list<int> myList;

    // Add elements to the list
    myList.push_back(1);
    myList.push_back(2);
    myList.push_back(3);

    // Insert an element at the beginning
    myList.push_front(0);

    // Iterate over the list and print elements
    std::cout << "List elements:";
    for (auto it = myList.begin(); it != myList.end(); ++it) {
        std::cout << ' ' << *it;
    }
    std::cout << std::endl;

    return 0;
```

```
}
```

In C++, std::list is a doubly linked list implementation that provides constant time insertion and removal of elements from anywhere in the container.

```
// Declaration: To use std::list, include the <list> header file.
#include <list>

// Initialization: std::list can be initialized using the default constructor or with initial elements.
std::list<int> myList; // Empty list
std::list<int> myList = {1, 2, 3, 4, 5}; // List with initial elements

// Insertion: Elements can be inserted at the beginning, end, or at a specific position in the list.
myList.push_back(6); // Insert at the end
myList.push_front(0); // Insert at the beginning
auto it = std::next(myList.begin(), 2); // Iterator to the third element
myList.insert(it, 10); // Insert 10 at the third position

// Accessing Elements: Elements in a std::list can be accessed using iterators. Random access is not
    supported.
for (auto it = myList.begin(); it != myList.end(); ++it) {
    std::cout << *it << " ";
}

// Removal: Elements can be removed from the list by value, range, or position.
myList.remove(3); // Remove all occurrences of 3
myList.pop_front(); // Remove the first element
myList.pop_back(); // Remove the last element
auto it = std::next(myList.begin(), 2); // Iterator to the third element
myList.erase(it); // Remove the third element

// Size and Clear: Obtain the size of the list using size() and clear the list using clear().
std::cout << "Size: " << myList.size() << std::endl;
myList.clear(); // Remove all elements

//Sorting: std::list can be sorted using the sort() method
myList.sort(); // Sort the list in ascending order

// Reverse: Reverse the elements in the list using reverse().
myList.reverse(); // Reverse the order of elements
```

```
// splice(): This function is used to transfer elements from one list to another or within the same list.
std::list<int> myList1 = {1, 2, 3};
std::list<int> myList2 = {4, 5, 6};

auto it = std::next(myList1.begin(), 2); // Iterator to the third element
myList1.splice(it, myList2); // Move all elements from myList2 to myList1 at position pointed by 'it'

//merge(): This function is used to merge two sorted lists into a single sorted list
std::list<int> myList1 = {1, 3, 5};
std::list<int> myList2 = {2, 4, 6};

myList1.merge(myList2); // Merge myList2 into myList1, both lists must be sorted
```

```
//unique(): This function is used to remove consecutive duplicate elements from the list.
std::list<int> myList = {1, 1, 2, 2, 3, 4, 4, 4, 5};
myList.unique(); // Removes consecutive duplicate elements, resulting in {1, 2, 3, 4, 5}

// resize(): This function is used to change the size of the list. If the new size is larger than the
    current size, new elements are default-inserted at the end. If the new size is smaller, elements are
    removed from the end.
std::list<int> myList = {1, 2, 3, 4, 5};
myList.resize(3); // Resize to 3 elements, resulting in {1, 2, 3}
```

list iteration

```
#include <iostream>
#include <list>

int main() {
    std::list<int> lst = {1, 2, 3, 4, 5};

    // Using iterator to access elements
    for (std::list<int>::iterator it = lst.begin(); it != lst.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Map In C++, std::map is an associative container that stores elements formed by a combination of a key and a mapped value. It is part of the Standard Template Library (STL) and is defined in the <map> header.
map iteration

```
#include <iostream>
#include <map>
#include <string>

int main() {
    // Declare a map with key as integer and value as string
    std::map<int, std::string> myMap;

    // Insert elements into the map
    myMap[1] = "One";
    myMap[2] = "Two";
    myMap[3] = "Three";

    // Access elements using the [] operator
    std::cout << "Value of key 2: " << myMap[2] << std::endl;

    // Iterate over the map
    std::cout << "Map elements:";
    for (auto it = myMap.begin(); it != myMap.end(); ++it) {
        std::cout << " (" << it->first << ": " << it->second << ")";
```

```
    }
    std::cout << std::endl;

    return 0;
}
```

```
#include <iostream>
#include <map>

int main() {
    std::map<int, std::string> myMap;
    myMap[1] = "One";
    myMap[2] = "Two";
    myMap[3] = "Three";

    // Using iterator to access elements
    for (std::map<int, std::string>::iterator it = myMap.begin(); it != myMap.end(); ++it) {
        std::cout << it->first << ": " << it->second << std::endl;
    }

    return 0;
}
```

Set In C++, std::set is an ordered associative container that contains a sorted set of unique objects. It is part of the Standard Template Library (STL) and is defined in the <set> header.

```
    #include <iostream>
#include <set>

int main() {
    // Declare a set of integers
    std::set<int> mySet;

    // Insert elements into the set
    mySet.insert(5);
    mySet.insert(3);
    mySet.insert(8);
    mySet.insert(1);
    mySet.insert(6);

    // Iterate over the set
    std::cout << "Set elements:";
    for (int num : mySet) {
        std::cout << ' ' << num;
    }
    std::cout << std::endl;

    // Check if an element exists in the set
    if (mySet.find(3) != mySet.end()) {
        std::cout << "3 is present in the set" << std::endl;
    }

    return 0;
```

```
}
```

set iteration

```
#include <iostream>
#include <set>

int main() {
    std::set<int> mySet = {3, 1, 4, 1, 5, 9};

    // Using iterator to access elements
    for (std::set<int>::iterator it = mySet.begin(); it != mySet.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

1. Write a C++ program that takes integers as input from the user until a negative integer is entered. The program should store the positive integers in a std::set and then print the elements of the set in ascending order.

```
    #include <iostream>
#include <set>

int main() {
    std::set<int> mySet;

    // Take input from the user
    int num;
    std::cout << "Enter a positive integer (or a negative integer to exit): ";
    std::cin >> num;

    // Continue taking input until a negative integer is entered
    while (num >= 0) {
        mySet.insert(num);
        std::cout << "Enter a positive integer (or a negative integer to exit): ";
        std::cin >> num;
    }

    // Print the set elements in ascending order
    std::cout << "Set elements:";
    for (int n : mySet) {
        std::cout << ' ' << n;
    }
    std::cout << std::endl;

    return 0;
}
```

Write a C++ program that takes integers as input from the user until a negative integer is entered. The program should store the positive integers in a std::list and then print the elements of the list in the reverse order.

```
    #include <iostream>
#include <list>

int main() {
    std::list<int> myList;

    // Take input from the user
    int num;
    std::cout << "Enter a positive integer (or a negative integer to exit): ";
    std::cin >> num;

    // Continue taking input until a negative integer is entered
    while (num >= 0) {
        myList.push_front(num);
        std::cout << "Enter a positive integer (or a negative integer to exit): ";
        std::cin >> num;
    }

    // Print the list elements in reverse order
    std::cout << "List elements in reverse order:";
    for (int n : myList) {
        std::cout << ' ' << n;
    }
    std::cout << std::endl;

    return 0;
}
```

You are given a list of student names along with their corresponding scores in a test. Each student can appear multiple times in the list with different scores. Your task is to write a C++ program that reads the student names and scores from the user until the user enters "end" as the student name. After reading the input, the program should store the student names and scores in a std::multimap, where the key is the student name and the value is a vector of scores corresponding to that student. Finally, the program should print the student names along with their scores in the format:

```
Ali 85
Baber 75
Ali 90
Baber 80
Zeeshan 95
Ali 88
end
sample output:
Ali: 85, 90, 88
Baber: 75, 80
Zeeshan: 95
```

```
    #include <iostream>
```

```cpp
#include <map>
#include <vector>
#include <string>

int main() {
    std::multimap<std::string, int> studentScores;
    std::string name;
    int score;

    // Read student names and scores from the user
    std::cout << "Enter student name and score (or 'end' to finish):" << std::endl;
    while (true) {
        std::cin >> name;
        if (name == "end") break;
        std::cin >> score;
        studentScores.insert({name, score});
    }

    // Print student names and scores
    for (auto it = studentScores.begin(), end = studentScores.end(); it != end; ) {
        std::cout << it->first << ": ";
        std::vector<int> scores;
        std::string currentName = it->first;  // Store the current name
        // Collect all scores for this student
        while (it != end && it->first == currentName) {
            scores.push_back(it->second);
            ++it;
        }
        // Print all scores for this student
        for (size_t i = 0; i < scores.size(); ++i) {
            std::cout << scores[i];
            if (i != scores.size() - 1) std::cout << ", ";
        }
        std::cout << std::endl;
    }

    return 0;
}
```