Accociative Containers

`map<K,V>`

The `map<K,V>` associative container in C++ store a collection of *key-value* pairs where the keys are unique and sorted in ascending order. It is generic in the sense that the type of the key `K` and the type of the value `V` can be any type.

The `map<K,V>` container has the following member functions (and more):

Function	Description	Usage example
`insert()`	Inserts a pair of the form `{key,value}` into the map. The key must be unique. Ignore if key is already present in the map.	`erp.insert({"Ali",12345})`
`operator``[]`	Returns a reference to the value associated with the given key. If the key is not present, it inserts a pair `{key,x}` where `x` is default value of type `V` and returns a reference to the value `x`.	`cout << erp["Ali"]` `erp["Ali"]=12543`
`find()`	Returns an iterator to the element with the given key. If key is not present in the map, it returns `end()`	**`if`**`(erp.find("Zia")==erp.end())` `cout << "Not found"`
`contains()`	Returns true if the map contains the given key (C++20)	**`if`**`(!erp.contains("Zia"))` `cout << "Not found"`
`erase()`	Removes the element with the given key	`erp.erase("Ali")`
`size()`	Returns the number of elements in the map	`erp.size()`
`empty()`	Returns true if the map is empty	`erp.empty()`
`clear()`	Removes all elements from the map	`erp.clear()`

The STL library contains an unordered variant of the `map<K,V>` container called `unordered_map<K,V>`. The `unordered_map<K,V>` container is defined in the `<unordered_map>` header file. It has the all the member functions described above.

In the following example, we use `map<string,int>` to store erp id of different students. The keys are the names of the students and the values are their erp ids. The keys are sorted in ascending order. The `map<K,V>` container is defined in the `<map>` header file.

```
#include <iostream>
#include <string>
#include <map>
using std::cout, std::endl, std::map, std::string;

int main() {
    map<string, int> erp;
    erp["Ali"] = 12345;   // insert pair ("Ali", 12345)
    erp["Ali"] = 13901;   // update value of key "Ali" to 13901
    erp["Beena"] = 12897; // insert pair ("Beena", 12897)
    erp.insert({"Beena", 14593}); // ignore, key "Beena" is already present
```

```
    erp.insert({"Chand", 13412}); // insert pair ("Chand", 13412)

    cout << erp["Ali"] << endl;   // prints 13901
    cout << erp["Beena"] << endl; // prints 12897
    cout << erp["Chand"] << endl; // prints 13412
    if(erp.find("Dua")==erp.end())
        cout << "Not present\n";
    cout << erp["Dua"] << endl;   // "Dua" is not present, insert pair ("Dua", 0), prints 0

    // Iterate using C++17
    for (auto& [key, value] : erp)
        std::cout << '[' << key << "] = " << value << "; ";

    // C++11 alternative:
    //  for (auto& n : erp)
    //      std::cout << n.first << " = " << n.second << "; ";
}
```

set<K>

The `set<K>` associative container in C++ stores a collection of unique keys that are sorted in ascending order. It is generic in the sense that the type of the key `K` can be any type.

The `set<K>` container has the following member functions (and more):

Function	Description	Usage example
`insert()`	Inserts a key into the set. The key must be unique. Ignore if key is already present in the set.	`itp.insert("Ali")`
`find()`	Returns an iterator to the element with the given key. If key is not present in the set, it returns `end()`	`if(itp.find("Zia")==itp.end()) cout << "Not found"`
`contains()`	Returns true if the set contains the given key (C++20)	`if(!itp.contains("Zia")) cout << "Not found"`
`erase()`	Removes the element with the given key	`itp.erase("Ali")`
`size()`	Returns the number of elements in the set	`itp.size()`
`empty()`	Returns true if the set is empty	`itp.empty()`
`clear()`	Removes all elements from the set	`itp.clear()`

The STL library contains an unordered variant of the `set<K>` container called `unordered_set<K>`. The `unordered_set<K>` container is defined in the `<unordered_set>` header file. It has all the member functions which are described above.

In the following example, we use `set<string>` to store names of different students. The keys are the names of the students. The keys are sorted in ascending order. The `set<K>` container is defined in the `<set>` header file.

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

```cpp
#include <set>
using std::cout, std::endl, std::set, std::string;

int main() {
    set<string> itp;
    itp.insert("Ali");   // insert key "Ali"
    itp.insert("Ali");   // ignore, key "Ali" is already present
    itp.insert("Beena"); // insert key "Beena"
    itp.insert("Chand"); // insert key "Chand"

    cout << itp.size() << endl; // prints 3
    if(itp.find("Dua")==itp.end())
        cout << "Not present\n";
    itp.erase("Ali"); // remove key "Ali"
    cout << itp.size() << endl; // prints 2

    return 0;
}
```

Reading from text file

The following program reads a text file line by line and prints the lines on the standard output. The `ifstream` class is defined in the `<fstream>` header file.

```cpp
#include <iostream>
#include <fstream>
#include <string>
using std::cout, std::endl, std::ifstream, std::string;

int main() {
    ifstream ifs("input.txt"); // assuming input.txt is in the current directory
    if(!ifs.is_open()) {
        cout << "Error opening file\n";
        return 1;
    }
    // ifs can be used like cin
    // e.g. int x; ifs >> x; // read an integer from file
    string line;
    while(getline(ifs, line)) {
        cout << line << endl;
    }
    return 0;
}
```

Lab Exercises

Exercise 1 .....

Highlighting browser hyperlinks. Browsers typically denote hyperlinks in blue, unless they've already been visited, in which case they're depicted in purple. Write a program that reads in a list of web addresses from standard input and output `blue` if it's the first time reading in that string, and `purple` otherwise.

Use a `unordered_set<string>` to store the visited web addresses.

Exercise 2 .....

Unique substrings of length L. Write a program that reads in text from standard input and calculate the number of unique substrings of length L that it contains. For example, if the input is `cgcgggcgcg` then there are 5 unique substrings of length 3: `cgc`, `cgg`, `gcg`, `ggc`, and `ggg`. (It has applications in data compression.)

Hint: Use the `string` method `substr(i, L)` to extract ith substring and insert into a `set<string>` or `unordered_set<string>`. Test it out on the first million digits of π or 10 million digits of π

Note: Digits of π are available in the files `pi-1million.txt` and `pi-10million.txt`.

Exercise 3 .....

Inverted index of a book. Write a program that reads in a text file and compiles an alphabetical index of which words appear on which lines, as in the following input. Ignore case and punctuation.

```
It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,

age 3-4
best 1
foolishness 4
it 1-4
of 1-4
the 1-4
times 1-2
was 1-4
wisdom 4
worst 2
```

Hint: create a `map<string,vector<int>>` whose key is a `string` that represents a word and whose value is a `vector<int>` that represents the list of pages on which the word appears.