Associative 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.

```cpp
#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

    return 0;
}
```

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>
#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.

```
#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) {
        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 ....

Spell checking. Write a `set<string>` client that takes as command-line argument the name of a file containing a dictionary of words, and then reads strings from standard input and prints out any string that is not in the dictionary. Use the following two files as dictionary:

- `words.txt` contains a list of 20,068 words, each on a separate line. It is available at `https://introcs.cs.princeton.edu/java/data/words.txt`.
- `wordlist.txt` contains a list of 224,714 words, each on a separate line. It is available at `https://introcs.cs.princeton.edu/java/data/wordlist.txt`.

Exercise 2 ....

Spell correction. Write an `map<string,string>` client that serves as a filter that replaces commonly misspelled words on standard input with a suggested replacement, printing the result to standard output. Use the file `misspellings.txt`, which contains many common misspellings and correction; it is available at `https://introcs.cs.princeton.edu/java/data/misspellings.csv`. For example, if the input is `hte`, your program should print `hte -> the`.

Exercise 3 ....

Password checker. Write a program that reads in a dictionary of words and a string from standard input, and checks whether it is a "good" password. Here, assume "good" means that it (i) is at least 8 characters long, (ii) is not a word in the dictionary, (iii) is not a word in the dictionary followed by a digit 0-9 (e.g., hello5), (iv) is not two words separated by a digit (e.g., hello2world)

Exercise 4 ....

Pointers and arrays. Write a function which takes two null-terminated C-like strings (char arrays) as arguments, and returns **`true`** if its first string argument is alphabetically smaller than its second string argument, **`false`** otherwise. It signature should be:

```
bool is_smaller(char *s1, char *s2);
```

You may assume that the two strings contain only lower case letters, and no blanks or other non-alphabetic characters. Test your function with a suitable `main()` program. Here is one:

```
int main() {
    char s1[] = "hello";
    char s2[] = "world";
    cout << is_smaller(s1, s2) << endl; // prints 1
    cout << is_smaller(s2, s1) << endl; // prints 0
    cout << is_smaller(s1, s1) << endl; // prints 0
    return 0;
}
```

When you are satisfied it works properly, convert the function to pointer arithmetic syntax, and check that it still behaves in the same way.

Exercise 5 ....

Pointers. Predict the output of the following program. Run the program to see if your prediction is right.

```
#include <iostream>
using std::cout;

using IntPtrType = int*;

int main()
{
    IntPtrType ptr_a = nullptr, ptr_b = nullptr;
    IntPtrType *ptr_c = nullptr;

    ptr_a = new int;
    *ptr_a = 3;
    ptr_b = ptr_a;
    cout << *ptr_a << " " << *ptr_b << "\n";

    ptr_b = new int;
    *ptr_b = 9;
    cout << *ptr_a << " " << *ptr_b << "\n";

    *ptr_b = *ptr_a;
    cout << *ptr_a << " " << *ptr_b << "\n";

    delete ptr_a;
    ptr_a = ptr_b;
    cout << *ptr_a << " " << *&*&*&*&*ptr_b << "\n";

    ptr_c = &ptr_a;
    cout << *ptr_c << " " << **ptr_c << "\n";

    delete ptr_a;
    ptr_a = nullptr;

    return 0;
}
```