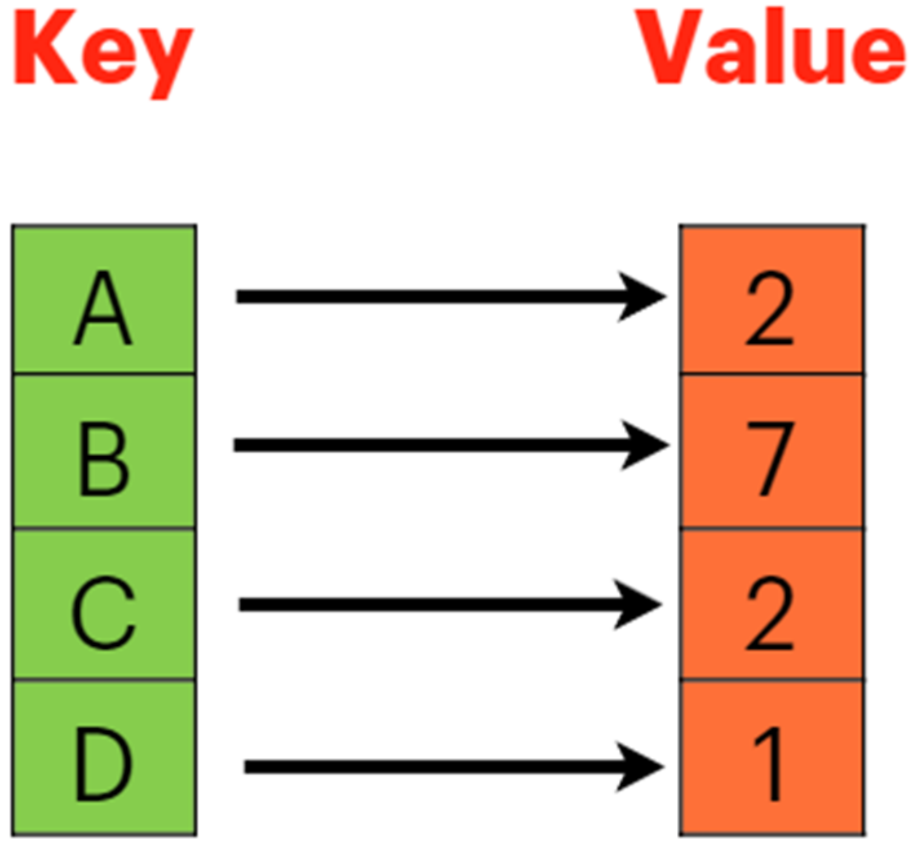
C++ Maps



By – Mohammad Imran

MAP Introduction

Maps are **associative containers** that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have the same key values.

std::map is the class template for map containers and it is defined inside the **<map>** header file.

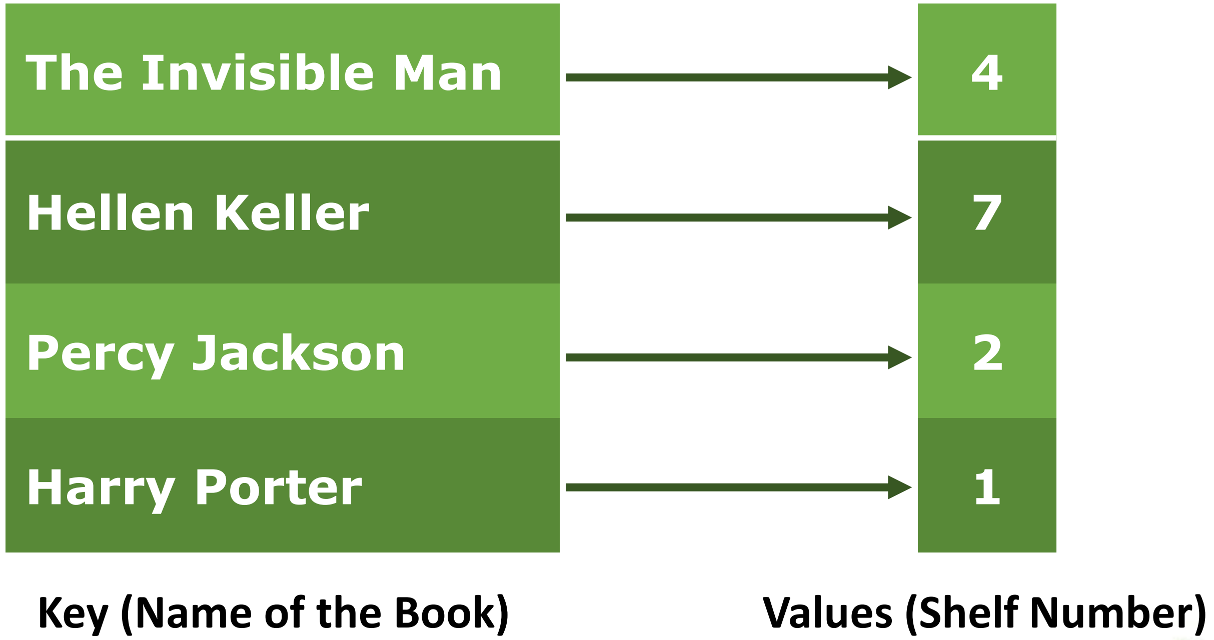
Introduction

Have you ever wondered how the mechanism of storing books in a library works?

Usually, library management systems make use of something similar to maps to search efficiently where a book must be kept. Each book is assigned a shelf number, which is stored in the computer system, for easy and fast lookup.

MAP Introduction

This is very similar to how maps work. Maps are container structures that store key-value pairs. This means that each key is unique and points to a specific value. Just like each book is unique and points to a specific shelf in the library.



By – Mohammad Imran

MAP Introduction

Not only this but maps can also be used in storing the memory addresses of variables in our code, in fact, it stores the elements in order with respect to the keys, Whenever we need to access a variable's value, we just need to look up its address on the map.

MAP Syntax

To declare a map in C++, we use the following syntax:

```
map < key_dataType, value_datatype > mapName;
```

Here,

- The key_dataType is the data type of the key.
- The value_dataType is the data type of the value.
- mapName is the name of the map.

Note: To declare the map in C++, you need to add a header file containing the template and the functions of the map.

MAP Example - 1

```cpp
#include <iostream>
#include <map>
using namespace std;
int main()
{
   map<string, int> mp;
   mp["Asia"] = 1;
   mp["Europe"] = 2;
   mp["Australia"] = 3;
   mp["Antarctica"] = 4;
   cout << "The key of Antarctica: "<< mp["Antarctica"] <<endl;
   cout << "The key of Europe:      " << mp["Europe"];
   return 0;
}
```

OUTPUT

```
The key of Antarctica: 4
The key of Europe:     2
```

By – Mohammad Imran

MAP Member Function

Some basic functions associated with std::map are:

- ✓ **begin()** – Returns an iterator to the first element in the map.
- ✓ **end()** – Returns an iterator to the theoretical element that follows the last element in the map.
- ✓ **size()** – Returns the number of elements in the map.
- ✓ **max_size()** – Returns the maximum number of elements that the map can hold.
- ✓ **empty()** – Returns whether the map is empty.
- ✓ **pair insert(keyvalue, mapvalue)** – Adds a new element to the map.

By – Mohammad Imran

MAP Member Function

- ✓ **erase(iterator position)** – Removes the element at the position pointed by the iterator.
- ✓ **erase(const g)** – Removes the key-value ‘g’ from the map.
- ✓ **clear()** – Removes all the elements from the map.

MAP | begin() and end() | Example - 2

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
   map <string, int> mp;
   mp["One"] = 1;
   mp["Two"] = 2;
   mp["Three"] = 3;
   map<string, int>::iterator it = mp.begin();
   while (it != mp.end())
   {
      cout << "Key->" << it->first<< "\t";
      cout << "Value: " << it->second << endl;
      ++it;
   }
   return 0;
}
```

OUTPUT

```
Key->One          Value: 1
Key->Three        Value: 3
Key->Two          Value: 2
```

By – Mohammad Imran

MAP | empty() and size() | Example - 3

```
#include <iostream>
#include <map>
using namespace std;
int main ()
{
   map<int, int> mp;
   if (mp.empty())
      cout << "The map is empty" << endl;
   else
      cout << "The map is not empty" << endl;
   cout << "The size of the map is: " << mp.size() << endl;
   mp[1] = 14;
   mp[2] = 45;
   mp[3] = 69;
   mp[4] = 25;
```

By – Mohammad Imran

MAP | empty() and size() | Example - 3

```
    if (mp.empty())
        cout << "The map is empty" << endl;
    else
        cout << "The map is not empty" << endl;
    cout << "The size of the map is: " << mp.size() << endl;
    cout << "The max size of the map is: " << mp.max_size() <<
    endl;
    return 0;
}
```

OUTPUT

```
The map is empty
The size of the map is: 0
The map is not empty
The size of the map is: 4
The max size of the map is: 461168601842738790
```

By – Mohammad Imran

MAP | insert() and delete() | Example - 4

```
#include <iostream>
#include <iterator>
#include <map>
using namespace std;
int main()
{
   map<int, int> quiz1;
   quiz1.insert(pair<int, int>(1, 40));
   quiz1.insert(pair<int, int>(2, 30));
   quiz1.insert(pair<int, int>(3, 60));
   quiz1.insert(pair<int, int>(4, 20));
   quiz1.insert(pair<int, int>(5, 50));
   quiz1.insert(pair<int, int>(6, 50));
```

MAP | insert() and delete() | Example - 4

```
// another way of inserting a value in a map
quiz1[7] = 10;

// printing map gquiz1
map<int, int>::iterator itr;
cout << "\nThe map quiz1 is : \n";
cout << "\tKEY\tELEMENT\n";
for (itr = quiz1.begin(); itr != quiz1.end(); ++itr)
{
    cout << '\t' << itr->first << '\t' << itr->second <<
    '\n';
}
cout << endl;
```

By – Mohammad Imran

MAP insert() and delete() Example - 4

```
// assigning the elements from quiz1 to quiz2
map<int, int> quiz2(quiz1.begin(), quiz1.end());

// print all elements of the map quiz2
cout << "\nThe map quiz2 after" << " assign from quiz1 is :
\n";
cout << "\tKEY\tELEMENT\n";
for (itr = quiz2.begin(); itr != quiz2.end(); ++itr)
{
   cout << '\t' << itr->first << '\t' << itr->second <<
   '\n';
}
cout << endl;
```

By – Mohammad Imran

MAP insert() and delete() Example - 4

```
// remove all elements up to element with key=3 in quiz2
cout << "\nquiz2 after removal of elements less than key=3 :
\n";

cout << "\tKEY\tELEMENT\n";

quiz2.erase(quiz2.begin(), quiz2.find(3));

for (itr = quiz2.begin(); itr != quiz2.end(); ++itr)
{
    cout << '\t' << itr->first << '\t' << itr->second <<
    '\n';
}
```

By – Mohammad Imran

MAP | insert() and delete() | Example - 4

```
// remove all elements with key = 4

int num;
num = quiz2.erase(4);
cout << "\nquiz2.erase(4) : ";
cout << num << " removed \n";
cout << "\tKEY\tELEMENT\n";
for (itr = quiz2.begin(); itr != quiz2.end(); ++itr)
{
    cout << '\t' << itr->first << '\t' << itr->second <<
    '\n';
}
cout << endl;
```

By – Mohammad Imran

MAP insert() and delete() Example - 4

```
    // lower bound and upper bound for map gquiz1 key = 5
    cout << "quiz1.lower_bound(5) : " << "\tKEY = ";
    cout << quiz1.lower_bound(5)->first << '\t';
    cout << "\tELEMENT = " << quiz1.lower_bound(5)->second <<
    endl;
    cout << "quiz1.upper_bound(5) : " << "\tKEY = ";
    cout << quiz1.upper_bound(5)->first << '\t';
    cout << "\tELEMENT = " << quiz1.upper_bound(5)->second <<
    endl;
    return 0;
}
```

By – Mohammad Imran

MAP | insert() and delete() | Example - 4

```
            OUTPUT

 The map quiz1 is :
         KEY       ELEMENT
         1         40
         2         30
         3         60
         4         20
         5         50
         6         50
         7         10
```

```
            OUTPUT

 The map quiz2 after assign from
quiz1 is :
         KEY       ELEMENT
         1         40
         2         30
         3         60
         4         20
         5         50
         6         50
         7         10
```

MAP insert() and delete() Example - 4

OUTPUT

```
 quiz2 after removal of
elements less than
key=3 :
        KEY       ELEMENT
        3         60
        4         20
        5         50
        6         50
        7         10
```

OUTPUT

```
 quiz2.erase(4) : 1 removed
       KEY       ELEMENT
       3         60
       5         50
       6         50
       7         10
```

OUTPUT

```
 quiz1.lower_bound(5) :  KEY = 5
ELEMENT = 50
quiz1.upper_bound(5) :  KEY = 6
ELEMENT = 50
```

MAP

Different Way To Initialize MAP

Map is an associative container available in the C++ Standard Template Library(STL) that is used to store key-value pairs. Let’s see the different ways to initialize a map in C++.

- ✓ Initialization using assignment and subscript operator
- ✓ Initialization using an initializer list
- ✓ Initialization using an array of pairs
- ✓ Initialization from another map using the map.insert() method
- ✓ Initialization from another map using the copy constructor
- ✓ Initialization through a range

By – Mohammad Imran

MAP Initialization

Using Subscript [] Operator

One of the simplest ways of initializing a map is to use the assignment(=) and the subscript([]) operators as shown below:

Syntax:

```
map <dataType, dataType> mapName;
```

Example:

```
map <string, string> myMap;
myMap[“5”] = “6”;
```

Note: [] is the subscript operator and = is the assignment operator

By – Mohammad Imran

MAP Initialization

Using Subscript [] Operator

Example - 5

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
   map <string, string> myMap;
   myMap["Ground"] = "Grass";
   myMap["Floor"] = "Cement";
   myMap["Table"] = "Wood";
   for(auto x : myMap)
   {
      cout << x.first << " -> " << x.second <<endl;
   }
   return 0;
}
```

By – Mohammad Imran

MAP Initialization

Using Initializer List

Another way of initializing a map is to use a predefined list of key-value pairs.

Syntax:

```
map<DataType, DataType>MapName = {{key1, value1},
                                  {key2, value2},
                                  {key3, value3}};
```

Example:

```
map <string, string> myMap;
myMap = {{"Ground","Grass"},{"Floor","Cement"},{"Table","Wood"}};
```

By – Mohammad Imran

MAP Initialization

Using Initializer List

Example - 6

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
   map <string, string> New_Map;
   New_Map = {{"Ground","Grass"},{"Floor","Cement"},
                                            {"Table","Wood"}};
   for(auto x: New_Map)
   {
      cout << x.first << "->" << x.second <<endl;
   }
   return 0;
}
```

By – Mohammad Imran

LIST

C++ STL

By – Mohammad Imran

LIST Introduction

Lists are one of the sequence containers available in C++ STL that store elements in a non-contiguous manner. It permits iteration in both directions. Insert and erase operations anywhere inside the sequence are completed in constant time. List containers are constructed as doubly-linked lists, which allow each of the elements, they contain to be stored in non-continuous memory locations.

LIST Introduction

std::list in C++ is a storage container. We can use the std::list to insert and remove items from any location. A doubly-linked list is used to implement the std::list. This means that list data can be retrieved in both directions. Internally, the ordering is maintained by associating each element with a connection to the element before it and a connection to the element after it.

Creating List

To create a list, we need to include the **list** header file in our program.

```
   #include <list>
```

Syntax:

```
   list <data_type> list_name;
```

- ✓ std::list - declares an STL container of type list
- ✓ <Type> - the data type of the values to be stored in the list
- ✓ list_name - a unique name given to the list
- ✓ value1, value2, ... - values to be stored in the list

LIST

Example - 9

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
   list<int> lst {12, 45, 8, 6};
   for (auto i : lst)
   {
      cout << i << ' ';
   }
   return 0;
}
```

OUTPUT

```
12 45 8 6
```

By – Mohammad Imran

LIST Why Use

The following are some of the benefits of using std::list:

- ✓ In comparison to other sequence containers such as array and vector, std::list performs better due to its non-contiguous storage.
- ✓ They perform better while inserting, moving, and removing items from any location. Insertion and deletion of elements from the list take O(1) $O(1)$ time.
- ✓ The std::list also performs better with algorithms that execute a lot of these operations.

LIST Member Functions

- ✓ **front() –** Returns the value of the first element in the list.
- ✓ **back() –** Returns the value of the last element in the list.
- ✓ **push_front() –** Adds a new element ‘g’ at the beginning of the list.
- ✓ **push_back() –** Adds a new element ‘g’ at the end of the list.
- ✓ **pop_front() –** Removes the first element of the list, and reduces the size of the list by 1.
- ✓ **pop_back() –** Removes the last element of the list, and reduces the size of the list by 1.
- ✓ **insert() –** Inserts new elements in the list before the element at a specified position.
- ✓ **size() –** Returns the number of elements in the list.
- ✓ **begin() –** begin() function returns an iterator pointing to the first element of the list.
- ✓ **end() –** end() function returns an iterator pointing to the theoretical last element which follows the last element.

By – Mohammad Imran

LIST Example - 10

```
#include <iostream>
#include <iterator>
#include <list>
using namespace std;
void print(list < int > lst)
{
   list < int > ::iterator it;
   for (it = lst.begin(); it != lst.end(); ++it)
      cout << * it << " ";
   cout << '\n';
}
int main()
{
   list < int > list1, list2;
```

By – Mohammad Imran

LIST Example - 10

```
for (int i = 0; i < 5; ++i)
{
   list1.push_back(i);
   list2.push_front(i + 5);
}
cout << "List 1 (list1) is : ";
print(list1);
cout << "\nList 2 (list2) is : ";
print(list2);
cout << "\nlist1.front() : " << list1.front();
cout << "\nlist1.back() : " << list1.back();
cout << "\nlist1.pop_front() : ";
list1.pop_front();
print(list1);
```

LIST Example - 10

```
    cout << "\nlist2.pop_back() : ";
    list2.pop_back();
    print(list2);
    cout << "\nlist1.reverse() : ";
    list1.reverse();
    print(list1);
    return 0;
}
```

OUTPUT

```
List 1 (list1) is : 0 1 2 3 4
List 2 (list2) is : 9 8 7 6 5
list1.front() : 0
list1.back() : 4
list1.pop_front() : 1 2 3 4
list2.pop_back() : 9 8 7 6
list1.reverse() : 4 3 2 1
```

By – Mohammad Imran

LIST

Some Other Functions

Function	Description
reverse()	Reverses the order of the elements.
sort()	Sorts the list elements in a particular order.
unique()	Removes consecutive duplicate elements.
empty()	Checks whether the list is empty.
clear()	Clears all the values from the list
merge()	Merges two sorted lists.

By – Mohammad Imran

LIST sort() and reverse() Example - 11

```
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;

class MyList
{
    list<int> myList;
   public:
    MyList()
    {
       myList = {5, 2, 9, 1, 7};
    }
```

By – Mohammad Imran

LIST | sort() and reverse() | Example - 11

```
void displayList()
{
   for (int n : myList)
   {
      cout << n << " ";
   }
   cout << endl;
}
void sortList()
{
   myList.sort();
   displayList();
}
```

LIST sort() and reverse() Example - 11

```
    void reverseList()
    {
        myList.reverse();
        displayList();
    }
};
int main()
{
    MyList lst;
    cout << "Original List : ";
    lst.displayList();
    cout << "Sorted List   : ";
    lst.sortList();
    cout << "Reverse List  : ";
    lst.reverseList();
    return 0;
}
```

OUTPUT

```
Original List : 5 2 9 1 7
Sorted List   : 1 2 5 7 9
Reverse List  : 9 7 5 2 1
```

By – Mohammad Imran

Coding Questions

By – Mohammad Imran

Question - 1

Write a C++ program display output as given below using MAP.

Output

```
Student[1]: Jacqueline
Student[2]: Blake
Student[3]: Denise
Student[4]: Blake
Student[5]: Aaron
```

Click here to see code

By – Mohammad Imran

Question - 2

You are tasked with developing a simple library management system using the std::map associative container in C++. The system will allow users to perform various operations like adding books, searching for books by their unique Book ID, and displaying all books. You are provided with the following requirements:

1. **Book Details:** Each book has a Book ID (integer) and a Book Name (string).
2. **Operations:**
   - **Add a book:** The user can add a book to the system by specifying its ID and Name.
   - **Search for a book:** The user can search for a book using its Book ID.
   - **Display all books:** The user can view all books stored in the system.
   - **Remove a book:** The user can remove a book from the system by specifying its Book ID.

By – Mohammad Imran

Question - 2

Task:

Your task is to enhance the existing code by implementing the **remove a book** feature and handling the following situations:

- **Scenario 1**: When the user tries to add a book with an **ID** that already exists, prompt them with a message saying the book already exists and ask whether they want to update the name.
- **Scenario 2**: When the user tries to remove a book with an **ID** that doesn't exist in the system, display a message indicating that no such book exists.
- **Scenario 3**: Ensure that if the user provides invalid input (like a non-integer Book ID), the program handles it gracefully by prompting them to enter a valid ID.

Click here to see code

By – Mohammad Imran