Data Structures & Algorithms (C++)

Map ADT (Dictionary Structure)

1. The Map ADT

Introduction

A Map Abstract Data Type (ADT) is a data structure that stores data in **key–value pairs**, where each key is unique and maps to a specific value.

It is also known as a dictionary or associative array in programming. The main idea is to quickly retrieve a value using its key instead of searching through the entire dataset.

Maps are widely used in databases, indexing systems, caching, and symbol tables in compilers.

Example

Key → Value
"Name" → "Ali"
"Age" → 21
"City" → "Kabul"

Key Properties

- Keys are unique
- Values can be duplicated
- Fast lookup using key
- Stores pairs (key, value)

2. A C++ Map Interface

Introduction

A Map interface defines the **operations that a map should support**, without specifying how they are implemented.

It provides abstraction so that different implementations (arrays, trees, hash tables) can follow the same structure.

This is important in object-oriented design for flexibility and modularity.

Typical Map Operations

- insert(key, value)
- remove(key)
- find(key)
- update(key)
- display()

C++ Interface Example

```
template <typename K, typename V>
class MapInterface {
public:
   virtual void insert(K key, V value) = 0;
   virtual void remove(K key) = 0;
   virtual V find(K key) = 0;
};
```

Explanation

- template allows generic key/value types
- virtual functions enforce implementation
- Acts as blueprint for map structures

3. The STL Map Class

Introduction

The STL map is a built-in associative container in C++ that stores key-value pairs in **sorted order based on keys**.

It is implemented internally using a **balanced binary search tree (Red-Black Tree)**.

This ensures efficient operations like insertion, deletion, and search in logarithmic time.

Features

- Automatically sorted keys
- No duplicate keys allowed
- O(log n) time complexity

- Built-in in <map> library

Syntax

```
#include <map>
map<int, string> mp;
```

Example

```
#include <iostream>
#include <map>
using namespace std;

int main() {
   map<int, string> student;

   student[1] = "Ali";
   student[2] = "Sara";
   student[3] = "Ahmed";

   for (auto it : student) {
      cout << it.first << " -> " << it.second << endl;
   }

   return 0;
}
```

Output

```
1 -> Ali
2 -> Sara
3 -> Ahmed
```

Advantages

- Very efficient (O(log n))
- Sorted data automatically
- Easy to use
- Safe and optimized

Disadvantages

- Slower than hash tables for some cases
- Requires more memory

4. Simple List-Based Map Implementation

Introduction

A list-based map is a simple implementation where key-value pairs are stored in a linear structure such as an array or linked list.

Searching is done sequentially, making it less efficient than STL map but easy to understand and implement.

It is useful for educational purposes and small datasets.

Structure

Each entry stores:

- Key
- Value

C++ Implementation

```
#include <iostream>
#include <vector>
using namespace std;

template <typename K, typename V>
class SimpleMap {
private:
   vector<pair<K, V>> data;

public:
   void insert(K key, V value) {
      for (auto &item : data) {
         if (item.first == key) {
            item.second = value;
            return;
         }
      }
      data.push_back({key, value});
   }

   V find(K key) {
      for (auto &item : data) {
         if (item.first == key)
            return item.second;
      }
```

```
        throw runtime_error("Key not found");
    }

    void remove(K key) {
        for (auto it = data.begin(); it != data.end(); it++) {
            if (it->first == key) {
                data.erase(it);
                return;
            }
        }
    }

    void display() {
        for (auto &item : data) {
            cout << item.first << " -> " << item.second << endl;
        }
    }
};
```

Main Function Example

```
int main() {
    SimpleMap<int, string> mp;

    mp.insert(1, "Ali");
    mp.insert(2, "Sara");
    mp.insert(3, "Omar");

    cout << "Map contents:\n";
    mp.display();

    cout << "\nSearch key 2: " << mp.find(2) << endl;

    mp.remove(1);

    cout << "\nAfter deletion:\n";
    mp.display();

    return 0;
}
```

Complexity

Operation	Time Complexity
Insert	O(n)
Search	O(n)
Delete	O(n)

Advantages

- Very simple to implement
- Easy to understand
- Works for small datasets

Disadvantages

- Slow for large data
- Linear search required
- Not optimized for performance

Comparison: STL Map vs List-Based Map

Feature	STL Map	List-Based Map
Structure	Balanced Tree	Linear List
Search	O(log n)	O(n)
Sorting	Automatic	None
Efficiency	High	Low
Use Case	Real applications	Learning purpose

Key Takeaways

- Map ADT stores data in **key–value pairs**
- Keys must be unique
- STL `map` uses a **balanced binary tree**
- STL map is efficient (O(log n))
- List-based map is simple but slow (O(n))
- Maps are widely used in:
    - Databases
    - Compilers (symbol tables)
    - Caching systems
    - Real-world applications