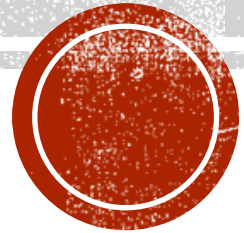
DATA STRUCTURES AND ALGORITHMS



UNORDERED MAP

- *unordered_map* is an associated container that stores elements formed by the combination of a key value and a mapped value.
- The key value is used to uniquely identify the element and the mapped value is the content associated with the key.
- Both key and value can be of any type predefined or user-defined.
- In simple terms, an *unordered_map* is like a data structure of dictionary type that stores elements in itself.
- It contains successive pairs (key, value), which allows fast retrieval of an individual element based on its unique key.
- Internally unordered_map is implemented using Hash Table, the key provided to map is hashed into indices of a hash table

CREATE AN UNORDERED MAP

In order to create an unordered map in C++, we first need to include the `unordered_map` header file.

```
#include <unordered_map>
```

Once we import this file, we can create an unordered map using the following syntax:

```
unordered_map<key_type, value_type> ump;
```

CREATE AN UNORDERED MAP

```
unordered_map<key_type, value_type> ump;
```

Here,

- `key_type` indicates the data type for the **key**
- `value_type` indicates the data type for the **value**

CREATE AN UNORDERED MAP

For example,

```
// create an unordered_map with integer key and value
unordered_map<int, int> ump_integer;

// create an unordered_map with string key and int value
unordered_map<string, int> ump_string;
```

```cpp
#include <iostream>
#include <unordered_map>
using namespace std;

int main() {

  // uniform initialization
  unordered_map<string, int> unordered_map1  {
  {"One", 1},
  {"Two", 2},
  {"Three", 3}
 };

  cout << "Key - Value" << endl;

  // loop across the unordered map
  // display the key-value pairs
  for(const auto& key_value: unordered_map1) {
    string key = key_value.first;
    int value = key_value.second;

    cout << key << " - " << value << endl;
  }

  return 0;
}
```

C++ UNORDERED MAP METHODS

In C++, the `unordered_map` class provides various methods to perform different operations on an unordered map.

Methods	Description
`insert()`	insert one or more key-value pairs
`count()`	returns **1** if key exists and **0** if not
`find()`	returns the iterator to the element with the specified key
`at()`	returns the element at the specified key
`size()`	returns the number of elements
`empty()`	returns true if the unordered map is empty
`erase()`	removes elements with specified key
`clear()`	removes all elements

INSERT IN UNORDERED MAP

```
unordered_map<string, int> unordered_map1;

// insert key-value pair {"One", 1}
unordered_map1["One"] = 1;

// insert a pair {"Two", 2}
unordered_map1.insert({"Two", 2});
```

ERASE IN UNORDERED MAP

```
unordered_map<int, string> student {
  {111, "John"},
  {132, "Mark"},
  {143, "Chris"}
};

cout << "Initial Unordered Map:\n";
display_unordered_map(student);

// remove element with key: 143
student.erase(143);
```

DISPLAY UNORDERED MAP

```
unordered_map<int, string> student {
  {111, "John"},
  {132, "Mark"},
  {143, "Chris"}
};

cout << "Initial Unordered Map:\n";
display_unordered_map(student);

// remove element with key: 143
student.erase(143);

cout << "\nFinal Unordered Map: \n";
display_unordered_map(student);
```

FIND IN UNORDERED MAP

```
unordered_map<int, string> student{
    {111, "John"},
    {132, "Mark"},
    {143, "Chris"}
};

cout << "Using find():" << endl;
cout << "Does id " << 143 << " exist? ";

// find() returns student.end() if the key is not found
if (student.find(143) != student.end()) {
  cout << "Yes";
}
else {
  cout << "No";
}
```

COUNT IN UNORDERED MAP

```
// count() returns 0 if the key doesn't exist
if (student.count(1433)) {
  cout << "Yes";
}
else {
  cout << "No";
}
```