Map

Maps are part of the C++ STL (Standard Template Library). Maps are the associative containers that store sorted key-value pair, in which each key is unique and it can be inserted or deleted but cannot be altered. Values associated with keys can be changed.

Syntax

```
template < class Key,                               // map::key_type
class T,                                  // map::mapped_type
class Compare = less<Key>,                // map::key_compare
class Alloc = allocator<pair<const Key,T> >    // map::allocator_type
> class map;
```

Parameter

key: The key data type to be stored in the map.

type: The data type of value to be stored in the map.

compare: A comparison class that takes two arguments of the same type bool and returns a value. This argument is optional and the binary predicate less<"key"> is the default value.

alloc: Type of the allocator object. This argument is optional and the default value is allocator.

Creating a map

Maps can easily be created using the following statement:

typedef pair<const Key, T> value_type;

The above form will use to create a map with key of type Key type and value of type value type. One important thing is that key of a map and corresponding values are always inserted as a pair, you cannot insert only key or just a value in a map.

Member Functions

Constructor/Destructor

Functions	Description
constructors	Construct map
destructors	Map destructor
operator=	Copy elements of the map to another map.

Iterators

Functions	Description
begin	Returns an iterator pointing to the first element in the map.
cbegin	Returns a const iterator pointing to the first element in the map.
end	Returns an iterator pointing to the past-end.
cend	Returns a constant iterator pointing to the past-end.
rbegin	Returns a reverse iterator pointing to the end.
rend	Returns a reverse iterator pointing to the beginning.
crbegin	Returns a constant reverse iterator pointing to the end.
crend	Returns a constant reverse iterator pointing to the beginning.

Capacity

Functions	Description
empty	Returns true if map is empty.
size	Returns the number of elements in the map.
max_size	Returns the maximum size of the map.

Element Access

Functions	Description
operator[]	Retrieve the element with given key.
at	Retrieve the element with given key.

Modifiers

Functions	Description
insert	Insert element in the map.
erase	Erase elements from the map.
swap	Exchange the content of the map.
clear	Delete all the elements of the map.
emplace	Construct and insert the new elements into the map.
emplace_hint	Construct and insert new elements into the map by hint.

Observers

Functions	Description
key_comp	Return a copy of key comparison object.
value_comp	Return a copy of value comparison object.

Operations

Functions	Description
find	Search for an element with given key.
count	Gets the number of elements matching with given key.
lower_bound	Returns an iterator to lower bound.
upper_bound	Returns an iterator to upper bound.
equal_range	Returns the range of elements matches with given key.

Allocator

Functions	Description
get_allocator	Returns an allocator object that is used to construct the map.

Non-Member Overloaded Functions

Functions	Description
operator==	Checks whether the two maps are equal or not.
operator!=	Checks whether the two maps are equal or not.
operator<	Checks whether the first map is less than other or not.
operator<=	Checks whether the first map is less than or equal to other or not.
operator>	Checks whether the first map is greater than other or not.
operator>=	Checks whether the first map is greater than equal to other or not.
swap()	Exchanges the element of two maps.

Example 1:

```
#include <string.h>
#include <iostream>
#include <map>
#include <utility>
using namespace std;
int main()
{
   map<int, string> Employees;
   // 1) Assignment using array index notation
   Employees[101] = "Nikita";
   Employees[105] = "John";
   Employees[103] = "Dolly";
   Employees[104] = "Deep";
   Employees[102] = "Aman";
   cout << "Employees[104]=" << Employees[104] << endl << endl;
   cout << "Map size: " << Employees.size() << endl;
   cout << endl << "Natural Order:" << endl;
   for( map<int,string>::iterator ii=Employees.begin(); ii!=Employees.end(); ++ii)
   {
       cout << (*ii).first << ": " << (*ii).second << endl;
   }
   cout << endl << "Reverse Order:" << endl;
   for( map<int,string>::reverse_iterator ii=Employees.rbegin();
ii!=Employees.rend(); ++ii)
   {
       cout << (*ii).first << ": " << (*ii).second << endl;
   }
}
```