Unordered Set

An unordered set in STL is a container that stores unique elements in no particular order. Every operation on an unordered set takes **O(1)** complexity in the average case and takes **O(n)** in the worst case.

Syntax:

```
unordered_set<object_type> variable_name;
```

Example:

```
unordered_set<int> s;
unordered_set<string> str;
```

The Whole Code —

```
// Containers--> Unordered Sets
#include <bits/stdc++.h>
using namespace std;
```

```
// Unordered Set --> Stored in unique and not store in sorted order
void explainUnorderedSet()
{
      /*
      lower_bound and upper_bound function does not works, rest all functions are same
      as above, it does not store in any particular order it has a better complexity
      than set in most cases except some when collision happens
      */

}
int main()
{
      explainUnorderedSet();
      return 0;
}

// Time Complexity --> O(1);
// Worst Case --> O(n)
```

Functions in unordered set:

insert() – to insert an element in the unordered set.

```
unordered_set<int> s;
s.insert(1);
s.insert(2);
```

begin() – return an iterator pointing to the first element in the unordered set.

```
s.begin();
```

end() – returns an iterator to the theoretical element after the last element.

```
s.end();
```

count() – it returns 1 if the element is present in the container otherwise 0.

```
unordered_set<int> s;
s.insert(1);
s.insert(2);
s.count(2); //returns true
```

clear() – deletes all the elements in unordered set.

```
s.clear();
```

find() – to search an element in the unordered set.

```
unordered_set<int> s;
s.insert(1);
s.insert(2);
if(s.find(2)!=s.end())
cout<<"true"<<endl;
```

erase() – to delete a single element or elements between a particular range.

```
s.erase();
```

size() – returns the size of the unordered set.

```
s.size();
```

empty() – to check if the unordered set is empty or not.

```
s.empty();
```

Striver’s Code

```
#include<bits/stdc++.h>

using namespace std;

int main() {
  unordered_set < int > s;
  for (int i = 1; i <= 10; i++) {
    s.insert(i);
  }

  cout << "Elements present in the unordered set: ";
  for (auto it = s.begin(); it != s.end(); it++) {
    cout << * it << " ";
  }
  cout << endl;
  int n = 2;
  if (s.find(2) != s.end())
    cout << n << " is present in unordered set" << endl;

  s.erase(s.begin());
  cout << "Elements after deleting the first element: ";
  for (auto it = s.begin(); it != s.end(); it++) {
    cout << * it << " ";
  }
  cout << endl;

  cout << "The size of the unordered set is: " << s.size() << endl;

  if (s.empty() == false)
    cout << "The unordered set is not empty " << endl;
  else
    cout << "The unordered set is empty" << endl;
  s.clear();
  cout << "Size of the unordered set after clearing all the elements: " << s.size();
}

Output:

Elements present in the unordered set: 10 9 8 7 2 1 3 4 5 6
2 is present in unordered set
Elements after deleting the first element: 9 8 7 2 1 3 4 5 6
The size of the unordered set is: 9
The unordered set is not empty
Size of the unordered set after clearing all the elements: 0
```

Other functions:

- **cbegin()** – it refers to the first element of the unordered set.

- **cend()** – it refers to the theoretical element after the last element of the unordered set.
- **bucket_size()** – gives the total number of elements present in a specific bucket in an unordered set.
- **emplace()** – to insert an element in the unordered set.
- **max_size()** – the maximum elements an unordered_set can hold.
- **max_bucket_count()** – to check the maximum number of buckets an unordered set can hold.