Unordered Set

std::unordered_set is a data structure that stores elements in no particular order and differs from **std::set** in their underlying implementation as well.

Properties

1. **Set Elements:** It is used to implement Set which is unordered in nature. It stores elements as keys. The value of such an element is key itself (which is used to identify too). keys are unique too.

2. **Time Complexity:** All operations usually on an average take **O(1)** which can in worst case go up to **O(n)**, depending upon internal Hash function. In practice though they perform very well and generally provide a constant look up operation.

3. **Unordered:** This container organizes its elements in using hash tables that allow fast access but don't have any arrangement to it i.e. they are unordered. Unlike **unordered_set**, **std::set** has an order (ascending order by default).

4. **Associative:** There is no Indexing of elements in the sets, instead they are referenced by their key and not an absolute position.

5. **Lookup:** Their elements can not be accessed using [ ] operator, instead one has to use predefined functions.

6. **Unique elements:** all Elements in an **unordered_set** is always unique, duplicate copies are removed.

7. **Internal implementation:** Internally a hash table is used, where keys are hashed into indices of hash tables so that insertion is randomized.

Note: For most questions std::set is enough.
** **std::unordered_set** is defined under header file **<unordered_set>**

Creation of unordered_set:

Syntax :

```
unordered_set<data_type> ust;
```

Example :

```
// create an unordered_set of integers
unordered_set<int> uset;

// create an unordered_set of strings
unordered_set<string> ust_string;
```

Intialisating :

We can use **insert()** function to initialize an unordered_set or we can use following :

```
// uniform initialization
     unordered_set<int> numbers {1, 100, 10, 70, 100};
```

Important Functions

1. **Lookup functions :**

- **find ()** : Gets iterator to the element. i.e. finds the element and returns its iterator. If element does not exist returns end() last iterator.
- **count ()** : returns the count of occurrences of an element in the set, 0 or 1.
- **equal_range ()** : Returns range that includes all elements equal to a given value.

---

2. **Iterators :**

- **begin ()** : Return iterator to beginning.
- **end ()** : Return iterator to end
- **cbegin ()** : Return const_iterator to beginning
- **cend ()** : Return const_iterator to end

---

3. **Capacity**

- **empty ()** : Test whether container is empty.
- **size ()** : Return container size.
- **max_size ()** : Return maximum size

---

4. **Modifiers**

- **insert()** : Insert elements.
- **erase()** : Erase elements.
- **clear()** : Clear content.
- **swap()** : Swap content.

Traversal in unordered_set:

Iterators are used to traverse in an unordered set. To make an iterator of an unordered set :

```
unordered_set<data_type>::iterator itr;
```

Iterators can be used in loops to traverse in an unordered set:

```
for (itr = duplicate.begin(); itr != duplicate.end(); itr++)
   cout << *itr << " ";
```

or

```
for(auto :& itr)
            cout << itr << endl;
```

An Example:

Here is an example to illustrate some of the functions :

```
// C++ program to demonstrate various function of
// unordered_set
#include <bits/stdc++.h>
using namespace std;

int main()
{
    // declaring set for storing string data-type
    unordered_set<string> stringSet;

    // inserting various string, same string will be stored once in set

    stringSet.insert("code");
    stringSet.insert("in");
    stringSet.insert("c++");
    stringSet.insert("is");
    stringSet.insert("fast");

    cout << "Size of Set is : " << stringSet.size() << endl;


    // find returns end iterator if key is not found,
    string key = "slow";
    if (stringSet.find(key) == stringSet.end())
        cout << key << " not found" << endl << endl;
    else
        cout << "Found " << key << endl << endl;
    // else it returns iterator to that key
    key = "fast";
    if (stringSet.find(key) == stringSet.end())
        cout << key << " not found\n";
    else
        cout << "Found " << key << endl;

    //Erasing an element
    stringSet.erase("fast");

    // now iterating over whole set and printing its content
    cout << "\nAll elements : ";
    unordered_set<string>::iterator itr;
    for (itr = stringSet.begin(); itr != stringSet.end();
         itr++)
        cout << (*itr) << ", ";
    cout << endl;


}
```

Output

```
Size of Set is : 5
slow not found

Found fast

All elements : is, c++, in, code,
```

There are many more functions in **unordered_set** which allow the programmer to create their own Hash Function and interact with hash Table and buckets.

Must Reads :

- Unordered Set vs Ordered Set
- More Functions