https://www.journaldev.com/43200/set-stl-c-plus-plus

5. Set in C++ STL - The Complete Reference - JournalDev

Web Clip

The Standard Template Library (STL) contains in-built data-structures that come in handy while implementing complex algorithms. One of these containers is a *"Set"*.

Table of Contents [hide]

Properties of a Set in C++

Properties of a Set in C++

Let us walk through some basic properties of a C++ Set before moving onto its implementation.

- **Uniqueness** – All elements inside a C++ Set are unique.
- **Sorted** – The elements inside a Set are always in a sorted manner.
- **Immutable** – Any element inside the Set cannot be changed. It can only be inserted or deleted.
- **Unindexed** – The STL Set does not support indexing.
- **Internal Implementation** – The Sets in STL are internally implemented by BSTs (Binary Search Trees).

Initializing a Set in C++

The initialization of a Set involves defining the data type for the elements to be stored and the order in which they are to be stored.

```
#include<iostream>
#include<set>
/*
Or

#include<bits/stdc++.h>
*/
using namespace std;

int main(){

    // Empty Set - Increasing Order (Default)
    set<int> s1;
    // s1 = {}

    // Empty Set - Decreasing Order
    set<int, greater<int>> s2;
    // s2 = {}

    // Set with values
    set<int, greater<int>> s3 = {6, 10, 5, 1};
    // s3 = {10, 6, 5, 1}
```

```
23
24      // Initialize Set using other set
25      set<int, greater<int>> s4(s3);
26      // s4 = {10, 6, 5, 1}
27
28      // Initializing a set from array
29      int arr[] = {10, 4, 5, 61};
30      set<int> s5(arr, arr+2);  // Only two elements
31      // s5 = {4, 10}
32
33      return 1;
34  }
```

The above code snippet explains the methods of initializing a STL Set.

Traversing a Set in C++

C++ has a concept of iterators for each specific STL data structure. Since a Set does not support indexing, we need to define iterators to fetch its elements.

```
1   #include<iostream>
2   #include<set>
3
4   using namespace std;
5
6   int main(){
7
8       // Set with values
9       set<int, greater<int>> s1 = {6, 10, 5, 1};
10
11      // Iterator for the set
12      set<int> :: iterator it;
13
14      // Print the elements of the set
15      for(it=s1.begin(); it != s1.end();it++)
16          cout<<*it<<" ";
17      cout<<endl;
18  }
```

Output:

```
10 6 5 1
```

Few things to note here are:

`'begin()'` – This function returns an iterator to the first element.

`'end()'` – This function returns an iterator past the last element.

`'*it'` – The value stored in the position iterator points to.

Adding elements to a Set

The trivial task of adding elements into a set is done by `insert()` function. The function takes in the value of the same nature as the elements in the set.

```
#include<iostream>
#include<set>

using namespace std;

int main(){

    // Set with values
    set<int, greater<int>> s1 = {6, 10, 5, 1};
    // s1 = {10, 6, 5, 1}

    // Inserting elements in the set
    s1.insert(12);
    s1.insert(20);
    s1.insert(3);

    // Iterator for the set
    set<int> :: iterator it;

    // Print the elements of the set
    for(it=s1.begin(); it != s1.end();it++)
        cout<<*it<<" ";
    cout<<endl;

}
```

Output:

```
20 12 10 6 5 3 1
```

One thing to keep in mind is that elements added are placed according to the arrangement defined at the time of the creation of the set. Therefore, the time complexity of inserting an element into a set is `O(logN)` , where **N** is the size of the set.

The size of the set can be retrieved by `'size()'` function.

Removing elements from a Set

The STL set supports the `erase()` function, that can take either the value or the iterator to the value to be removed from the set.

```cpp
#include<iostream>
#include<set>

using namespace std;

int main(){

    // Set with values
    set<int, greater<int>> s1 = {6, 10, 5, 1};
    // s1 = {10, 6, 5, 1}

    // Erasing element value = 1
    s1.erase(1);
    // s1 = {10, 6, 5}

    // Erasing the first element
    s1.erase(s1.begin());
    // s1 = {6, 5}

    // Iterator for the set
    set<int> :: iterator it;

    // Print the elements of the set
    for(it=s1.begin(); it != s1.end();it++)
        cout<<*it<<" ";
    cout<<endl;

}
```

Output:

```
6 5
```

The C++ code does not prompt an error if the element to be removed is not present in the set. In case, an invalid iterator is passed, there is a runtime error raised.

Similar to the `insert()` function, `erase()` function takes `O(logN)` time to remove an element.

Searching an element in a Set

There is a predefined function `find()` that does the job of searching an element within a set. The function returns an iterator to the value if it is present otherwise returns an iterator past the last element (`end()`).

```cpp
#include<iostream>
#include<set>

using namespace std;

int main(){

    // Set with values
    set<int, greater<int>> s1 = {6, 10, 5, 1};
    // s1 = {10, 6, 5, 1}

    // The value to be searched
    int val = 5;

    // Check if the iterator returned is not the ending of set
    if(s1.find(val) != s1.end())
        cout<<"The set contains "<<val<<endl;
    else
        cout<<"The set does not contains "<<val<<endl;

    // The value to be searched
    val = 11;

    // Check if the iterator returned is not the ending of set
    if(s1.find(val) != s1.end())
        cout<<"The set contains "<<val<<endl;
    else
        cout<<"The set does not contains "<<val<<endl;

}
```

Output:

```
The set contains 5
The set does not contains 11
```

The code is self-explanatory as it prints the message according to the presence or absence of the value.

Other commonly used Set functions

The following are a few Set functions that may come in handy while working with STL Sets.

`size()` – Returns the size of the Set

`empty()` – Returns true if the Set is empty, otherwise false.

`rbegin()` – Returns a reverse iterator pointing to the last element of the Set.

`rend()` – Returns a reverse iterator pointing before the first element of the Set.

`clear()` – Empties the entire Set.

`count(value)` – Returns 1 if the ‘value’ is present in the Set, otherwise 0.

`swap(set1, set2)` – Swaps the contents of both sets.

`emplace(value)` – Inserts the ‘value’ inside the set, if it is not present.

This winds up the tutorial on STL Sets in C++.

Conclusion

The Standard Template Library has several important containers that must be in every experienced C++ programmer's arsenal.

We hope this article on STL Sets, was easy to follow. If it was, be sure to go through other C++ articles on Journaldev. Feel free to comment below for any kind of feedback.