2020/03/11 - Code Processor

09 Tháng Ba 2020 12:04 CH

SYNOPSIS

- Ok, back to "~jplank" directories (this time, for good).
- Get used to std::map and std::set, as well as their iterator subclasses. You will use them... A LOT.

STD::MAP IN A NUTSHELL

- Associative Array pretty much.
- Unlike std::vector, you define two types here:

```
map < string , int >  m;
      ^        ^
      KEY      VALUE
```

- When inserting, use operator[] or map.insert with make_pair. With operator[], the syntax is "map[key] = value;"

(Ex.

```
map < string , int >  m;

// Insert/Access via "operator[]"
m["Clara"] = 1337;

// Insert via "map.insert" & "make_pair"
m.insert(make_pair("Clara", 1337));
```

- When accessing, use map.find with an iterator. You may also use operator[] for access, but it will add a new element into the map if it didn't exist prior. Be smart.

(Ex.
```
map<double, string> m;

//Insert some elements
m[0.123] = "buz";
m[0.456] = "plank";

//Access 0.123 via find                    (print "0.123 -> buz")
map<double, string>::iterator ii;
ii = m.find(0.123);
printf("%lg -> %s", ii->first, ii->second.c_str());

//Access 0.456 via operator[]  (print "0.456 -> plank")
printf("%lg -> %s", 0.456, m[0.456]);

//Try to access 3.14 via operator[]  ("3.14 -> BLANK")
printf("%lg -> %s", 3.14, m[3.14]);
```

- When erasing, use map.find & map.erase. It's the same as with std::list, so I won't show an example.

- std::map stores everything as a balanced BST (Binary Search Tree) that is sorted by key. If you iterate through a map and print the keys, they'll be sorted.

STD::SET IN A NUTSHELL

- It's just a std::map without the value...
  - map<key, value> m;
  - set<key> s;
- Like std::map, these are sorted by key as well. If you iterate through, you'll get the keys in sorted order.