std::set

Storing “distinct” element with fast look up

Set

- Storing distinct data of same type
  - The data type must be comparable, i.e., we can tell if a is more or less than b
- Somewhat slow insert
- Fast look up
- Fast erase
- Iterator starts from “minimum element” and goes in increasing value direction
  - Can be used to (somewhat) fast sorting

Basic

- Notice that s does not include duplicate elements
- Also see that when we iterate, member is sorted
  - This is distinct characteristic of set

```cpp
#include <iostream>
#include <set>
using namespace std;

int main() {
  set<int> s = {4,1,3,2,1,1,3,4};

  cout << "Size of s is " << s.size() << endl;

  s.insert(10);
  s.insert(5);
  s.erase(3);

  cout << "member of s: ";
  for (auto it = s.begin(); it != s.end(); it++)
    cout << *it << " ";
}
```

```
Size of s is 4
Member of s: 1 2 4 5 10
```

Somewhat slow insert, iterate but fast find

- We will see this in the detail around last part of this course
  - For now, please believe that
    - If there is N elements in the set
    - Insert take times directly proportional to log(N)
    - Each it++ or it-- take times directly proportional to log(N)
    - Each find takes times directly proportional to log(N)

Demo Comparing Vector & Set

See set-2.cpp and set-3.cpp

Set iterator

- We cannot do s.begin() + x
  - Because, going to the next element (which is the successor) in set is not as fast as vector, c++ forbids begin() + x
  - We cannot compare by > or <
- We can still use it++ or it-- to go to the next or previous (successor or predecessor) or x

```
abc,6
abcd,-3
somchai,-4
somchai,5
z,-1
z,0
z,9
-- find --
z,-1
somchai,-4
somchai,5
```

```cpp
#include <iostream>
#include <set>
using namespace std;

int main() {
  set< pair<string,int> > s = { {"somchai",5},
    {"abc",6}, {"abcd",-3}, {"somchai",-4},
    {"z",0}, {"z", -1}, {"z",9} };

  for (auto &x : s) {
    cout << x.first << "," << x.second << endl;
  }

  cout << "-- find -- " << endl;
  auto it =  s.find( {"z",-1});
  cout << (*it).first << "," << (*it).second << endl;
  it--;
  it--;
  cout << it->first << "," << it->second << endl;
  it++;
  cout << it->first << "," << it->second << endl;
```

Additional Function

- set.lower_bound
- set.upper_bound
- set.count

std::map

Association data structure with same property as set

Map

- Is very similar to Python's dict in usage
- Is internally implemented as a set with "pair" data type
  - Same properties, same limitations as set but more convenience to use as associative data structure
- Associative (mapping) between a Key Type and a Mapped Type

Basic

```cpp
#include <iostream>
#include <map>
using namespace std;

int main() {
  //map between "Key Type" string and "Mapped Type" int
  map<string,int> m;
  m["somchai"] = 10;
  m["somying"] = -5;
  cout << "Size = " << m.size() << endl;

  //accessing unseen Key create a map with default value
  cout << "m[\"xxx\"] = " << m["xxx"] << endl;
  //each element is a pair of Key Type and Mapped Type
  for (auto it = m.begin(); it != m.end();it++) {
    cout << it->first << " is mapped to " << it->second << endl;
  }

  //this will create mapping "abc" to 0 first and then increase it
  m["abc"]++;
  cout << "now size = " << m.size() << endl;
  for (auto &x : m) {
    cout << x.first << " is mapped to " << x.second << endl;
  }
}
```

```
Size = 2
m["xxx"] = 0
somchai is mapped to 10
somying is mapped to -5
xxx is mapped to 0
now size = 4
abc is mapped to 1
somchai is mapped to 10
somying is mapped to -5
xxx is mapped to 0
```

Checking if map has this key?

- Use find()

```
Key 99 is mapped to nattee
exists
```

```cpp
#include <iostream>
#include <map>
using namespace std;

int main() {
  //map between "Key Type" string and "Mapped Type" int
  map<int,string> m;
  m[1] = "somchai";
  m[99] = "nattee";

  int k = 99;
  map<int,string>::iterator it;
  if ((it = m.find(k)) != m.end()) {
    cout << "Key " << it->first << " is mapped to " << it->second << endl;
  } else {
    cout << "Key " << k << " is not exists in m." << endl;
  }

  //this is not the correct way to check if key exists why??
  if (m[k] != "") {
    cout << "exists" << endl;
  } else {
    cout << "does not exists" << endl;
  }
}
```

Requirement of std::set and std::map

- Set data type and map Key Type must be comparable
  - We must be able to compare order of two element
- Type that we can use directly
  - int, bool, float, string, double, char... and most of other numerical data type
  - Pair can also be used if both the type of first and second are comparable
    - Pair compare first then second

Practice reading c++ docs

- Both map and set has insert and erase function
- What is the return value of both function of each data structure?
  - For set<int> s, we can do s.erase(20)
  - For map<string,bool> m, can we do m.erase("Somchai") ??
- If we wish to erase element from index 3 to index 4096 in a vector
  - Is there any function from vector that we can easily use?

Problem

- Pair Sum
  - Given an array of integers, our task is to find whether there exists a pair of elements in the array such that their summation equal to X
- Input:
  - Array of integer (our main array) <-- this is a large array
  - M values of X, for each value X, we have to determine if a pair whose sum equal to X exists.
- Output:
  - For each value of X, print “YES” if we found such pair; print “NO” otherwise