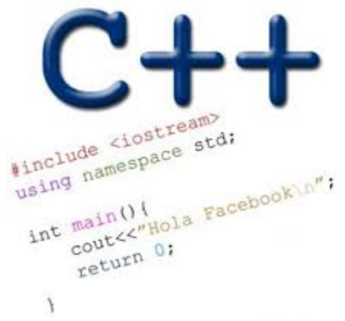
C++ STL : SET & MAP ITERATORS

Problem Solving with Computers-II

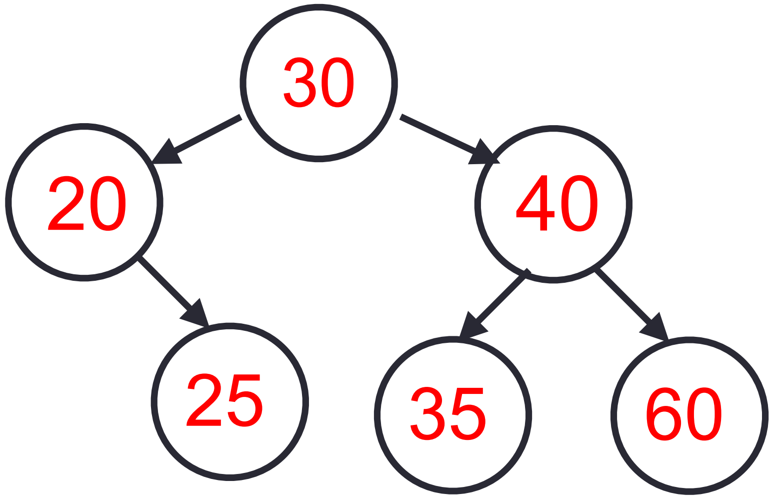


```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;

}
```

std::set: Balanced BST that stores unique keys

```
void printKeys(set<int>& s)  {
  for(auto item : s){
        cout << item <<" ";
  }
  cout<<endl;
}
```

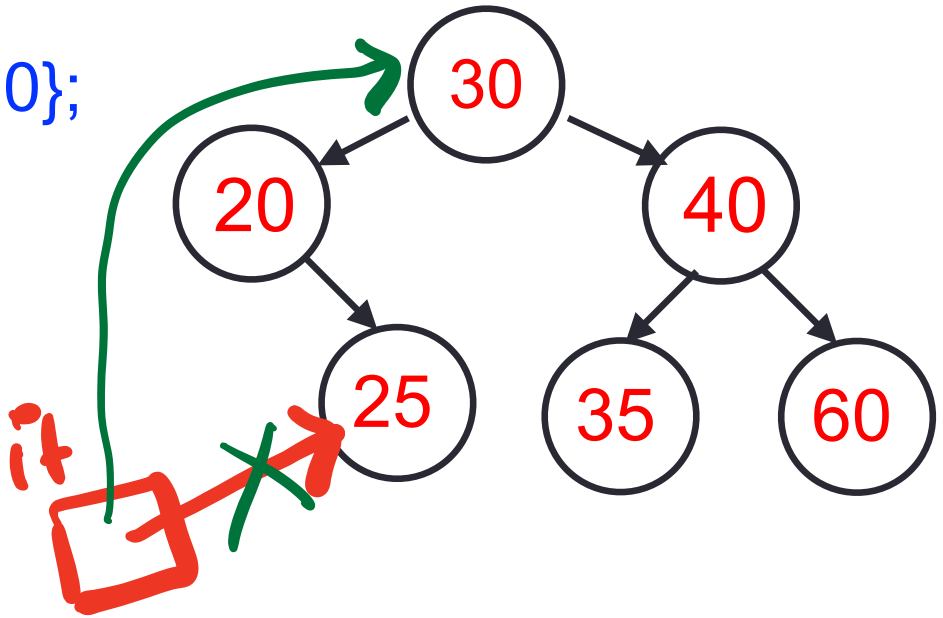
set<int> s {30, 20, 25, 40, 35, 60};
printKeys(s);



20 25 30 35 40 60

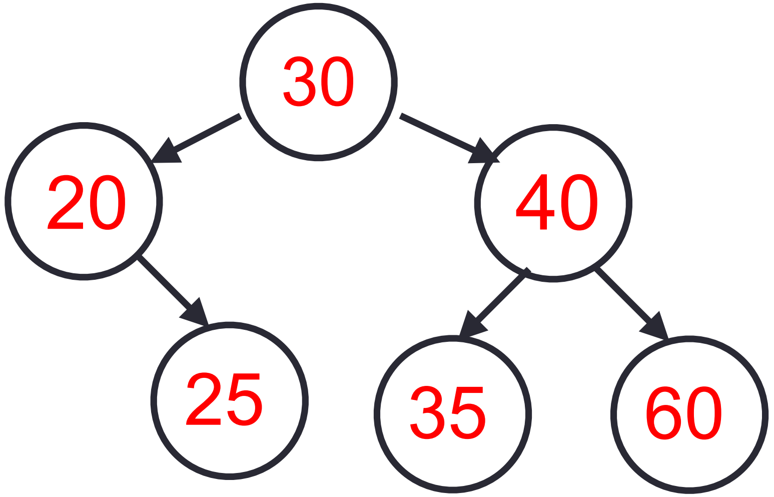
An iterator is an object that behaves like a pointer

```
set<int> s {30, 20, 25, 40, 35, 60};
auto it = s.find(25);
cout << *it;
it++;
```



An iterator is an object that behaves like a pointer

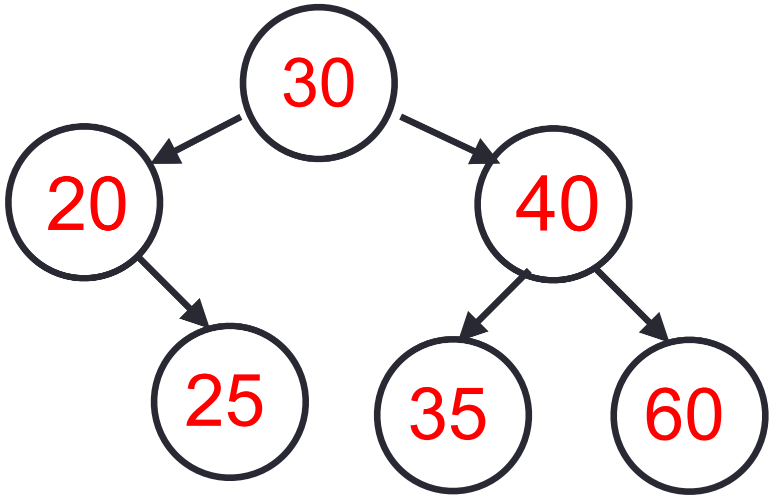
```
set<int> s {30, 20, 25, 40, 35, 60};
auto it = s.find(25);
cout << *it;
it = s.find(32);
```



But what if the value we are searching for is not there?

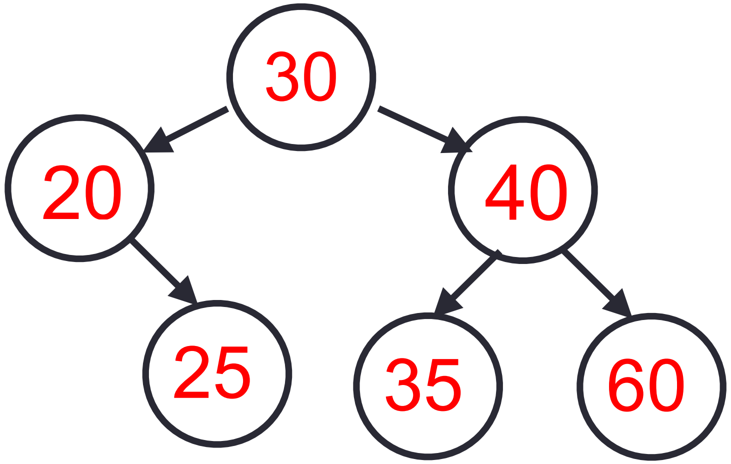
Delete 25 from the set, then insert 26

```
set<int> s {30, 20, 25, 40, 35, 60};
auto it = s.find(25);
s.erase(it);
s.insert(26);
```


$$\begin{tikzcd}[column sep=small] & & 30 \arrow[dl] \arrow[drr] & & \\ & 20 \arrow[dr] & & & 40 \arrow[dl] \arrow[dr] & \\ & & 25 & 35 & & 60 \end{tikzcd}$$

Iterating through set

```
void printKeys(set<int>& s) {
   auto it = s.begin();
   while(it!= s.end()){
        cout << *it <<" ";
        it++;
   }
}
```

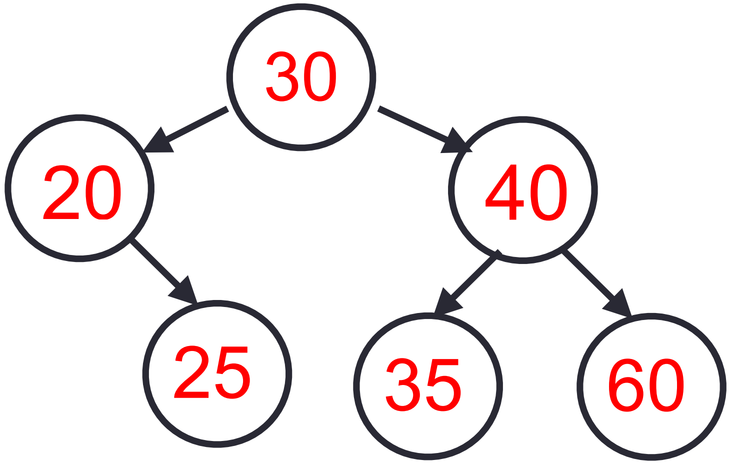


Does the above code work? Why or or Why not?

A. It works because the set class overloads the ∗ and ++ operators
B. It works because the iterator class overloads the ∗ and ++ operators
C. It doesn't work because elements of the BST are not contiguous in memory
D. It doesn't work because <fill in your reason>

Iterating through set

```
void printKeys(set<int>& s) {
    set<int>::iterator it = s.begin();
    while(it!= s.end()){
        cout << *it <<" ";
        it++;
    }
}
```



Does the above code work? Why or or Why not?

A. It works because the set class overloads the ∗ and ++ operators
B. It works because the iterator class overloads the ∗ and ++ operators
C. It doesn’t work because elements of the BST are not contiguous in memory
D. It doesn’t work because <fill in your reason>

Storing a grocery list

Which data structure would you use to store a grocery list?

A . vector of strings

B. vector of vector ✗

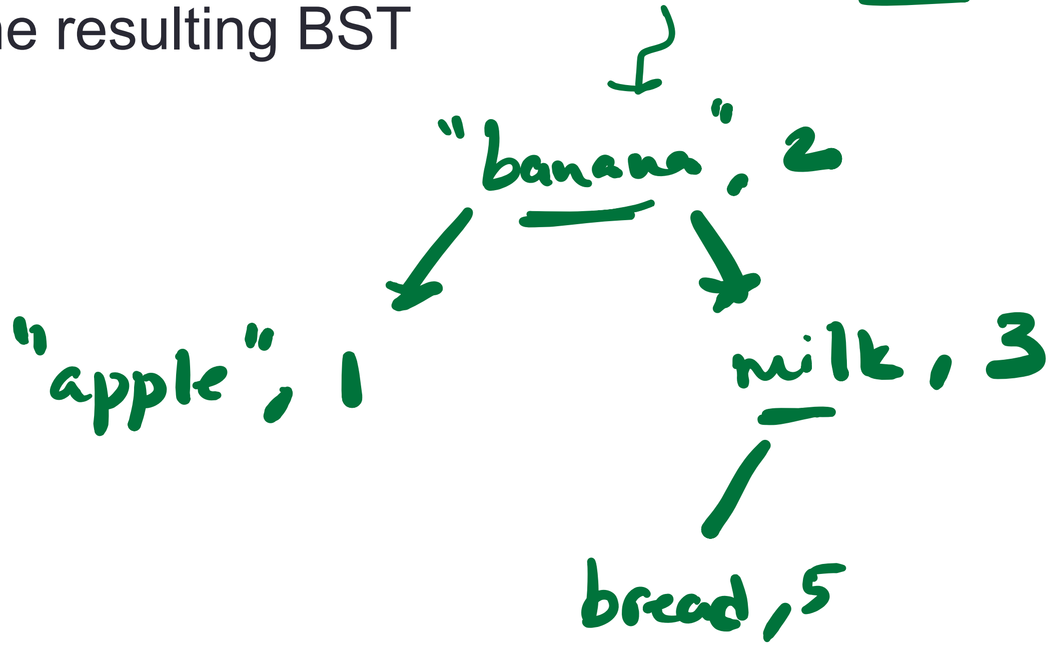
[✓ C. set containing (string, int) pair values]

D. Something else

{ {"banana", 2 }, {"apple", 1 } }

"Banana", 2
"Apple" , 1
"Milk" , 3
"Bread", 5

Insert the items in the grocery list into a BST, using the strings as keys. Draw the resulting BST



"Banana", 2
"Apple" , 1
"Milk" , 3
"Bread", 5

std::map: Balanced BST that stores (key, value) pairs

```
map<string, int> groceries;
groceries["Banana"] = 2;
groceries["Apple"] = 1;
groceries["Milk"] = 3;
groceries["Bread"] = 5;
```

insert

"Banana",	2
"Apple" ,	1
"Milk" ,	3
"Bread",	5

Other operations of map are very similar to set:

find()

erase()

Standard way of traversing the map (in order of keys) using iterators

Activity: merge similar items

key value

```
Input:  items1 = [[1,1],[4,5],[3,8]],
        items2 = [[3,1],[1,5]]
Output:  [[1,6],[3,9],[4,5]]
```

Return a 2D vector: `ret` *where* `ret[i] = [key_i, value_i]`,
with value`_i` *being the **sum of values** of all items with key* key`_i`.
The vector should be in ascending order of keys

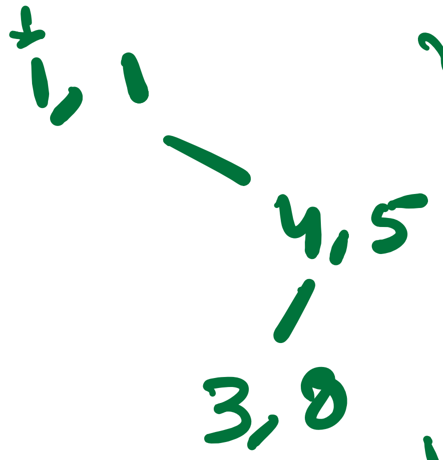
(5 mins): Brainstorm ideas on possible strategies

Activity (5 mins): Working with std::map

```
items1 = [[1,1],[4,5],[3,8]]
```

map<int, int> M;

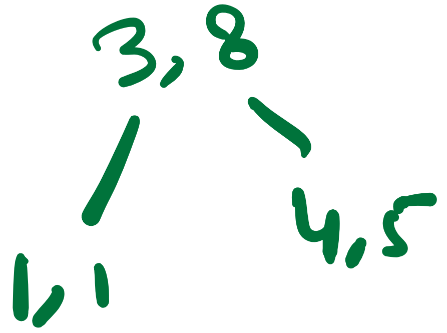
BST with regular inserts



(Not balanced)

map always maintains a balanced BST

$\longrightarrow$



balanced BST stored by std::map

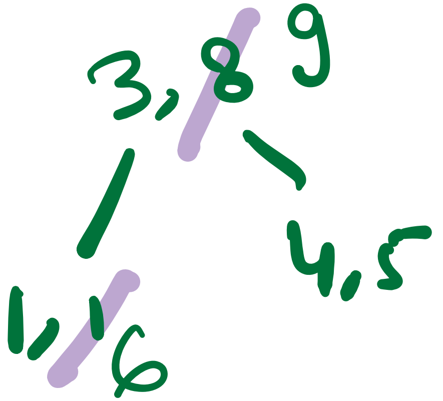
Insert the elements of items1 in a BST (std:: map)

Draw the resulting BST

Activity (5 mins): Working with std::map

```
items1 = [[1,1],[4,5],[3,8]]

items2 = [[3,1],[1,5]]
```

The final output can be obtained by reading out the elements of map in order (see code written in lecture)



BST (std:: map) after updating the tree using items2

Insert the elements of items1 in a BST (std:: map)

Insert the elements of items2 into the BST

What should we do if a key already exists? (update its value)

C++STL

- The C++ Standard Template Library is a handy set of three built-in components:
  - Containers: Data structures
  - Iterators: Standard way to search containers
  - Algorithms: These are what we ultimately use to solve problems

C++ STL container classes

```
                    array
                   vector
             forward_list
                     list
                    stack
                    queue
                      set
                      map
            unordered_set
            unordered_map
           priority_queue
multiset (non unique keys)
                    deque
                 multimap
                   bitset
```