STL SET
C++ ITERATORS
QUEUE ADT

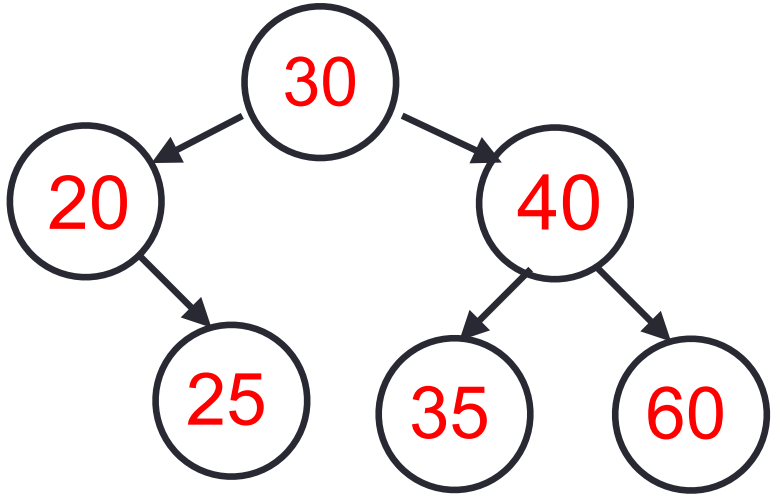
Problem Solving with Computers-II

Iterators: Standard way to iterate through containers

```
template <class T>
void printKeys(T& t)  {
  for(auto item : t){
        std::cout << item <<" ";
  }
  cout<<endl;
}
```

30	20	25	40	35	60

vector<int> v {30, 20, 25, 40, 35, 60};



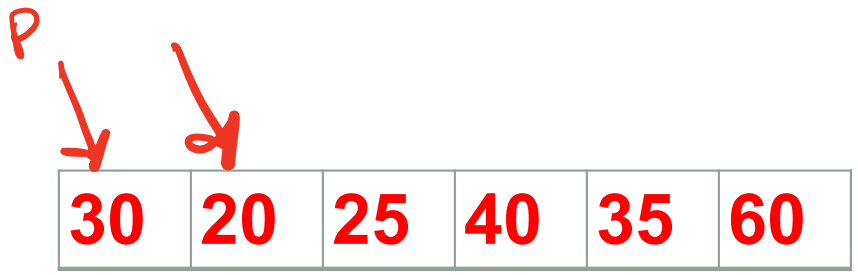
set<int> s {30, 20, 25, 40, 35, 60};

Iterating through a vector using pointers

- Let's consider how we generally use pointers to parse an array or vector

```
void printKeys(vector<int>& t) {
    int *p = &(t[0]);
    for(int i = 0; i < t.size(); i++) {
        cout << *p <<" ";
        ++p;
    }
}
```

30

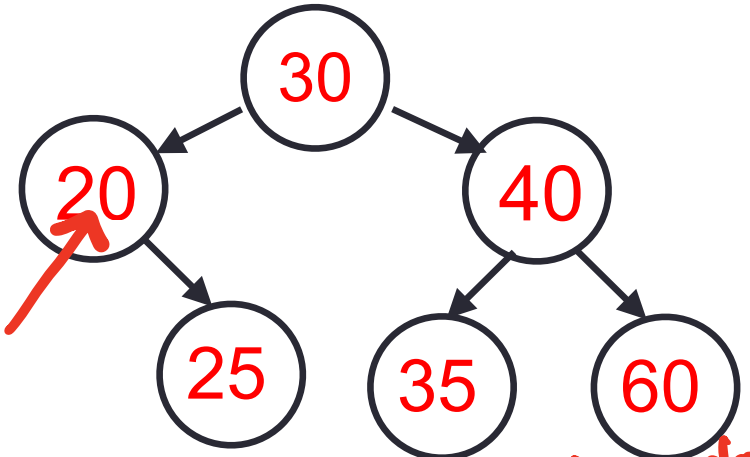


- We would like our print "algorithm" to also work with other data structures e.g. linked list or BST

Iterating through set: first try

```
void printKeys(set<int>& t) {

    int *p = &(t[0]);
    for(int i = 0; i < t.size(); i++) {
        cout << *p <<" ";
        ++p;
    }
}
```



We cannot overload operators for basic type like pointers, integers

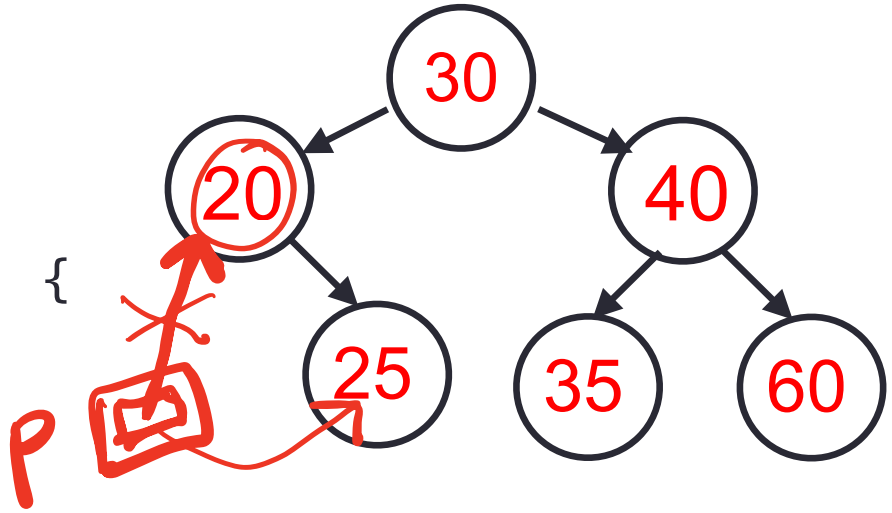
Does the above code work? Why or or Why not?

A. It works if the set class overloads the * and ++ operators
B. It works if the set class overloads the [] operator
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: first try

```
void printKeys(set<int>& t) {

    set<int>::iterator p = t.begin();
    for(int i = 0; i < t.size(); i++) {
        cout << *p <<" ";
        ++p;
    }
}
```



The variable p is an iterator (a class that stores a simple pointer to a BST node)

A. It works because ~~**set**~~ iterator class overloads the * and ++ operators

Allowing a standard way to iterate through the elements of set (in order)

Iterating through set using the set<T>::iterator

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

C++ shorthand: auto

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

Finally: unveiling the range based for-loop

```
template <class T>
void printKeys(T& t){
    for (auto item : t){
        cout << item << " ";
    }
    cout << endl;
}
```

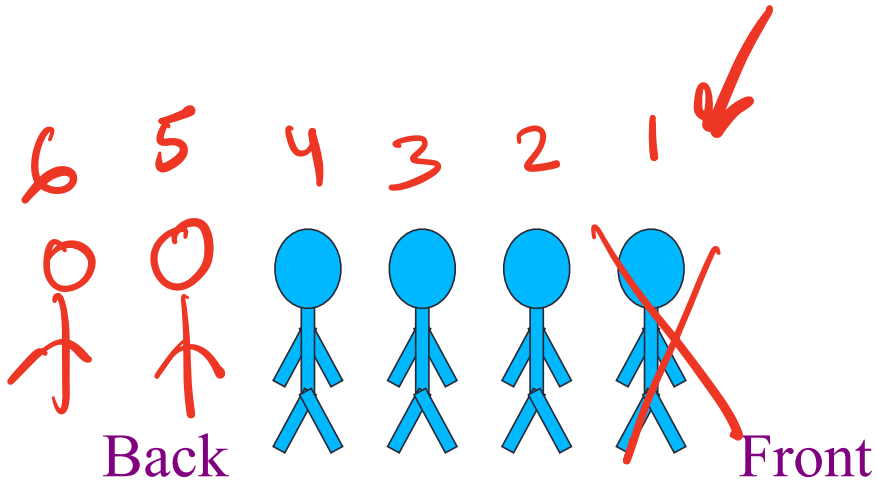
The range-based for loop is just a shorthand for code that uses iterators.

Activity (2 min) Write the expanded version of the printKeys() function using iterators

Note that not all containers have iterators. For example the same code would not work with stack, queue, or priority_queue

Queue

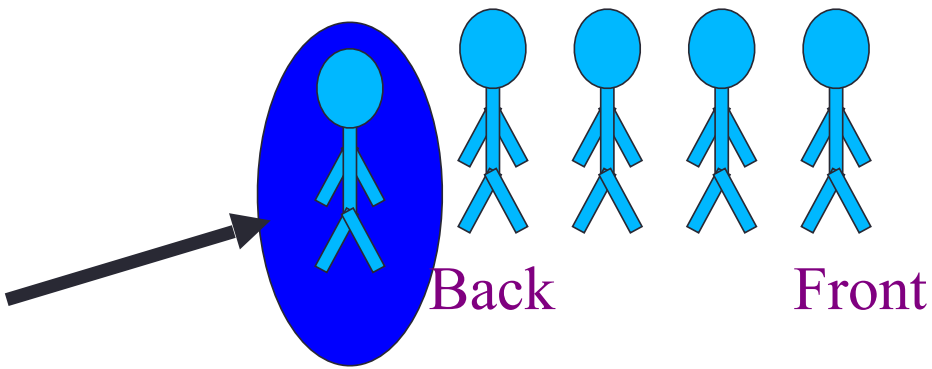
- A queue is like a queue of people waiting to be serviced
- The queue has a **front** and a **back**.



front() //1
back() //5
push(6)
pop() //
delete 1

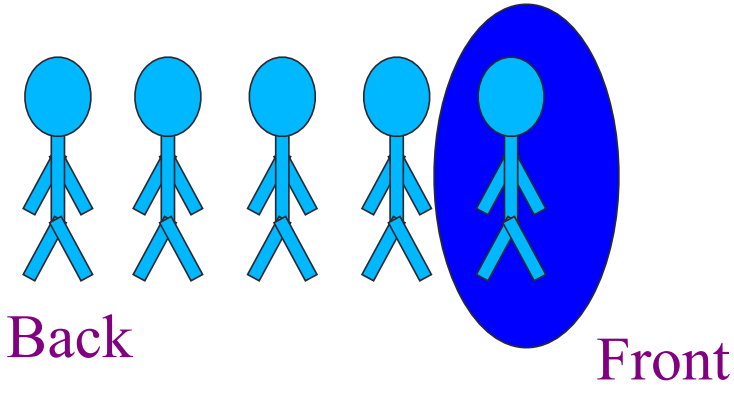
Queue Operations: push, pop, front, back

New people must enter the queue at the back. The C++ queue class calls this a **push** operation.



Queue Operations: push, pop, front, back

- When an item is taken from the queue, it always comes from the front. The C++ queue calls this a **pop**



Queue class

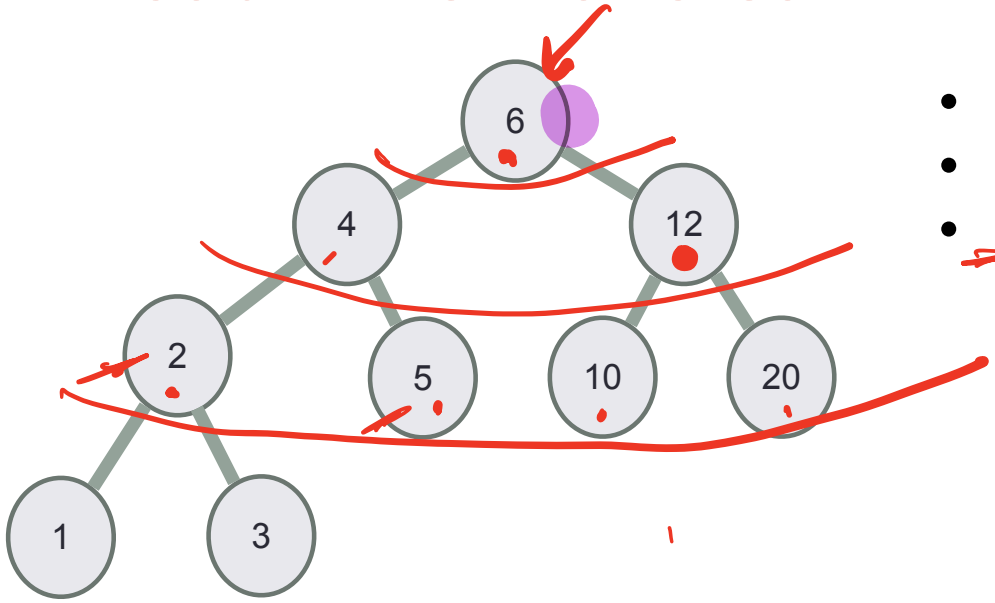
- The C++ standard template library has a queue template class.
- The template parameter is the type of the items that can be put in the queue.

```
template <class Item>
class queue<Item>
{
public:
   queue(  );
   void push(const Item& entry);
   void pop(  );
   bool empty(  ) const;
   Item front(  ) const;
   Item back(  ) const;

};
```

Breadth first traversal

Preorder: 6 4 2 1 3 5 12 10 20



- Create an empty Queue.
- Start from the root, insert the root into the Queue.
- Now while Queue is not empty,
  - Extract the node from the Queue and insert all its children into the Queue.
  - Print the extracted node.

Desired output:

Output from tracing the algo

6 4 12 2 5 10 20 1 3

6 4 12 2 5 10 20 1 3

queue

20 10 5 ~~2~~ ~~12~~ ~~4~~ ~~6~~

Reminder: Please fill course and TA mid quarter evaluations