Final exam: 03/21 (Tuesday) noon - 3p

- All information posted on the exam page:
  - https://ucsb-cs24.github.io/w23/exam/e02/
- Paper pencil, closed-book, closed-notes
- Location: NH 1006 or Phelps 3526, seating will be assigned

ITERATORS
QUEUES

Problem Solving with Computers-II

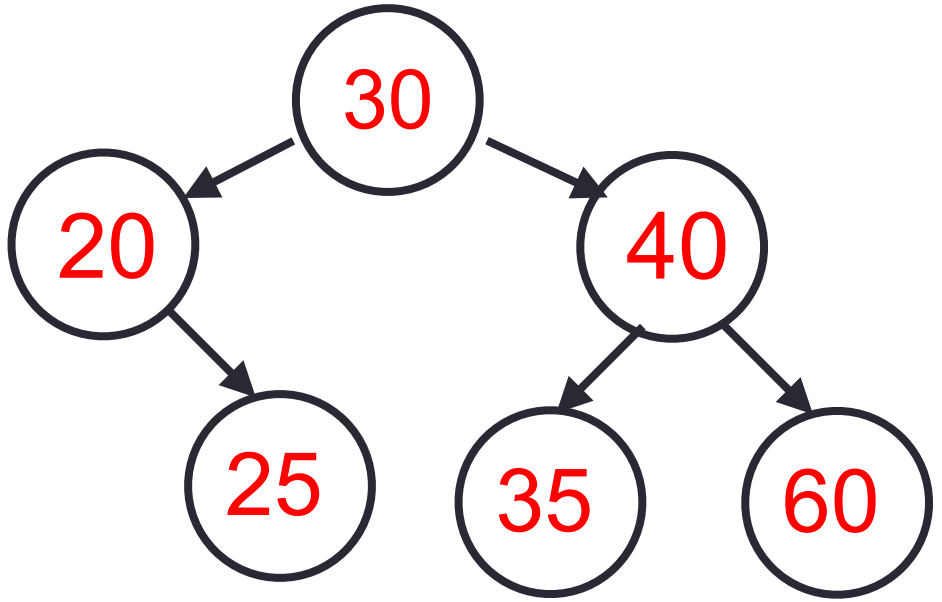
Make a copy of the handout for today's lecture:
https://bit.ly/cs24-lect16-handout

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	20	25	40	35	60

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

Iterating through set: first try

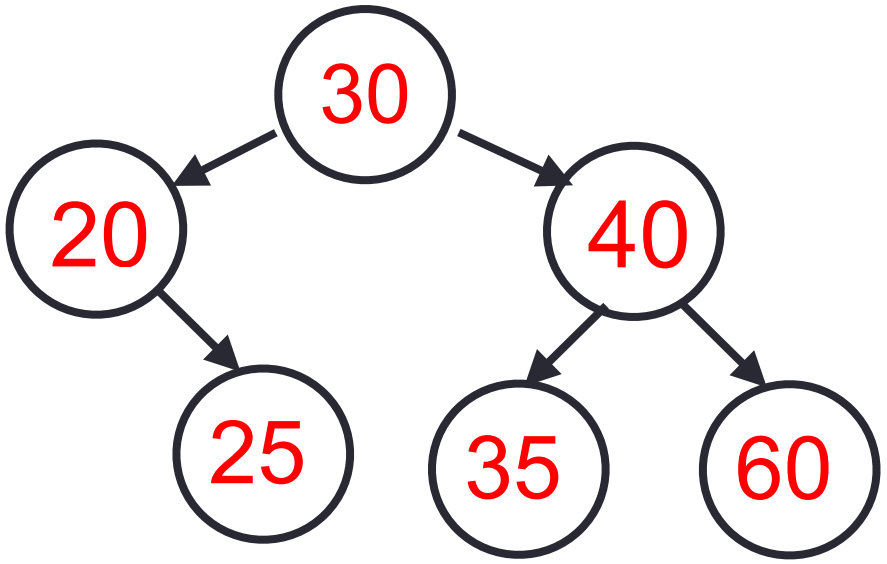
```
void printKeys(set<int>& t) {

    ____________________________// Declare p
    for(int i = 0; i < t.size(); i++) {
        cout << *p <<" ";
        ++p;
    }
}
```

How should **p** be declared and initialized**?**

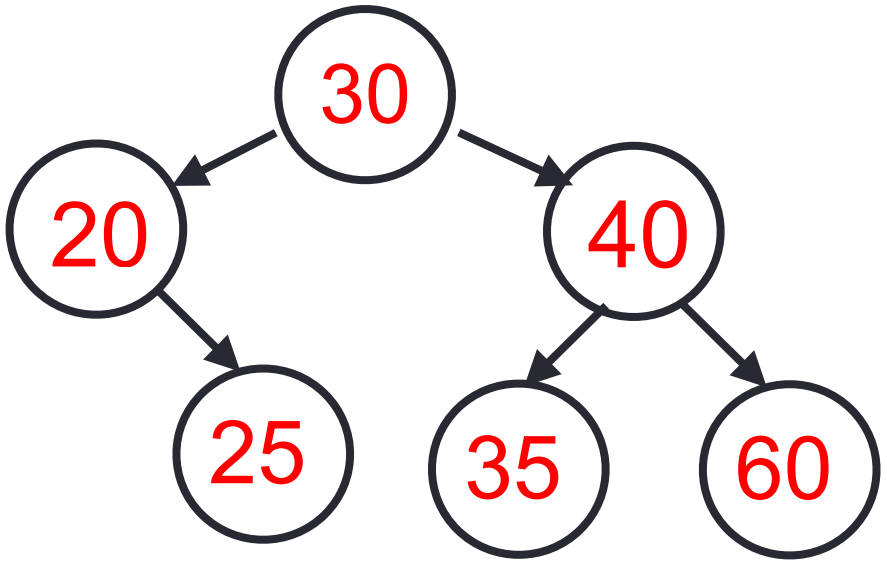
We need ***p** to return an int and

++p to call the successor function of set



What is an iterator?

```
set<int> s{30, 20, 25, 40, 35, 60};
set<int>::iterator it;
_______ =  s.find(25);
```

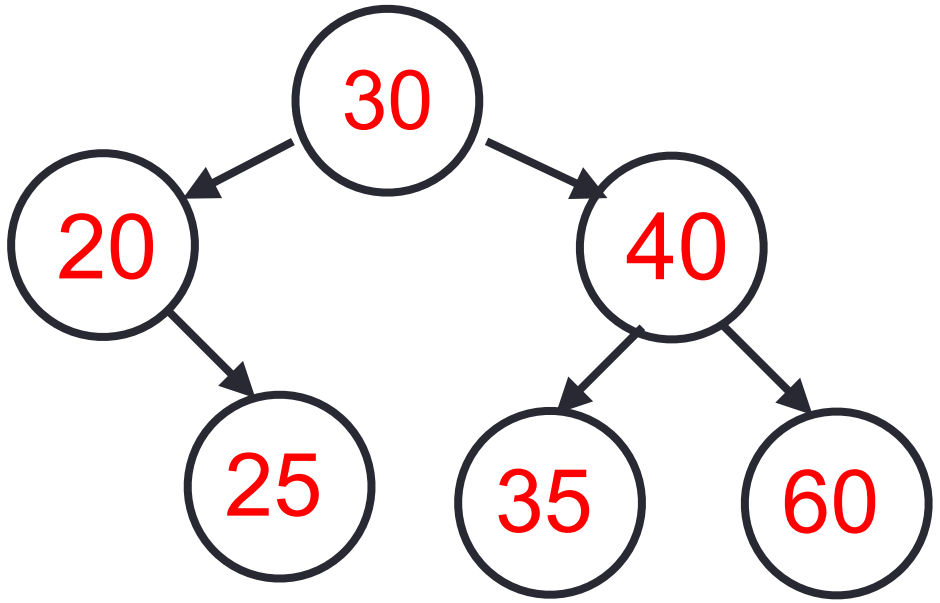


Iterators behave like **pointers** but in fact they are a class (ADT)

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

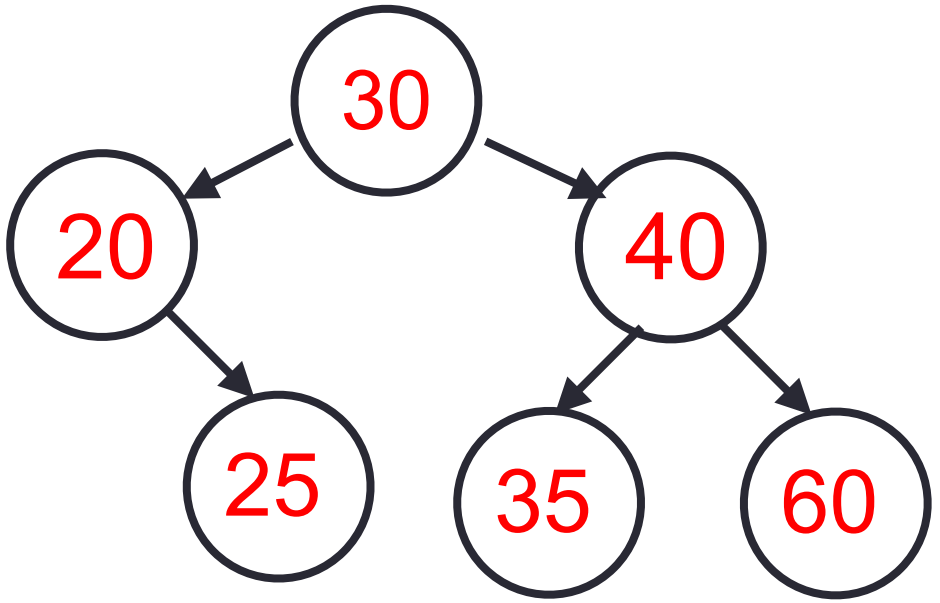
Which operators that must be overloaded for set<int>::iterator?

A. *
B. ++
C. <<
D. All of the above
E. Only A and B



How can we initialize an iterator?

```
set<int> s{30, 20, 25, 40, 35, 60};
set<int>::iterator it = s.begin();
set<int>::iterator en = s.end();
```



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;
}
```

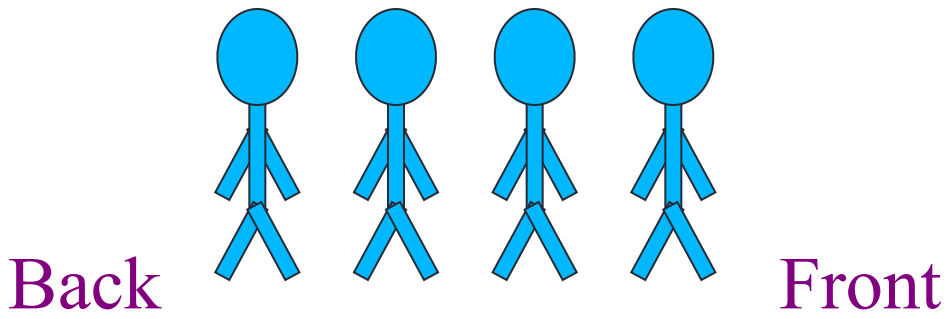
The range-based for loop is just a shorthand for code that uses iterators.

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

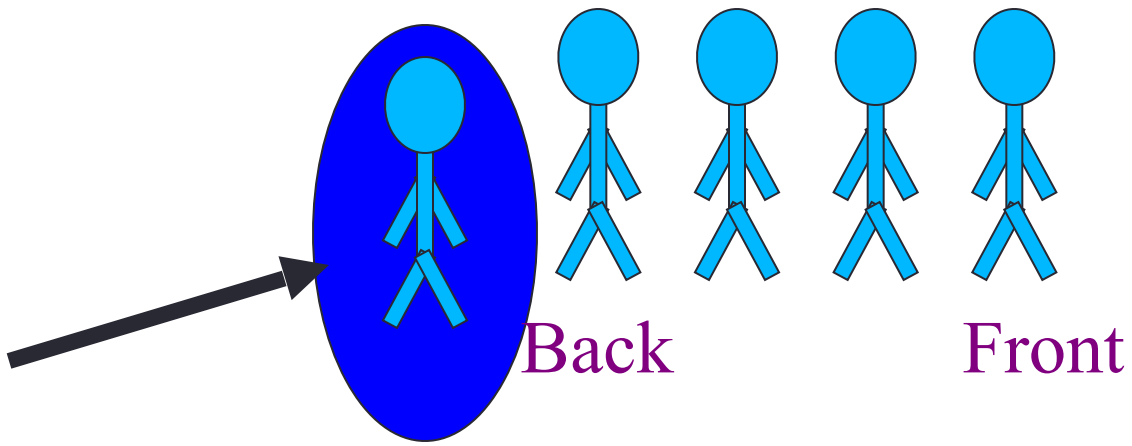
Queue

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



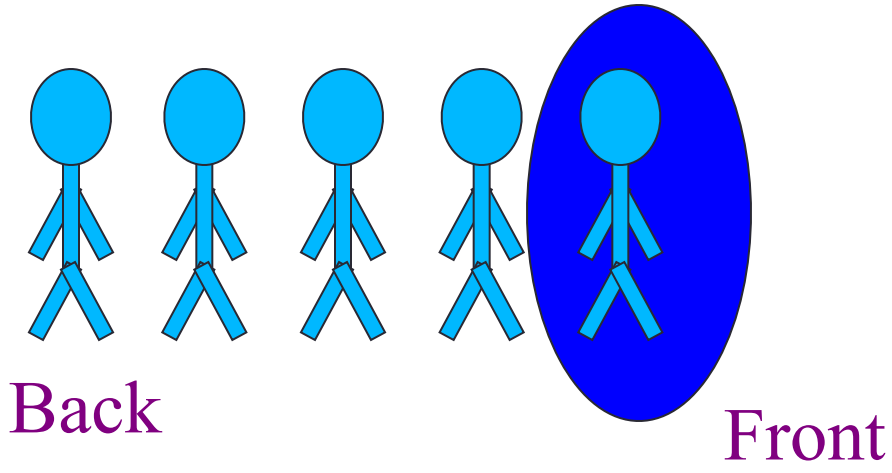
Queue Operations: push, pop, front, back

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**



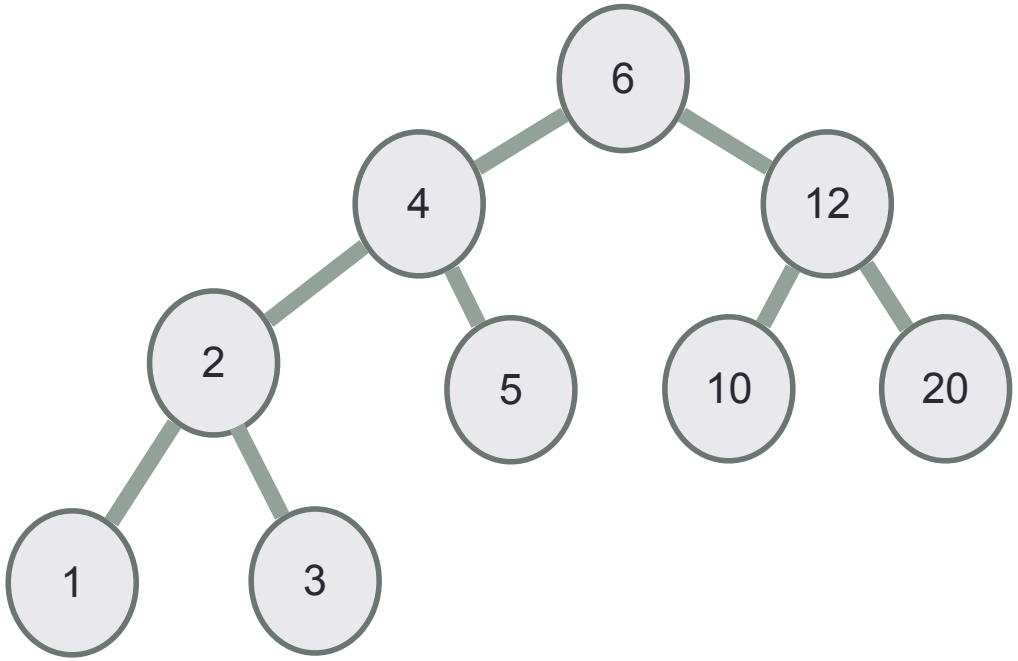
Queue class

- 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



- 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.

std::priority_queue template arguments

```
template <
      class T,
      class Container= vector<T>,
      class Compare = less <T>
      > class priority_queue;
```

The template for priority_queue takes 3 arguments:

1. Type elements contained in the queue.
2. Container class used as the internal store for the priority_queue, the default is **vector<T>**
3. Class that provides priority comparisons, the default is **less**

Comparison class: A class for comparing objects

```
template <class T>
class myCompare{
        bool operator()(T& a, T& b) const {
                return a > b;
        }
};
```

```
int main(){
    myCompare<int> cmp;
    cout<<cmp(20, 10)<<endl;
}
```

If cmp(x, y) returns true, priority queue will interpret this as:

x has _____ priority than y

Which element will be at the top of such a priority queue?

std::priority_queue template arguments

```
//Template parameters for a max-heap
priority_queue<int, vector<int>, std::less<int>> pq;

//Template parameters for a min-heap
priority_queue<int, vector<int>, std::greater<int>> pq;
```

Reminder: Please fill course and TA evaluations

If more than 70% of the class fills the course evaluations by Wed, we will have a raffle on the last class where you can win Amazon gift cards!