ITERATORS:
AN ADT SPECIALIZED FOR TRAVERSAL

Problem Solving with Computers-II

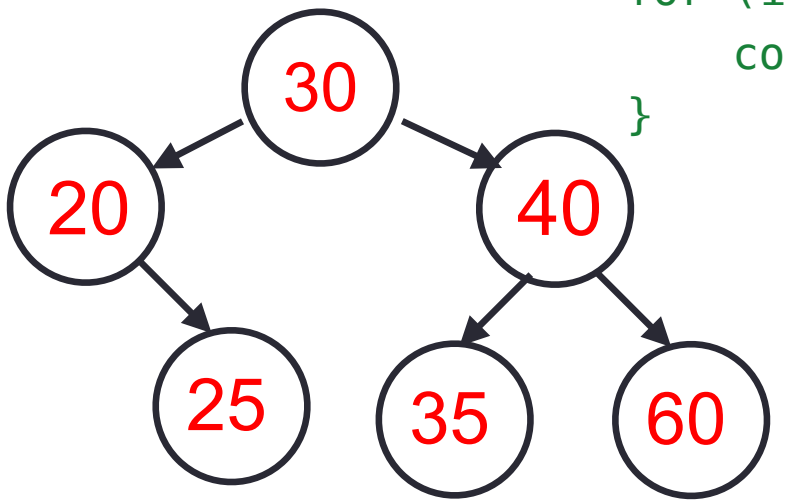
Recursive vs. Iterative traversal of a BST

Recursive in order: `printInorder`

```
void bst::printInorder(Node *r) const{
    if (!r) return;
    printInorder(r->left);
    cout << r->data << " ";
    printInorder(r->right);
}
```

Iterative: offered by `std::set`

```
std::set<int> s =
{30, 20, 25, 40,35, 60};
for (int x : s) {
    cout << x << " ";
}
```

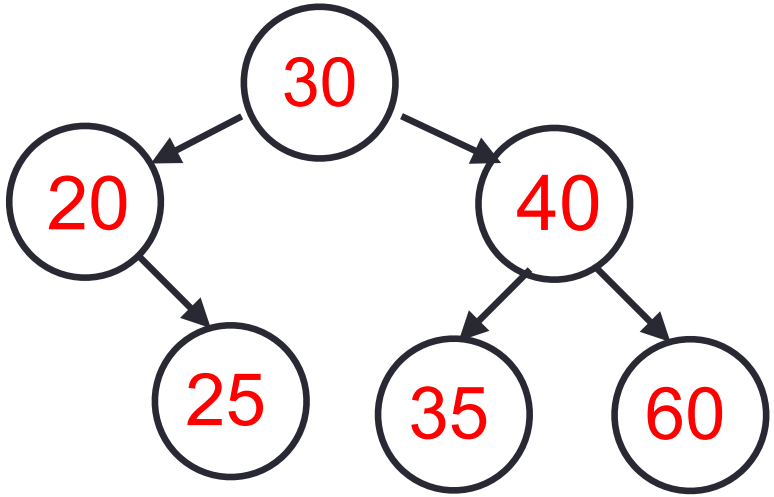


Why doesn't std::set have a printInorder() function?

Our goal: Implement one-at-a-time navigation for custom BST

What you write…

```
std::set<int> s = {. . .}
for (int x : s) {
     cout << x << " ";
}
```



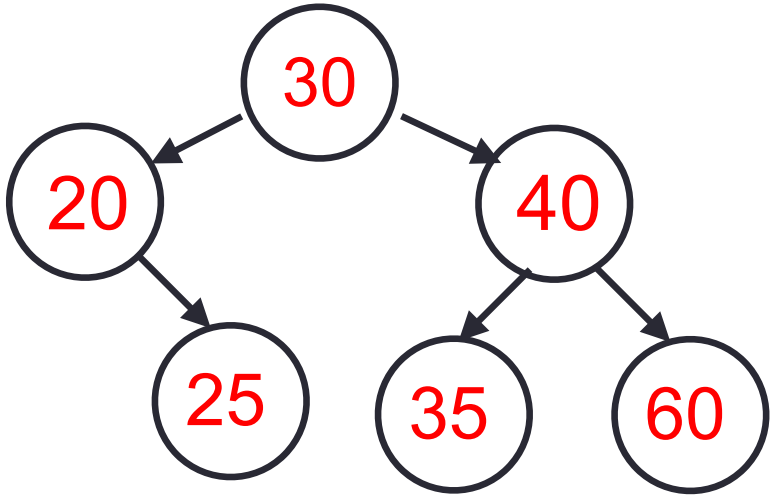
What actually happens:

```
for (std::set<int>::iterator it = s.begin(); it != s.end(); ++it) {
    cout << *it << " ";
}
```

Roadmap to implementing one at-a-time navigation for bst class

(1) Implement helpers: getmin and successor

```
Node* r = b.getmin(root);
while(r){
    cout << r->data << " ";
    r = b.successor(r);
}
```



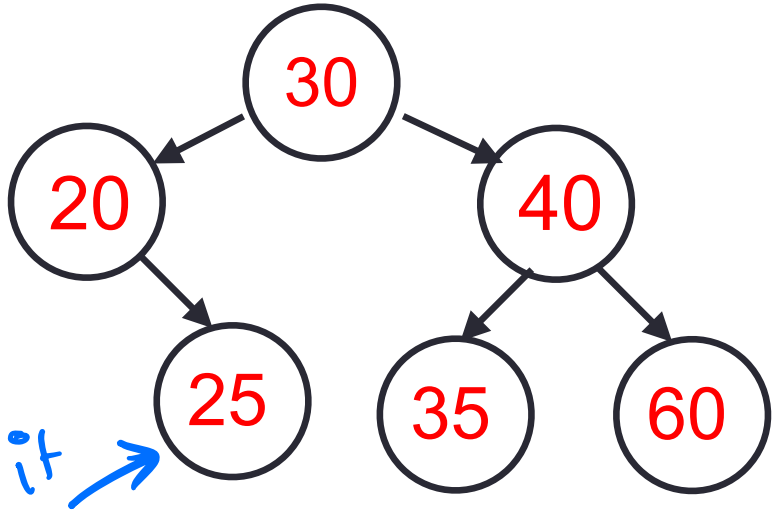
Are we done? Why/why not? - Discuss (2 mins)

Roadmap to implementing one at-a-time navigation for bst class

(1) Implement helpers: getmin and successor

(2) Implement a new ADT called `iterator` that **abstracts a traversal pointer**!!!

```
iterator it;
*it = 25  (data)
++it; // Moves to 30
```



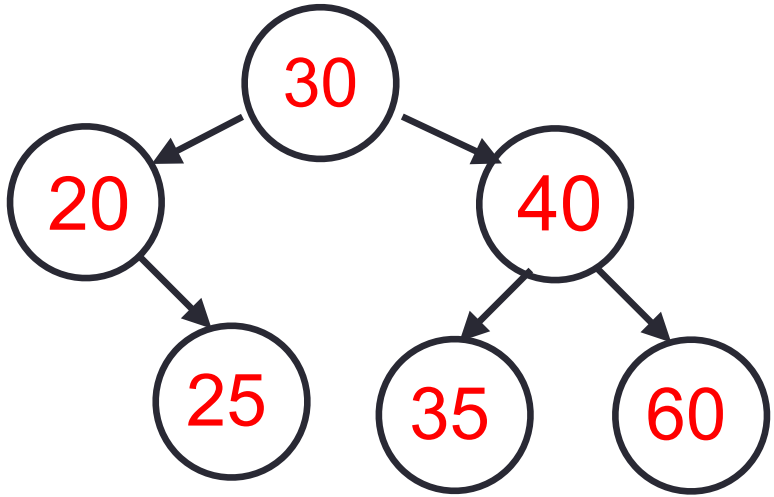
Discuss (2 mins):
What functions does iterator ADT need to allow operations like `*it` and `++it`?

Task 1: Implement two useful functions: getmin and successor

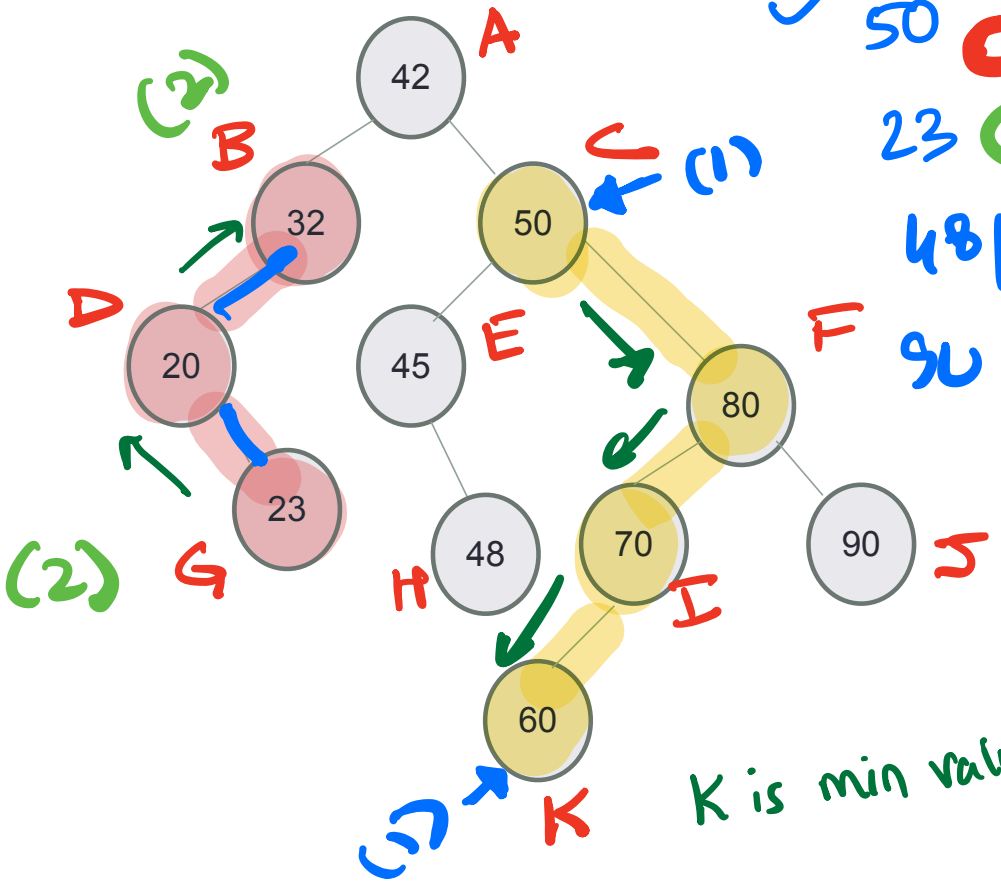
`getmin(root)`: returns pointer to Node with minimum value

`successor(n)`: returns pointer to the next Node (after n) in an in order traversal

Nodes visited in an inorder traversal:
20, 25, 30, 35, 40, 60



Discover the algo for successor



node	successor
50 C	→ K 60
23 G	→ B 32
48 H	→ C 50
90 J	→ null

(case 1)

K is min value in C's right subtree

Your turn (10 min): Work through handout 1.1-1.3

Task 1.1: Implement getmin to return the leftmost node in a subtree.

root of a BST

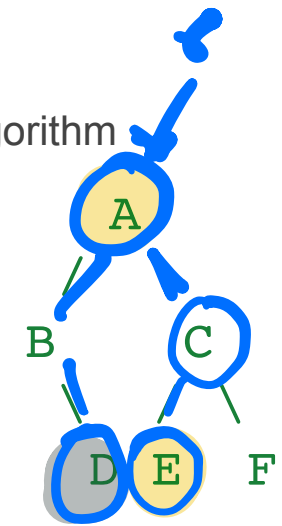
```
Node* bst::getmin(Node* r) const {
    // Fill in the code
```

return pointer to the leftmost node in the tree

```
}
```

Task 1.2: Discover the Successor Algorithm

Consider BST with keys masked by labels:



Succesor of A? E case 1

Successor of D? A case 2

What steps did you take in each case?

Task 1.3: Implement successor

any node in the BST

```
Node* bst::successor(Node* r) const {
    // Fill in the code
```

Case 1: If node (r) has a right child, then return min value in node(r)'s right subtree

Case 2: (No right child)

Traverse parent pointers until you reach a node (p) where r is in p's left subtree.

```
}
```

Brainstorm next steps

We can now write an iterative traversal for bst

```
Node* r = b.getmin(root);
while(r){
    cout << r->data << " ";
    r = b.successor(r);
}
```

Problem we encountered before: `Node` is private, so `Node*` can't be used externally.

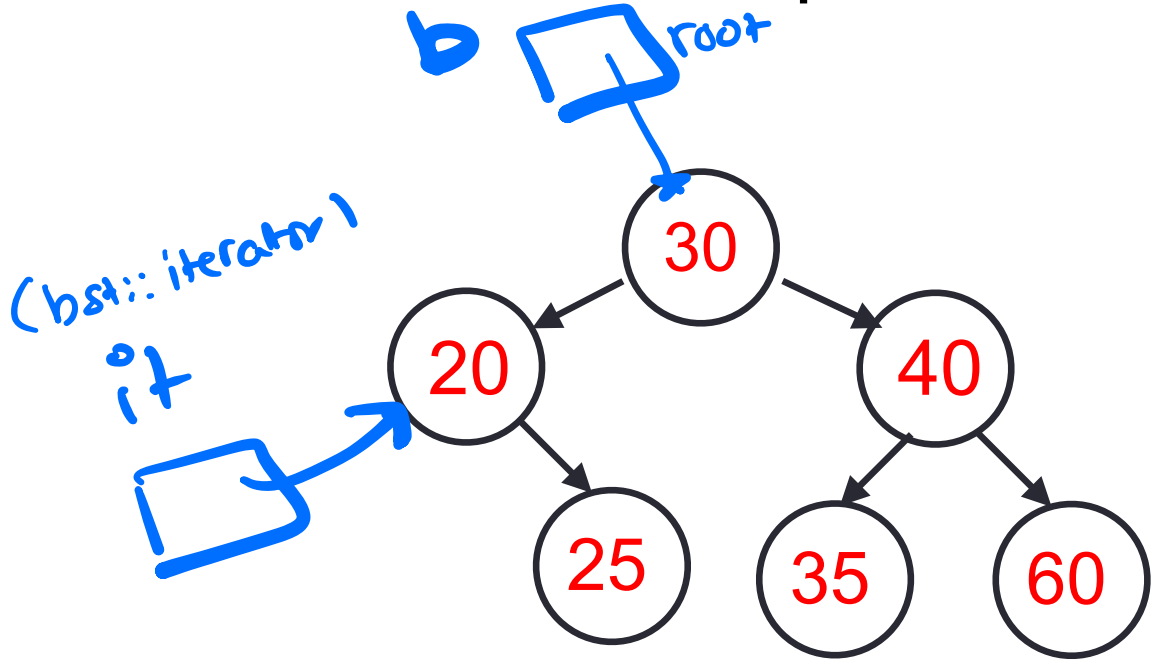
Big idea: Create a new ADT `iterator` that behaves like a traversal pointer.

Big idea: Create a new ADT called `iterator` that behaves like a pointer.

```
class bst::iterator {
  public:




  private:
    Node * curr;
    bst * tree;
};
```




```
bst::iterator it;
*it = 20 (data)
++it; // Moves to 25
```

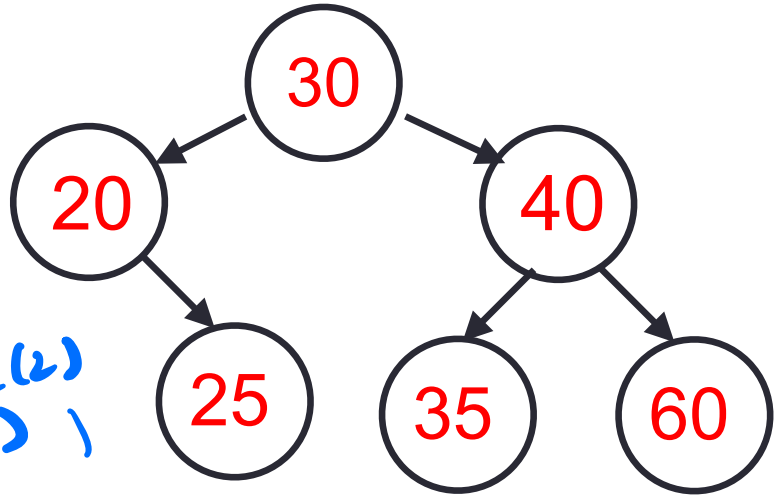
(5 min) Convert code to use iterator ADT

```
Node* r = b.getmin(root);
while(r){
    cout << r->data << " ";
    r = b.successor(r);
}
```

```
bst::iterator it = (1);
while ( it != (2) ){
    cout << *it << " ";
    ++it;
}
```

Discuss: What problem(s) do you encounter?

1. Need a way to initialize the iterator
   CBlank(1) → (implement bst::begin())
2. Need an "end" iterator to complete the condition for the while loop Blank(2)
   (implement bst::end())



BST Helper functions to initialize iterators

Task 3.1: Implement `begin`: Returns an iterator to the smallest node.

```
bst::iterator bst::begin() {
    // Fill in the code

        return iterator( getmin(root), this);


}
```

Task 3.2: Implement `end`: Returns an iterator for "past the end."

```
bst::iterator bst::end() {
 // Fill in the code

        return iterator( nullptr, this);


}
```

Task 4.1: Implement `operator*`

```
int bst::iterator::operator*() const {
    // Fill in the code



}
```

See code from lecture on github.

Task 4.2: Implement `operator++`

```
bst::iterator& bst::iterator::operator++() {
    // Fill in the code



}
```

Task 4.3: Implement `operator!=`

```
bool bst::iterator::operator!=(const iterator& rhs) {
    // Fill in the code



}
```

C++STL

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

In this lecture, you learned how to implement an iterator for any custom ADT. Useful for working with STL classes and writing clean code in the upcoming assignment (PA01) where you have to implement a card game.
The big challenge is to iterate through the cards of two players in a seamless way (no passing around pointers like Node* in the main logic of your game). Use iterators!