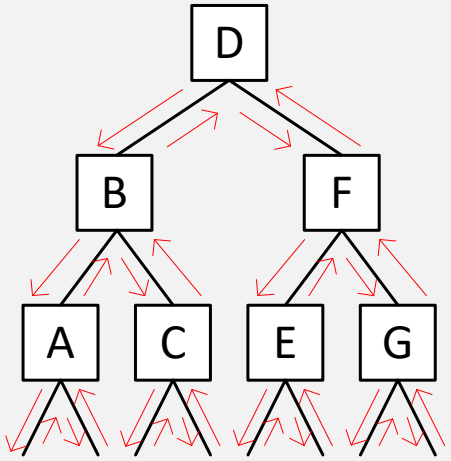
ITERATORS AND NESTED CLASSES

Adding sequential access to binary trees

Delroy A. Brinkerhoff

ITERATORS VS. "WALKING THE TREE"



Left until null
Visit
Right until null

- "Walking the tree"
  - Ordered processing of each node
  - All processing is done in the "walk" function
- Iterators
  - Ordered processing of each node
  - Program can leave the iterator, returning to the next node anytime
  - Iterators "remember" where they are

SPECIFYING NESTED CLASSES

```
class Outer
{
    public:
        class Nested
        {
            ...
        };
};
```

```
class Outer
{
    public:
        class Nested;
};

class Outer::Nested
{
    ...
};
```

TREE MEMBERS SUPPORTING ITERATORS

- `class iterator;`
- `int count(int number = 0);`
-
```
iterator get_keys()
{
    iterator i(this);
    return i;
}
```

```
template <class K, class V>
int KVTree<K,V>::count(int number)
{
    if (left != nullptr)
        number = left->count(number + 1);
    if (right != nullptr)
        number = right->count(number + 1);
    return number;
}
```

THE ITERATOR CLASS

```
template <class K, class V>
class KVTree<K, V>::iterator
{
    private:
        int size = 0;
        int index = 0;
        K* keys = nullptr;

    public:
        iterator(KVTree<K,V>* outer);
        iterator(iterator& i);
        ~iterator() { delete[] keys; }
        K next() { return keys[index++]; }
        bool has_next() { return index < size; }
        void reset() { index = 0; }
    private:
        void add_keys(KVTree<K,V>* tree);
}
```

ITERATOR FUNCTIONS

```
template<class K, class V>
KVTree<K,V>::iterator::
    iterator(KVTree<K,V>* outer)
{
    size = outer->count();
    keys = new K[size];
    if (outer->right != nullptr)
        add_keys(outer->right);
    else
        return;
    index = 0;
}
```

```
template <class K, class V>
void KVTree<K,V>::iterator::
    add_keys(KVTree<K,V>* outer)
{
    if (outer->left != nullptr)
        add_keys(outer->left);
    keys[index++] = outer->key;
    if (outer->right != nullptr)
        add_keys(outer->right);
}
```

USING THE TREE ITERATOR

```
KVTree<string, int>::iterator  keys = tree.get_keys();

while (keys.has_next())
{
    string  word = keys.next();
    int     count = *tree.search(word);
    cout << left << setw(20) << word <<
        right << setw(3) << count << endl;
}
```