5.3

Binary Tree Traversal and Tree Iterators

5.3

Binary Tree Traversal

- Visit each node in a tree exactly once
- Treat each node and its subtrees in the same fashion

Every time we visit a node A:

Inorder:	visit left → root → right
Preorder:	visit root → left → right
Postorder:	visit left → right → root

Binary Tree Traversal

- Visit each node in a tree exactly once
- Treat each node and its subtrees in the same fashion

Every time we visit a node A:

Inorder:	visit left → root → right:	B A C
Preorder:	visit root → left → right:	
Postorder:	visit left → right → root:	

Binary Tree Traversal

- Visit each node in a tree exactly once
- Treat each node and its subtrees in the same fashion

Every time we visit a node A:

Inorder:	visit left → root → right:	BAC
Preorder:	visit root → left → right:	A B C
Postorder:	visit left → right → root:	

29

Binary Tree Traversal

- Visit each node in a tree exactly once
- Treat each node and its subtrees in the same fashion

Every time we visit a node A:

Inorder:	visit left → root → right:	BAC
Preorder:	visit root → left → right:	ABC
Postorder:	visit left → right → root:	B C A

30

5.3.2

Inorder Traversal

- Steps of traversal:
  1. Moving down the tree toward the **left** until you can go no farther.
  2. **Visit** the node.
  3. Move one node to the **right** and continue step1.
- Use recursion to describe this traversal.

31

Inorder Traversal: Code

```
template < class T >
void Tree<T>::Inorder()
{ // Start a recursive inorder traversal
  // This function is a public member function of Tree
  Inorder(root);
}

template <class T>
void Tree<T>::Inorder(TreeNode<T>* currentNode)
{ // Recursive inorder traversal function
  // This function is a private member function of Tree
  if(currentNode){
     Inorder(currentNode->leftChild);
     Visit(currentNode); // e.g., printout information
     Inorder(currentNode->RightChild);
  }
}
```

5.3.3

Preorder Traversal: Code

```
template < class T >
void Tree<T>::Preorder()
{ // Start a recursive preorder traversal
  // This function is a public member function of Tree
  Preorder(root);
}

template <class T>
void Tree<T>::Preorder(TreeNode<T>* currentNode)
{ // Recursive preorder traversal function
  // This function is a private member function of Tree
  if(currentNode){
     Visit(currentNode); // e.g., printout information
     Preorder(currentNode->leftChild);
     Preorder(currentNode->RightChild);
  }
}
```

33

5.3.4

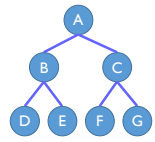
Postorder Traversal: Code

```
template < class T >
void Tree<T>::Postorder()
{ // Start a recursive postorder traversal
  // This function is a public member function of Tree
  Postorder(root);
}

template <class T>
void Tree<T>::Postorder(TreeNode<T>* currentNode)
{ // Recursive postorder traversal function
  // This function is a private member function of Tree
  if(currentNode){
     Postorder(currentNode->leftChild);
     Postorder(currentNode->RightChild);
     Visit(currentNode); // e.g., printout information
  }
}
```

34

Quiz



Traversal	Output ordered list
Inorder	
Preorder	
Postorder	

Tree Iterator

- We would like to visit nodes in a fashion as using **iterator's** visiting elements in a container
- Recursive traversal is no longer suitable
- We need an **iterator** version, but how?
  - Use stack to store non-visited nodes!

Non-Recursive Inorder Traversal

```
template < class T >
void Tree<T>::NonrecInorder()
{ // Non recursive inorder traversal using stack
  Stack<TreeNode<T>*> s;    // declare and init a stack
  TreeNode<T>* currentNode = root;
  while(1){
    while(currentNode){     // move down leftChild field
       s.Push(currentNode); // add to stack
       currentNode = currentNode->leftChild;
    }
    if(s.IsEmpty()) return; // all nodes are visited
    currentNode = s.Top();
    s.Pop();
    Visit(currentNode);    // e.g., printout information
    currentNode = currentNode->rightNode;
  }
}
```

We only need this part to develop tree iterator

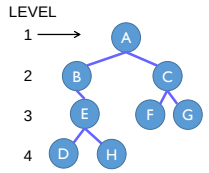
Inorder Iterator

```
Class InorderIterator{ // A nested class within Tree
public:
  InorderIterator() { currentNode = root}
  T* Next();
private:
  Stack<TreeNode<T>*> s;
  TreeNode<T>* currentNode;
};

T* InorderIterator::Next()
{
   while(currentNode){     // Move down leftChild field
      s.Push(currentNode); // Add to stack
      currentNode = currentNode->leftChild;
   }
   if(s.IsEmpty()) return NULL; // All nodes are visited
   currentNode = s.Top();
   s.Pop();
   T& temp = currentNode->data;
   currentNode = currentNode->rightNode;
   return &temp;
}
```

5.3.6 Level-Order Traversal

- Visit nodes in a top to down, left to right manner: A B C E F G D H



Preorder	Inorder	Postorde	Level-
Stack	Stack	Stack	Queue

40

Level-Order Traversal: Code

```
template <class T>
void Tree<T>::LevelOrder()
{ // Traverse the binary tree in level order
  Queue<TreeNode<T>*> q;
  TreeNode<T>* currentNode = root;
  while(currentNode){
    Visit(currentNode);
    if(currentNode->leftChild) q.Push(currentNode->leftChild);
    if(currentNode->rightChild) q.Push(currentNode->rightChild);
    if(q.IsEmpty()) return;
    currentNode = q.Front();
    q.Pop();
  }
}
```

Self-Study Topics

- Binary tree operations
  - Preorder traversal (Non-recursive & iterator)
  - Postorder traversal (Non-recursive & iterator)
  - Copying Binary Trees
  - Testing Equality