Chap 5. Trees (2)

Contents

5.3 Binary Tree Traversal

- Traversing a tree
  - Visiting each node in the tree exactly once

- When traversing a binary tree,
  - L, V, R : *moving left*, *visiting the node*, *moving right*
  - Six possible combinations of traversal
    - LVR, LRV, VLR, VRL, RVL, RLV

  - If we traverse left before right, only tree remains
    - LVR: ***inorder***
    - LRV: ***postorder***
    - VLR: ***preorder***

- There is a natural correspondence between
  - *these traversals* and *producing the infix, postfix, and prefix forms of an expression*.

- Consider a binary tree for *A/B*C*D+E*
  - For each node that contains an operator,
    - its left subtree gives the left operand and
    - its right subtree the right operand.

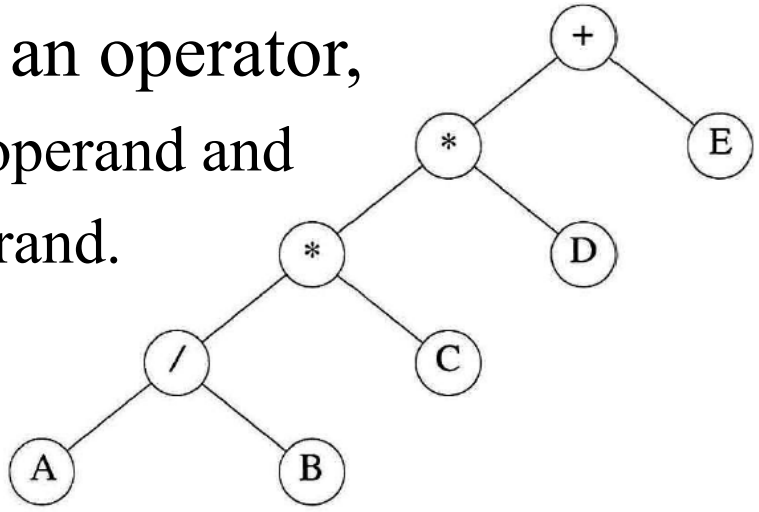


Figure 5.16: Binary tree with arithmetic expression

- Binary tree for postfix expression

postfix expression으로부터

연결리스트를 사용한 이진트리를 만드는 알고리즘

왼쪽에서 오른쪽으로 수식을 스캐닝하면서 다음을 수행함

*AB/C*D*E+*

1. 토큰이 operand 라면

   ① 노드를 생성한 후 값을 넣고 //※ 노드의 두 링크는 NULL

   ② stack에 push

2. 토큰이 operator 이면

   ① 노드를 생성한 후

   ② 오른쪽 자식으로 stack에서 pop한 노드를 연결하고

   ③ 왼쪽 자식으로 stack에서 또 pop한 노드를 연결한 후

   ④ 그 operator 노드를 stack에 push

3. stack에 마지막으로 남은 노드가 root이다.

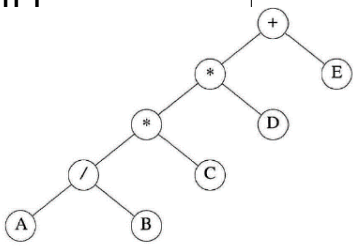


Figure 5.16: Binary tree with arithmetic expression

5.3.1 Inorder Traversal

```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
      inorder(ptr→leftChild);
      printf("%d",ptr→data);
      inorder(ptr→rightChild);
   }
}
```

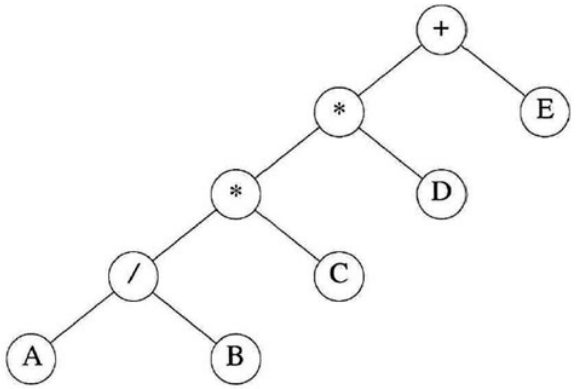
Program 5.1: Inorder traversal of a binary tree

1. Return if the tree is null
2. Inorder traversal of the left subtree
3. Print the value
4. Inorder traversal of the right subtree

Example

```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
      inorder(ptr→leftChild);
      printf("%d",ptr→data);
      inorder(ptr→rightChild);
   }
}
```

Program 5.1: Inorder traversal of a binary tree



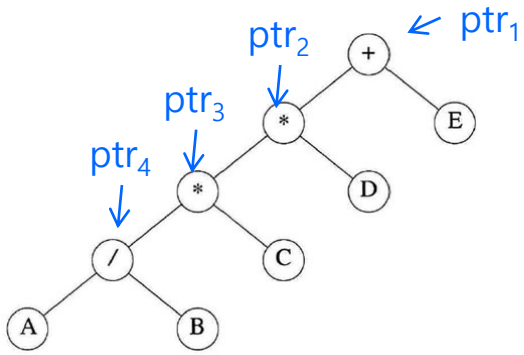
A/B*C*D+E

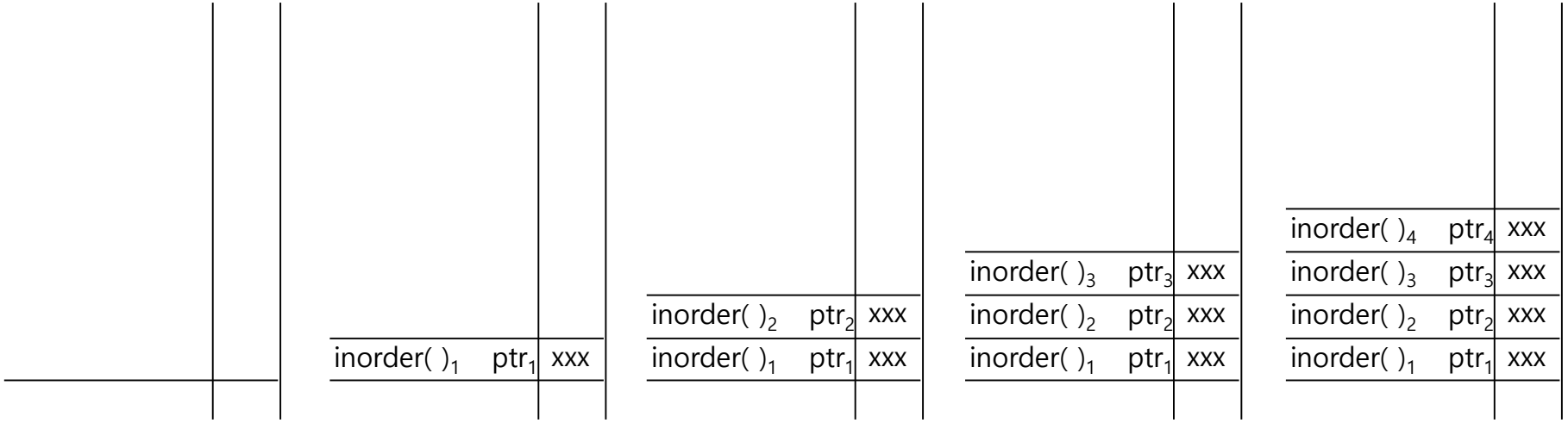
Call of *inorder*	Value in *root*	Action	Call of *inorder*	Value in *root*	Action
1	+		11	C	
2	*		12	NULL	
3	*		11	C	**printf**
4	/		13	NULL	
5	A		2	*	**printf**
6	NULL		14	D	
5	A	**printf**	15	NULL	
7	NULL		14	D	**printf**
4	/	**printf**	16	NULL	
8	B		1	+	**printf**
9	NULL		17	E	
8	B	**printf**	18	NULL	
10	NULL		17	E	**printf**
3	*	**printf**	19	NULL	

Trace of Program 5.1

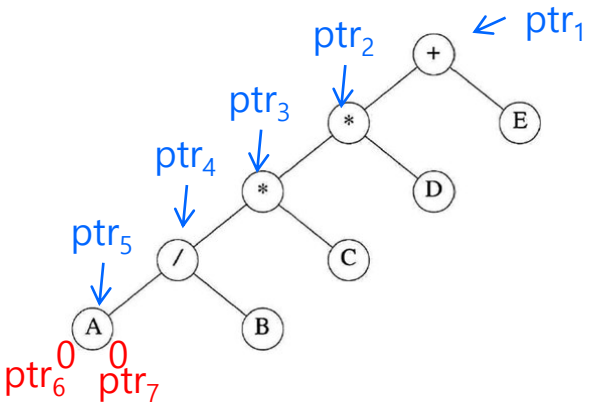
System Stack

```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
   ① inorder(ptr→leftChild);
     printf("%d",ptr→data);
   ② inorder(ptr→rightChild);
   }
}
```





```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
      inorder(ptr→leftChild);
      printf("%d",ptr→data);
      inorder(ptr→rightChild);
   }
}
```



inorder( )$_5$	ptr$_5$	xxx
inorder( )$_4$	ptr$_4$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_6$	ptr$_6$	**0**
inorder( )$_5$	ptr$_5$	xxx
inorder( )$_4$	ptr$_4$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

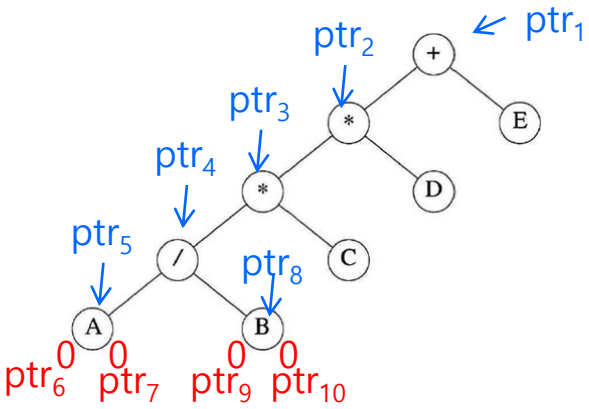
inorder( )$_5$	ptr$_5$	xxx
inorder( )$_4$	ptr$_4$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_7$	ptr$_7$	**0**
inorder( )$_5$	ptr$_5$	xxx
inorder( )$_4$	ptr$_4$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_5$	ptr$_5$	xxx
inorder( )$_4$	ptr$_4$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

Output: A

```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
   ① inorder(ptr→leftChild);
     printf("%d",ptr→data);
   ② inorder(ptr→rightChild);
   }
}
```



inorder( )4	ptr4	xxx
inorder( )3	ptr3	xxx
inorder( )2	ptr2	xxx
inorder( )1	ptr1	xxx

inorder( )8	ptr8	xxx
inorder( )4	ptr4	xxx
inorder( )3	ptr3	xxx
inorder( )2	ptr2	xxx
inorder( )1	ptr1	xxx

inorder( )9	ptr9	**0**
inorder( )8	ptr8	xxx
inorder( )4	ptr4	xxx
inorder( )3	ptr3	xxx
inorder( )2	ptr2	xxx
inorder( )1	ptr1	xxx

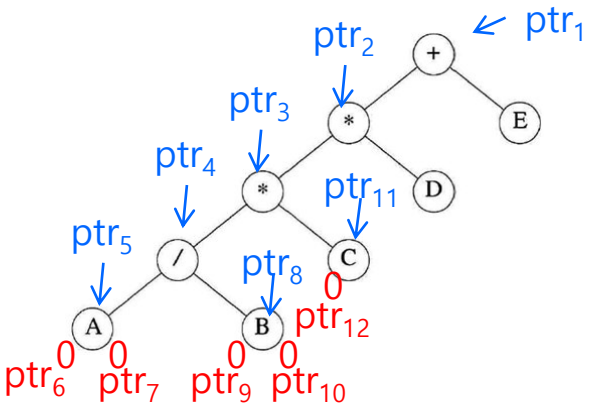
Output: /

inorder( )8	ptr8	xxx
inorder( )4	ptr4	xxx
inorder( )3	ptr3	xxx
inorder( )2	ptr2	xxx
inorder( )1	ptr1	xxx

inorder( )10	ptr10	**0**
inorder( )8	ptr8	xxx
inorder( )4	ptr4	xxx
inorder( )3	ptr3	xxx
inorder( )2	ptr2	xxx
inorder( )1	ptr1	xxx

Output: B

```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
      inorder(ptr→leftChild);
      printf("%d",ptr→data);
      inorder(ptr→rightChild);
   }
}
```



inorder( )$_8$	ptr$_8$	xxx
inorder( )$_4$	ptr$_4$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_4$	ptr$_4$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

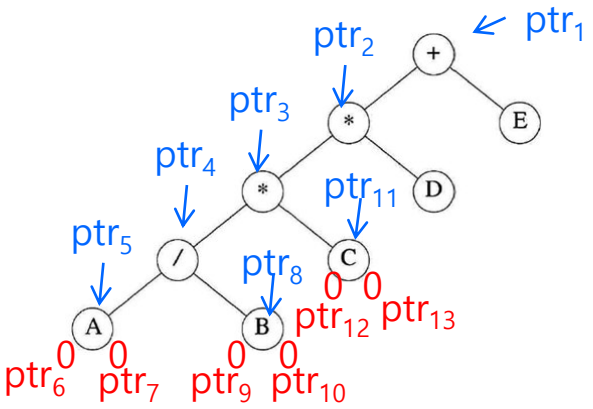
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_{11}$	ptr$_{11}$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_{12}$	ptr$_{12}$	**0**
inorder( )$_{11}$	ptr$_{11}$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

Output: *

```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
      inorder(ptr→leftChild);
      printf("%d",ptr→data);
      inorder(ptr→rightChild);
   }
}
```



inorder( )$_{11}$	ptr$_{11}$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_{13}$	ptr$_{12}$	**0**
inorder( )$_{11}$	ptr$_{11}$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_{11}$	ptr$_{11}$	xxx
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

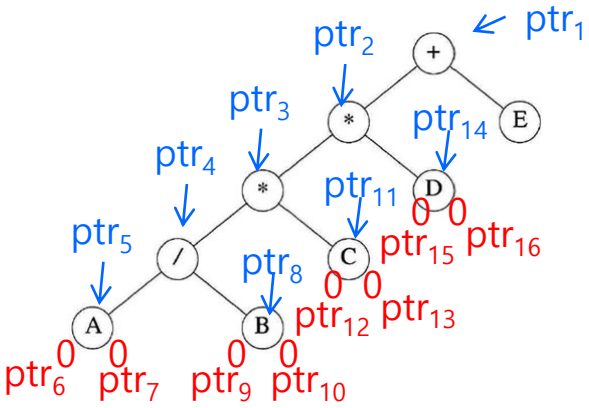
inorder( )$_3$	ptr$_3$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

Output: C

Output: *

```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
      inorder(ptr→leftChild);
      printf("%d",ptr→data);
      inorder(ptr→rightChild);
   }
}
```



inorder( )$_{14}$	ptr$_{14}$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_{15}$	ptr$_{15}$	**0**
inorder( )$_{14}$	ptr$_{14}$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

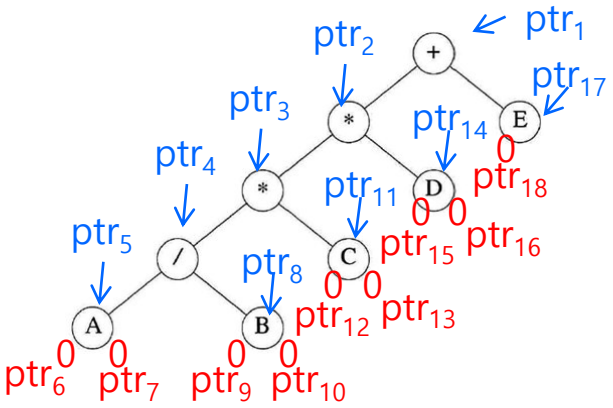
inorder( )$_{14}$	ptr$_{14}$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

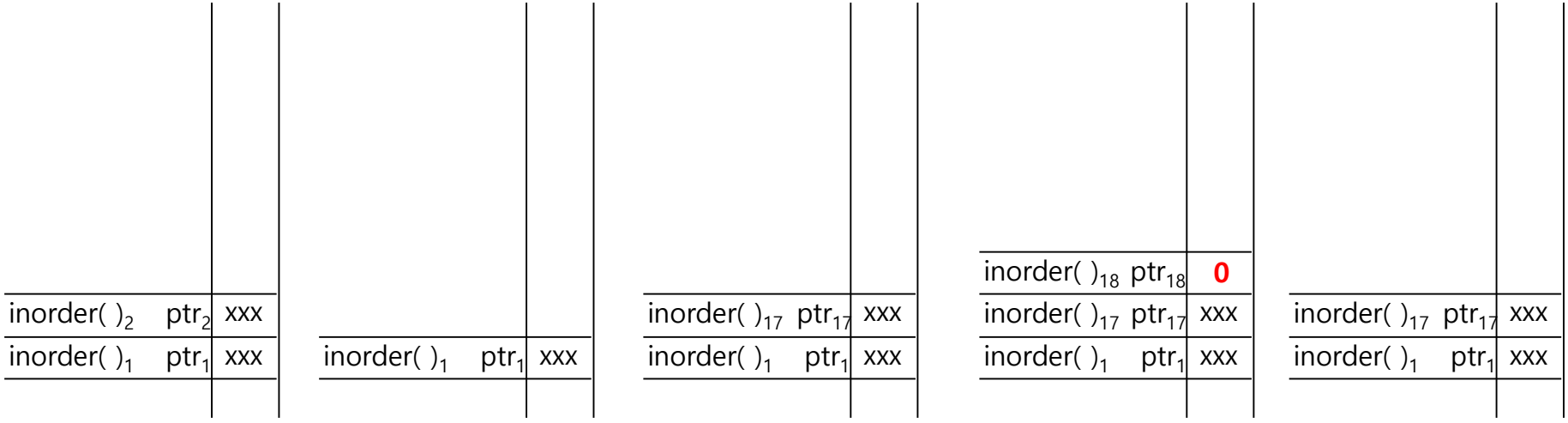
inorder( )$_{16}$	ptr$_{16}$	**0**
inorder( )$_{14}$	ptr$_{14}$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

inorder( )$_{14}$	ptr$_{14}$	xxx
inorder( )$_2$	ptr$_2$	xxx
inorder( )$_1$	ptr$_1$	xxx

Output: D

```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
      inorder(ptr→leftChild);
      printf("%d",ptr→data);
      inorder(ptr→rightChild);
   }
}
```

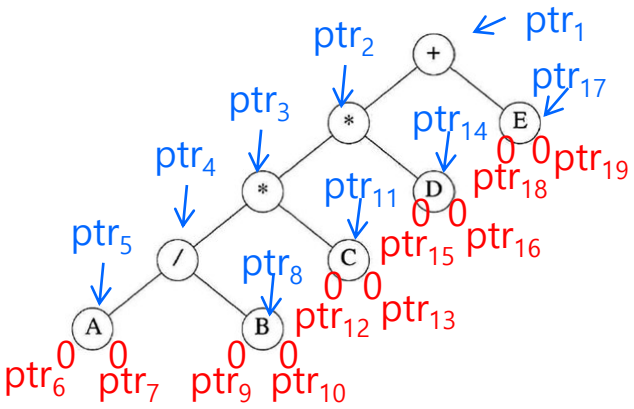


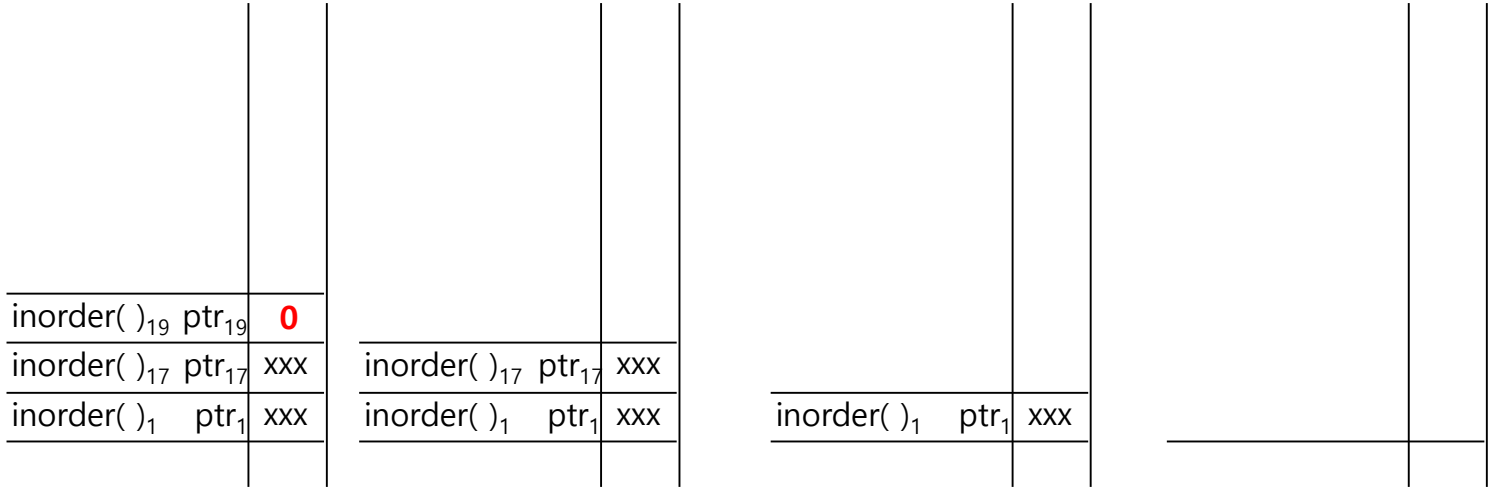


Output: +

Output: E

```
void inorder(treePointer ptr)
{/* inorder tree traversal */
   if (ptr) {
      inorder(ptr→leftChild);
      printf("%d",ptr→data);
      inorder(ptr→rightChild);
   }
}
```



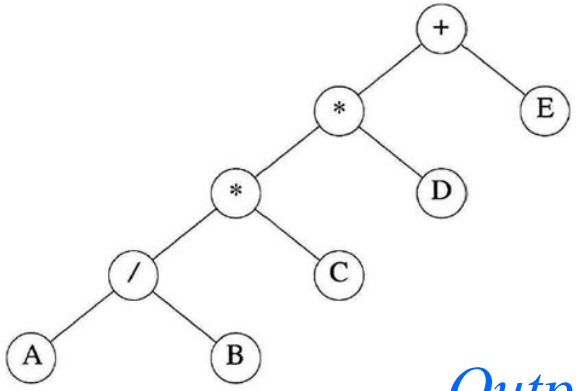


The number of calls of inorder?

Example

```
void postorder(treePointer ptr)
{/* postorder tree traversal */
   if (ptr) {
      postorder(ptr→leftChild);
      postorder(ptr→rightChild);
      printf("%d",ptr→data);
   }
}
```

Program 5.3: Postorder traversal of a binary tree



Output ?

AB/C*D*E+

Call of *postorder*	Value in *root*	Action	Call of *postorder*	Value in *root*	Action

5.3.2 Preorder Traversal

```
void preorder(treePointer ptr)
{/* preorder tree traversal */
   if (ptr) {
      printf("%d",ptr→data);
      preorder(ptr→leftChild);
      preorder(ptr→rightChild);
   }
}
```

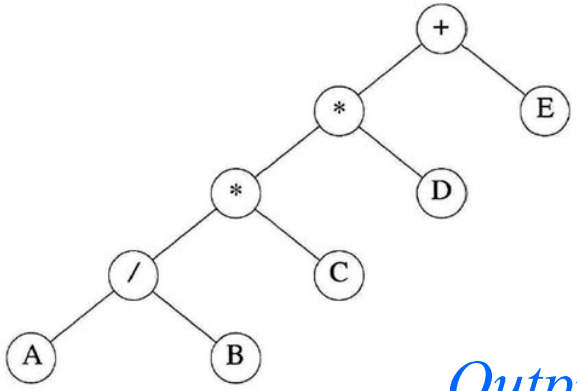
Program 5.2: Preorder traversal of a binary tree

1. Return if the tree is null
2. Print the value
3. Preorder traversal of the left subtree
4. Preorder traversal of the right subtree

Example

```
void preorder(treePointer ptr)
{/* preorder tree traversal */
   if (ptr) {
      printf("%d",ptr→data);
      preorder(ptr→leftChild);
      preorder(ptr→rightChild);
   }
}
```

Program 5.2: Preorder traversal of a binary tree



Output ?

+/ABCDE**

Call of *preorder*	Value in *root*	Action	Call of *preorder*	Value in *root*	Action

5.3.3 Postorder Traversal

```
void postorder(treePointer ptr)
{/* postorder tree traversal */
   if (ptr) {
      postorder(ptr→leftChild);
      postorder(ptr→rightChild);
      printf("%d",ptr→data);
   }
}
```

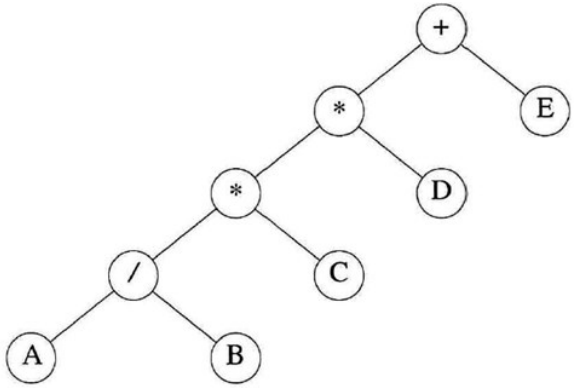
Program 5.3: Postorder traversal of a binary tree

1. Return if the tree is null
2. Postorder traversal of the left subtree
3. Postorder traversal of the right subtree
4. Print the value

Example

```
void postorder(treePointer ptr)
{/* postorder tree traversal */
   if (ptr) {
      postorder(ptr→leftChild);
      postorder(ptr→rightChild);
      printf("%d",ptr→data);
   }
}
```

Program 5.3: Postorder traversal of a binary tree



AB/C*D*E+

Call of *postorder*	Value in *root*	Action	Call of *postorder*	Value in *root*	Action

5.3.4 Iterative Inorder Traversal

- We can develop equivalent iterative functions instead of using recursion.
- To simulate recursion, we must create *our own stack*.

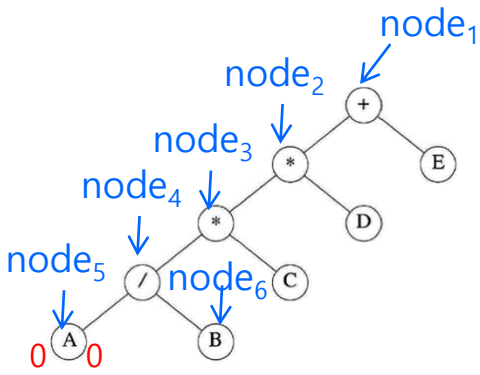
```
void iterInorder(treePointer node)
{
   int top = -1; /* initialize stack */
   treePointer stack[MAX_STACK_SIZE];
   for (;;) {
      for(; node; node = node→leftChild)
         push(node); /* add to stack */
      node = pop(); /* delete from stack */
      if (!node) break; /* empty stack */
      printf("%d", node→data);
      node = node→rightChild;
   }
}
```

※ Declare as global variables

Program 5.4: Iterative inorder traversal

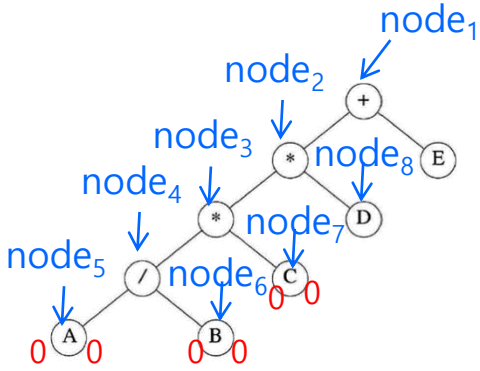
User-Defined Stack

```
void iterInorder(treePointer node)
{
   int top = -1; /* initialize stack */
   treePointer stack[MAX_STACK_SIZE];
   for (;;) {
      for(; node; node = node→leftChild)
         push(node); /* add to stack */
      node = pop(); /* delete from stack */
      if (!node) break; /* empty stack */
      printf("%d", node→data);
      node = node→rightChild;
   }
}
```



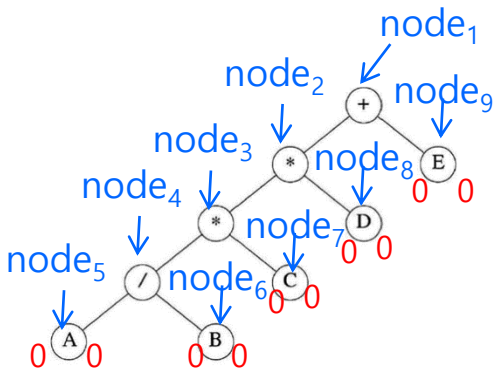
					$node_5$	$node_5$				
			$node_4$	$node_4$	$node_4$	$node_4$	$node_4$		$node_6$	
		$node_3$	$node_3$	$node_3$	$node_3$	$node_3$	$node_3$	$node_3$	$node_3$	
	$node_2$	$node_2$	$node_2$	$node_2$	$node_2$	$node_2$	$node_2$	$node_2$	$node_2$	
stack	$node_1$	$node_1$	$node_1$	$node_1$	$node_1$	$node_1$	$node_1$	$node_1$	$node_1$	$node_1$
node	$node_1$	$node_2$	$node_3$	$node_4$	$node_5$	0	$node_5$	0	$node_4$	$node_6$
output							A		/	

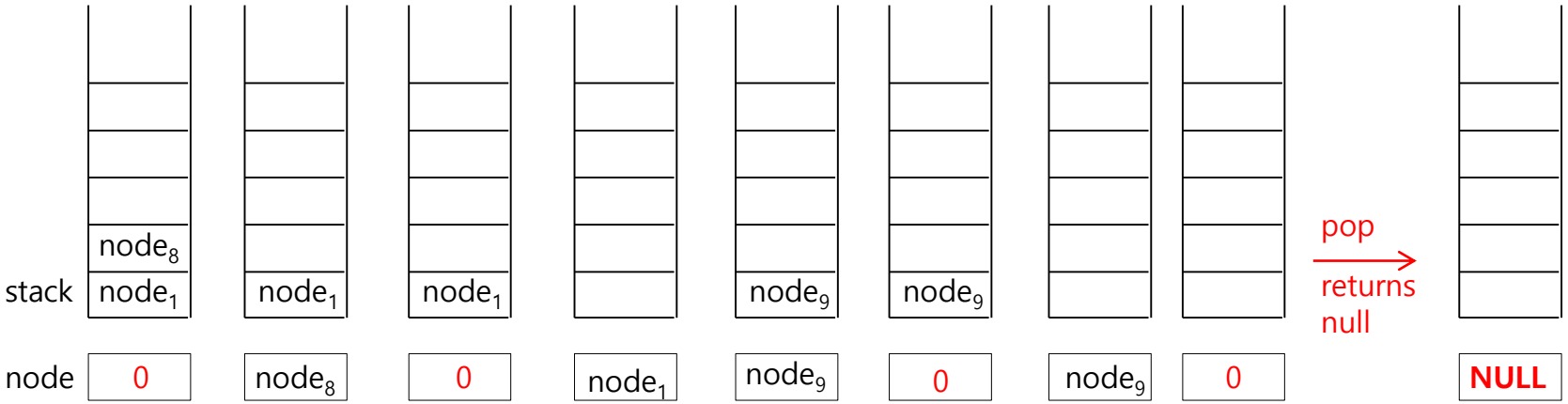
```
void iterInorder(treePointer node)
{
   int top = -1; /* initialize stack */
   treePointer stack[MAX_STACK_SIZE];
   for (;;) {
      for(; node; node = node→leftChild)
         push(node); /* add to stack */
      node = pop(); /* delete from stack */
      if (!node) break; /* empty stack */
      printf("%d", node→data);
      node = node→rightChild;
   }
}
```



	node$_6$								
	node$_3$	node$_3$	node$_3$		node$_7$	node$_7$			
	node$_2$	node$_2$	node$_2$	node$_2$	node$_2$	node$_2$	node$_2$	node$_2$	node$_8$
stack	node$_1$	node$_1$	node$_1$	node$_1$	node$_1$	node$_1$	node$_1$	node$_1$	node$_1$
node	0	node$_6$	0	node$_3$	node$_7$	0	node$_7$	0	node$_2$	node$_8$
output		B		*			C		*	

```
void iterInorder(treePointer node)
{
   int top = -1; /* initialize stack */
   treePointer stack[MAX_STACK_SIZE];
   for (;;) {
      for(; node; node = node→leftChild)
         push(node); /* add to stack */
      node = pop(); /* delete from stack */
      printf("%d", node→data);
      node = node→rightChild;
   }
}
```





The number of calls of push?

5.3.5 Level-Order Traversal

- A traversal that requires a *queue*.
- Visit the root first, the root's left child, followed by the root's right child
- Continue, visiting the node at each new level from the leftmost node to the rightmost node

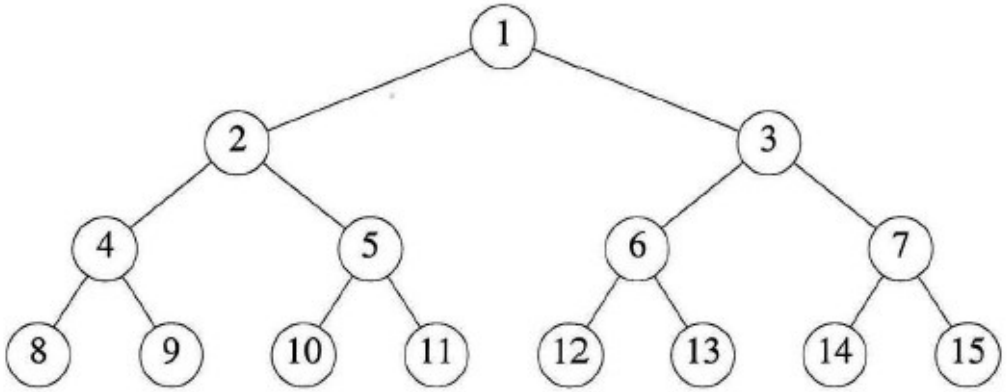


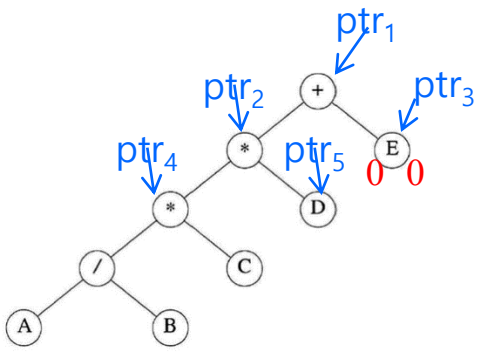
Figure 5.11: Full binary tree of depth 4 with sequential node numbers

```
void levelOrder(treePointer ptr)
{/* level order tree traversal */
   int front = 0, rear = 0;
   treePointer queue[MAX_QUEUE_SIZE];
   if (!ptr) return; /* empty tree */
   addq(ptr);
   for (;;) {
      ptr = deleteq();
      if (ptr) {
         printf("%d",ptr→data);
         if(ptr→leftChild)
            addq(ptr→leftChild);
         if (ptr→rightChild)
            addq(ptr→rightChild);
      }
      else break;
   }
}
```

※ Declare as a global circular queue

Program 5.5: Level-order traversal of a binary tree

```
void levelOrder(treePointer ptr)
{/* level order tree traversal */
   int front = rear = 0;
   treePointer queue[MAX_QUEUE_SIZE];
   if (!ptr) return; /* empty tree */
   addq(ptr);
   for (;;) {
      ptr = deleteq();
      if (ptr) {
         printf("%d",ptr→data);
         if(ptr→leftChild)
            addq(ptr→leftChild);
         if (ptr→rightChild)
            addq(ptr→rightChild);
      }
      else break;
   }
}
```



			output
ptr	ptr_1	f r	
ptr	ptr_1	ptr_1 f r	
ptr	ptr_1	ptr_1 fr	+
ptr	ptr_1	ptr_1 ptr_2 ptr_3 f r	
ptr	ptr_2	ptr_3 f r	*
ptr	ptr_2	ptr_3 ptr_4 ptr_5 f r	
ptr	ptr_3	ptr_4 ptr_5 f r	E
ptr	ptr_4	ptr_5 f r	*

```
void levelOrder(treePointer ptr)
{/* level order tree traversal */
   int front = rear = 0;
   treePointer queue[MAX_QUEUE_SIZE];
   if (!ptr) return; /* empty tree */
   addq(ptr);
   for (;;) {
      ptr = deleteq();
      if (ptr) {
         printf("%d",ptr→data);
         if(ptr→leftChild)
            addq(ptr→leftChild);
         if (ptr→rightChild)
            addq(ptr→rightChild);
      }
      else break;
   }
}
```