Data Structure

Week 11

KyuDong SIM

1. 이번 주 실습 내용

-Binary Tree Treversal

Traversal

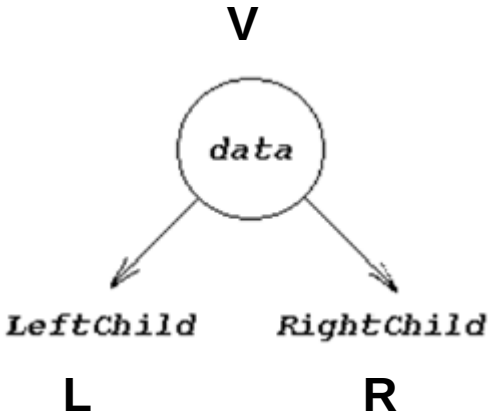
- 모든 Node를 지나는 방법
- Binary Tree는 L, V, R로 구성
  -> 3! = LVR, LRV, VLR, VRL, RVL, RLV

- Left 다음 Right 면 3가지

  LVR: inorder -> infix

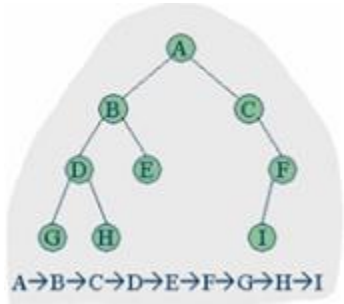
  LRV: postorder -> postfix

  VLR: preorder -> prefix



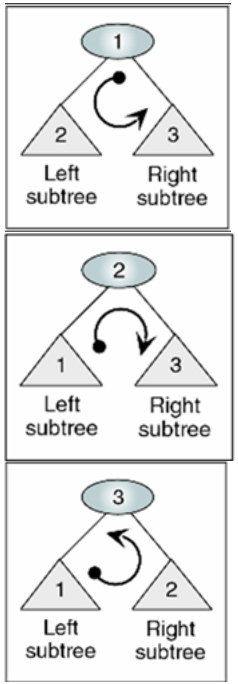
Traversal

- 단순 Singly Linked List와 달리 각 Node 들을 방문하기 위한 규칙 필요
- Stack 기반 (참고 : Depth First Search)
  - Pre-order : 전위 탐색
  - In-order : 중위 탐색, 대칭 탐색
  - Post-order : 후위 탐색
  - Recursive 호출을 통해 구현 가능 (함수 호출이 스택과 동일한 효과임을 이용)
- Queue 기반 (참고 : Breath First Search)
  - Level-order
  - Queue 자료 구조로 구현



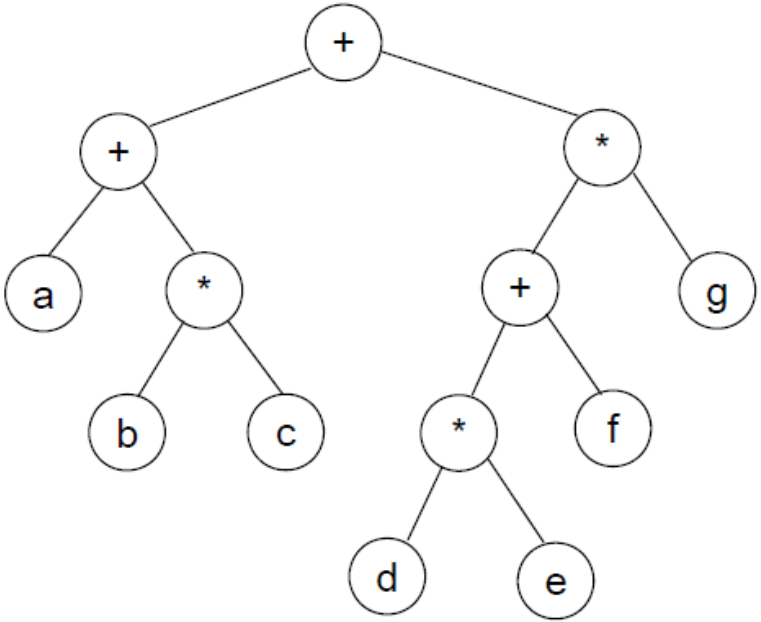
Stack based Traversal

- Pre-order : 자기를 먼저. 그 다음 왼쪽. 마지막에 오른쪽 탐색.
  [+,+,a,*,b,c,*,+,*,d,e,f,g]
- In-order : 왼쪽 먼저 탐색. 그 다음 자기 자신. 마지막에 오른쪽 탐색.
  [a,+,b,*,c,+,d,*,e,+,f,*,g]
- Post-order : 왼쪽 먼저 탐색. 그 다음 오른쪽 탐색. 마지막에 자기 자신.
  [a,b,c,*,+,d,e,*,f,+,g,*,+]



Stack based Traversal

예시 그래프)



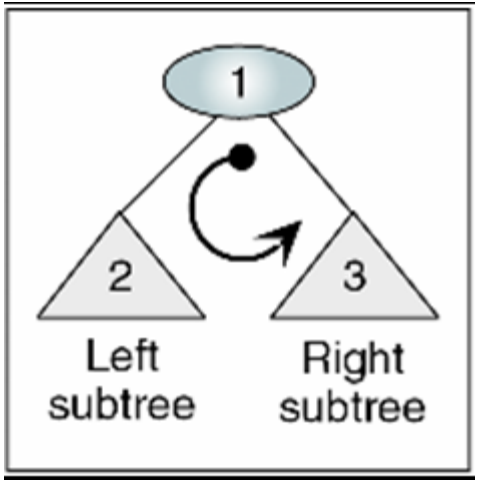
Division of Computer Science and Engineering, Hanyang University

Pre-order

- 현재 노드, Left subtree, Right subtree 순으로 출력

```
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

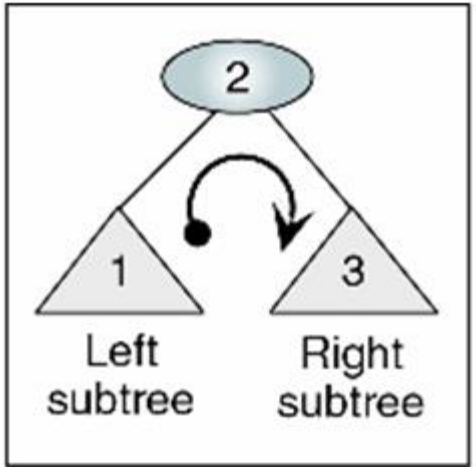


In-order

- Left subtree, 현재 노드, Right subtree 순으로 출력

```
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

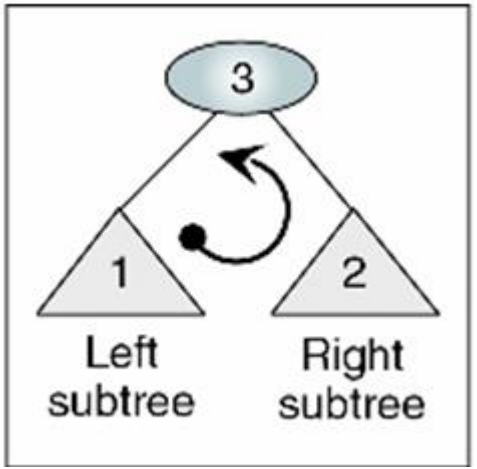


Post-order

- Left subtree, Right subtree 현재 노드, 순으로 출력

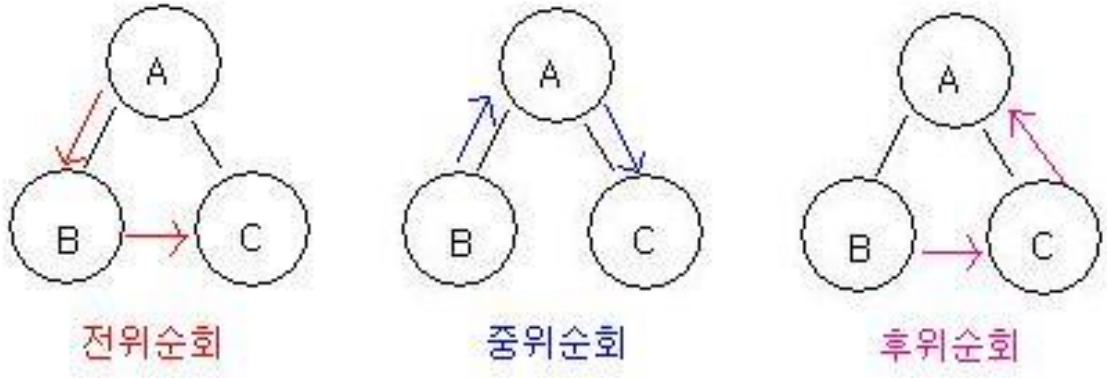
```
void postorder(treePointer ptr)
{/* postorder tree traversal */
  if (ptr) {
    postorder(ptr->leftChild);
    postorder(ptr->rightChild);
    printf("%d",ptr->data);
  }
}
```

Program 5.3 : Postorder traversal of a binary tree

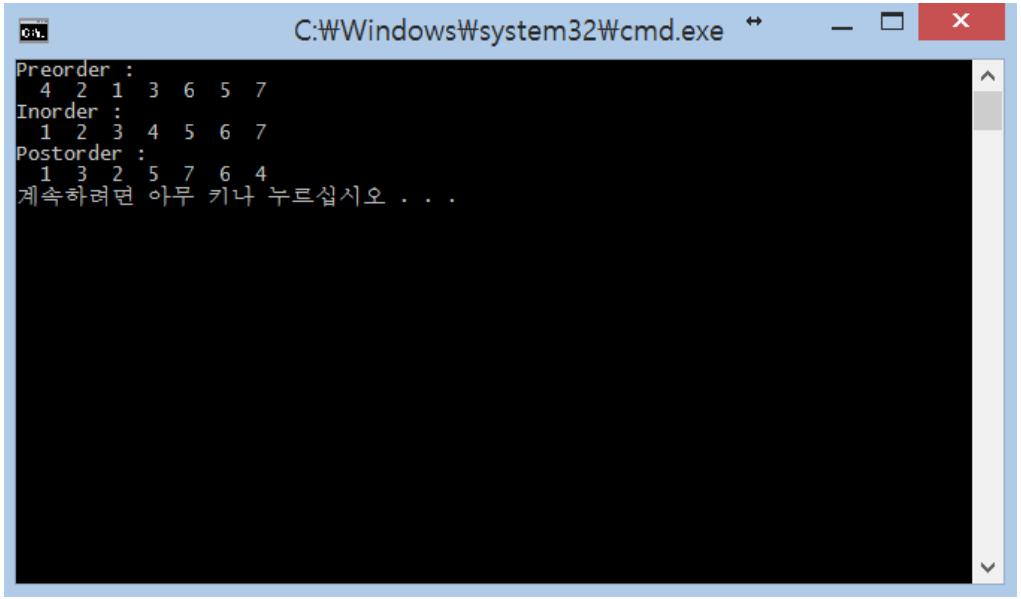


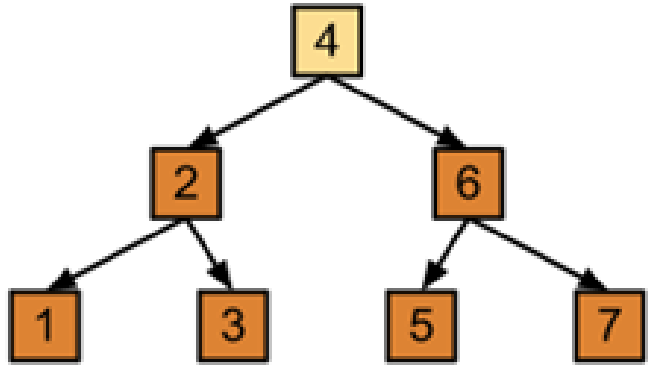
Pre-order vs. In-order vs. Post-order

- Pre-order : A B C
- In-order : B A C
- Post-order : B C A



실습 결과 예






```
TreeNode* tree = NULL;
insertTreeNode(&tree, 4);
insertTreeNode(&tree, 2);
insertTreeNode(&tree, 6);
insertTreeNode(&tree, 1);
insertTreeNode(&tree, 3);
insertTreeNode(&tree, 5);
insertTreeNode(&tree, 7);
```

Tree data type

```
typedef struct _TreeNode TreeNode;

struct _TreeNode
{
    int data;
    TreeNode* leftChild;
    TreeNode* rightChild;
};
```

- 각 노드는 데이터, leftChild, rightChild를 가짐

Insert Tree

```
void insertTreeNode(TreeNode** p, int value)
{
    if ((*p) == NULL)
        (*p) = createTreeNode(value);
    else if ((*p)->data > value)
        insertTreeNode(&((*p)->leftChild), value);
    else
        insertTreeNode(&((*p)->rightChild), value);

}
```

- Leaf에 Value값을 가지는 노드 생성
- 입력 value가 현재 노드의 값보다 작으면 leftChild로 접근
- 입력 value가 현재 노드의 값보다 크면 rightChild로 접근

제출 및 알림

수업 중 확인 or 메일제출 (학번 기입)

메일 제출 :

주소 : (89kdsim@naver.com)

기한 : ~2016-05-18