Data Structure #5

Tree

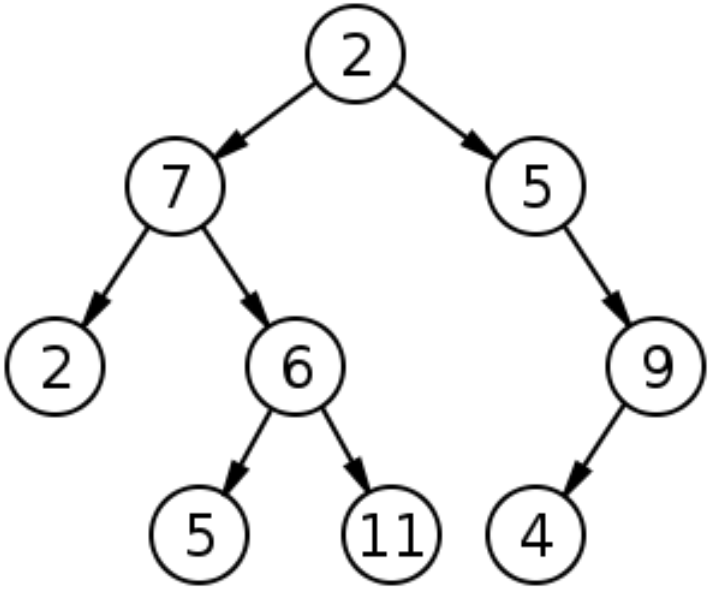
2020년 1학기

Intro.

- **실습주제 소개**
  - tree ADT
  - tree traversal
    - Inorder
    - Preorder
    - Postorder

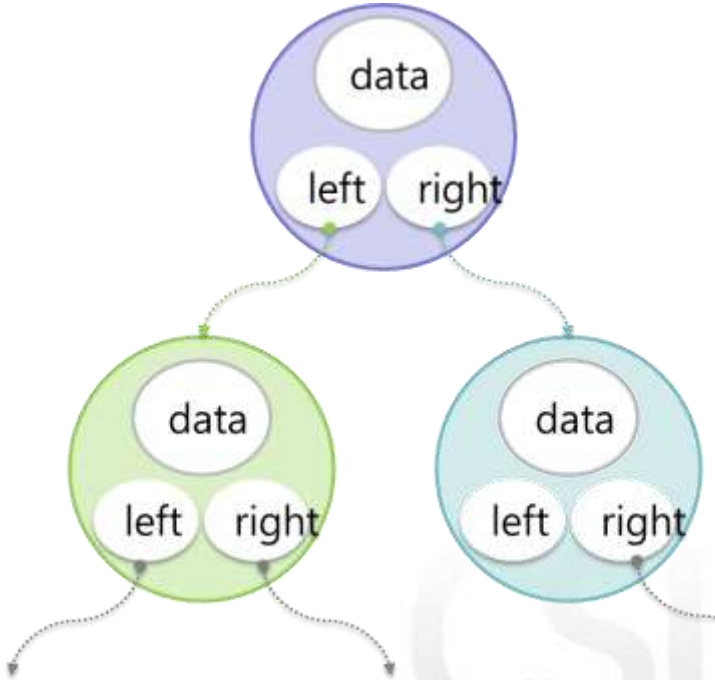
이진 트리

- **이진 트리란?**
  - 모든 노드가 최대 2개의 서브 트리를 가질 수 있는 트리
  - 차수: 서브트리의 수를 차수(degree)라고 함



이진 트리

- **이진트리의 구현 시 필요한 요소**
  - 새로운 노드 생성(왼쪽 서브 트리 포인터, 데이터, 오른쪽 서브 트리 포인터)
  - 노드에 데이터 삽입
  - 서브 트리의 루트 데이터 표시
  - 서브 트리 삽입 연산



이진 트리의 구현

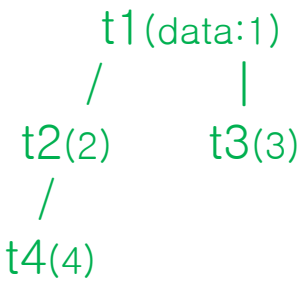
- 이진트리의 구조체

```
typedef int element;
typedef struct _TreeNode
{
        element data;
        struct _TreeNode * left;
        struct _TreeNode * right;
} TreeNode;
```

이진 트리의 구현(전역변수로 구현)

– 전역변수, 메인 함수 **[실제로 잘 사용하지 않음!]**

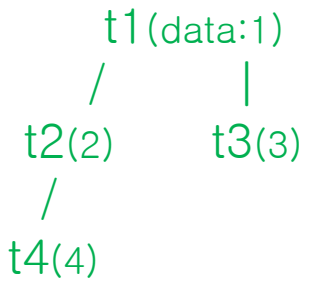
```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

TreeNode t4 = {4,NULL, NULL};
TreeNode t3 = {3,NULL, NULL};
TreeNode t2 = {2, &t4, NULL};
TreeNode t1 = {1, &t2, &t3};

int main(void){

        printf("%d", t2.data); //result: 2
        printf("%d", t4.data); //result: 4

        return 0;
}
```



이진 트리의 구현(동적할당으로 구현)

– 메인 함수

```
int main(void){
        TreeNode * t1, * t2, * t3, * t4;
        t1 = (TreeNode *)malloc(sizeof(TreeNode));
        t2 = (TreeNode *)malloc(sizeof(TreeNode));
        t3 = (TreeNode *)malloc(sizeof(TreeNode));
        t4 = (TreeNode *)malloc(sizeof(TreeNode));

        t1-> data = 1;
        t2-> data = 2;
        t3-> data = 3;
        t4-> data = 4;

        t1->left    = t2;
        t1->right  = t3;
        t2->left    = t4;

        t2->right = NULL;
        t3->left   = NULL;
        t3->right = NULL;

        printf("%d", t2 -> data); //result: 2
        printf("%d", t4 -> data); //result: 4

        return 0;
}
```



이진 트리의 구현(개인 실습#1)

- 노드 생성 함수

```
TreeNode * createTreeNode(void)
{

        //fill in the blank


}
```

HINT:
TreeNode* node를 만들어서 메모리 할당함.
node right, left = NULL로 초기화하고
node를 반환함.

이진 트리의 구현(개인 실습#1)

- 노드 데이터 삽입 및 획득 함수

```
void setData(TreeNode * node, element data)
{
        //fill in the blank
}

element getData(TreeNode * node)
{
        //fill in the blank
}
```

HINT:
setData:
매개변수로 받은node의 data에
매개변수로 받은 element data를 저장함.

getData:
매개변수로 받은 node의 data를 반환.

- 서브 트리의 루트 데이터 획득 함수

```
TreeNode * getLeftSubTree(TreeNode * node)
{
      //fill in the blank
}

TreeNode * getRightSubTree(TreeNode * node)
{
      //fill in the blank
}
```

getLeftSubTree:
매개변수로 받은 node의 left 를 반환.

getRightSubTree:
매개변수로 받은 node의 right 를 반환.

이진 트리의 구현(개인 실습#1)

- 서브 트리 연결 함수

```
void makeLeftSubTree(TreeNode * main, TreeNode * sub)
{
        //fill in the blank
}

void makeRightSubTree(TreeNode * main, TreeNode * sub)
{
        //fill in the blank
}
```

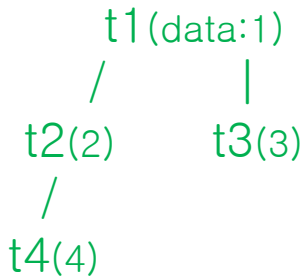
HINT:
makeLeftSubTree:
매개변수로 받은 main의 left가
이미 존재하면 수행 x
존재하지 않으면 매개변수로 받은
sub을 넣음.

makeRightSubTree:
매개변수로 받은 main의 right가
이미 존재하면 수행 x
존재하지 않으면 매개변수로 받은
sub을 넣음.

이진 트리의 구현(개인 실습#1)

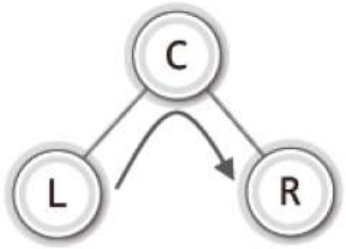
– 메인 함수

```
int main(void){
        TreeNode * t1 = createTreeNode();
        TreeNode * t2 = createTreeNode();
        TreeNode * t3 = createTreeNode();
        TreeNode * t4 = createTreeNode();

        setData(t1, 1);
        setData(t2, 2);
        setData(t3, 3);
        setData(t4, 4);

        makeLeftSubTree(t1, t2);
        makeRightSubTree(t1, t3);
        makeLeftSubTree(t2, t4);

        printf("%d \n", getData(getLeftSubTree(t1)));                    //result: 2
        printf("%d \n", getData(getLeftSubTree(getLeftSubTree(t1))));    //result: 4
        return 0;
}
```

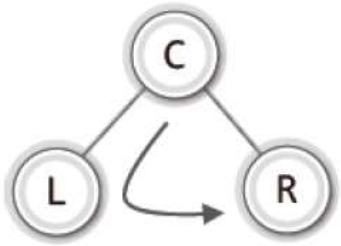


이진 트리의 순회

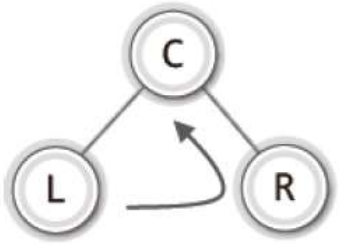
- **순회의 세 가지 방법**
  - 순회의 기준은 **루트 노드**를 방문하는 시기
  - 중위 순회 (Inorder Traversal):



  - 전위 순회 (Preorder Traversal) :



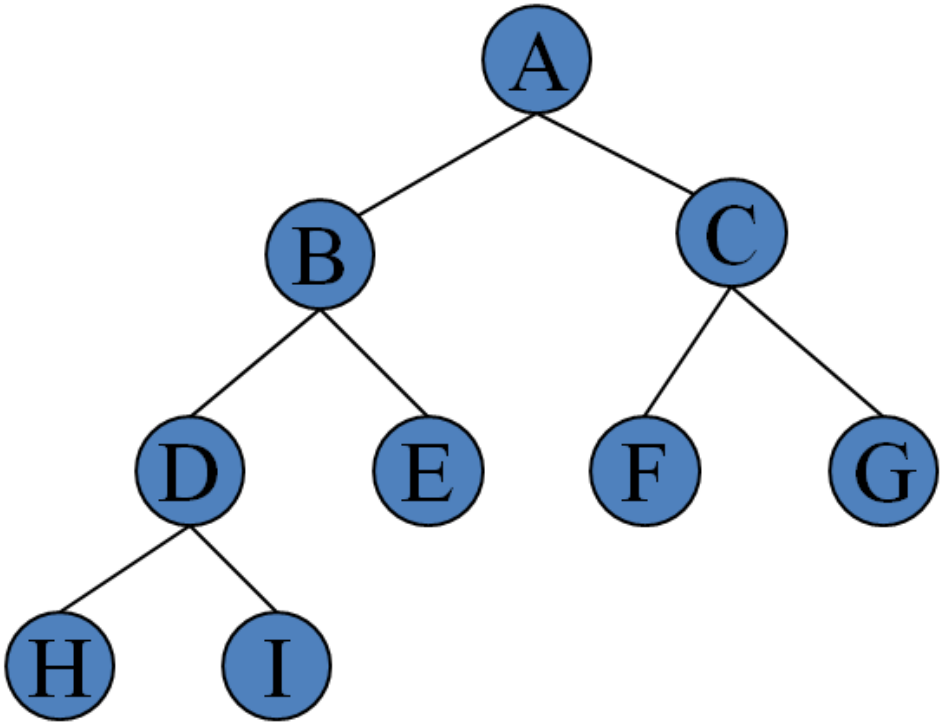
  - 후위 순회 (Postorder Traversal):



이진 트리의 순회

- **트리 순회(tree traversal)**
  - 트리에 있는 모든 노드를 한 번씩만 방문
  - 순회 방법 : LCR, LRC, CLR
    - L : 왼쪽 이동, C : 노드방문, R : 오른쪽 이동
  - 왼쪽을 오른쪽보다 먼저 방문(LR)
    - LCR : 중위(inorder) 순회
    - CLR : 전위(preorder) 순회
    - LRC : 후위(postorder) 순회

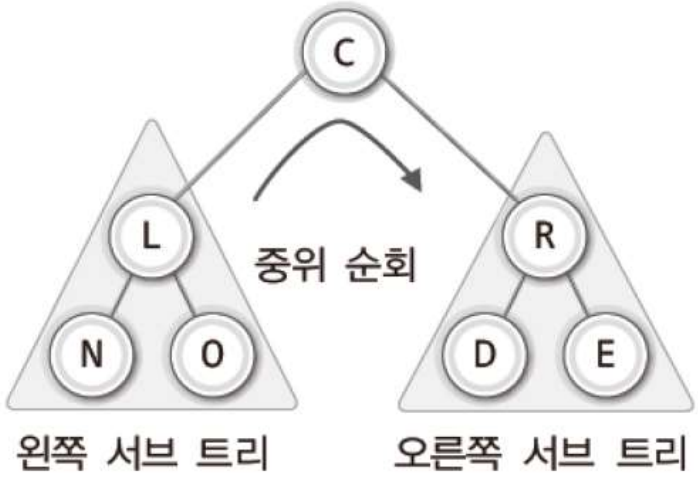
이진 트리의 순회(중위순회)



H->D->I->B->E->A->F->C->G

이진 트리의 순회

- **중위 순회의 구현**
  - 재귀를 사용하여 순서대로 모든 노드를 접근할 수 있음
  - 재귀를 통해 구현 시 다음과 같은 순서를 따름
    1. 왼쪽 서브 트리의 순회
    2. 루트 노드의 방문
    3. 오른쪽 서브 트리의 순회



중위 순회의 구현

- 중위 순회 함수

```
void Inorder(TreeNode * node)
{
        if(node == NULL) //node 가 NULL이면 재귀 탈출
                return;

        Inorder(node->left);
        printf("%d \n", node->data);
        Inorder(node->right);
}
```

중위 순회의 구현

– 메인 함수

```
int main(void)
{
        TreeNode * t1 = createTreeNode();
        TreeNode * t2 = createTreeNode();
        TreeNode * t3 = createTreeNode();
        TreeNode * t4 = createTreeNode();

        SetData(t1, 1);
        SetData(t2, 2);
        SetData(t3, 3);
        SetData(t4, 4);

        MakeLeftSubTree(t1, t2);
        MakeRightSubTree(t1, t3);
        MakeLeftSubTree(t2, t4);

        Inorder (t1);
        return 0;
}
```

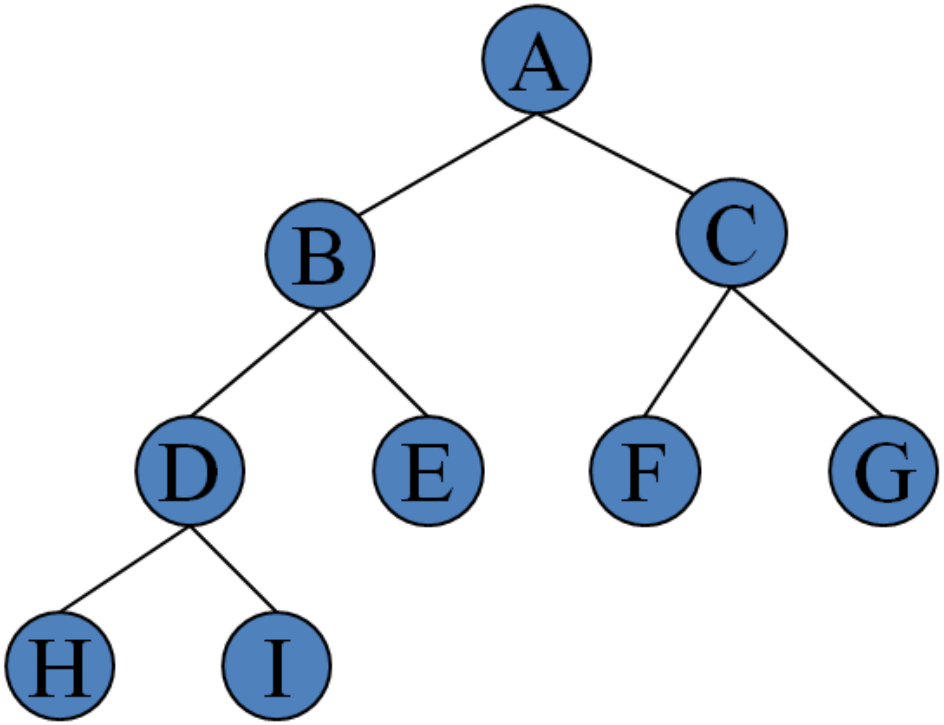
실행결과

```
4
2
1
3
```

이진 트리의 순회

- **전위 순회의 구현**
  - 재귀를 사용하여 순서대로 모든 노드를 접근할 수 있음
  - 재귀를 통해 구현 시 다음과 같은 순서를 따름
    1. 루트 노드의 방문
    2. 왼쪽 서브 트리의 순회
    3. 오른쪽 서브 트리의 순회

이진 트리의 순회(전위순회)

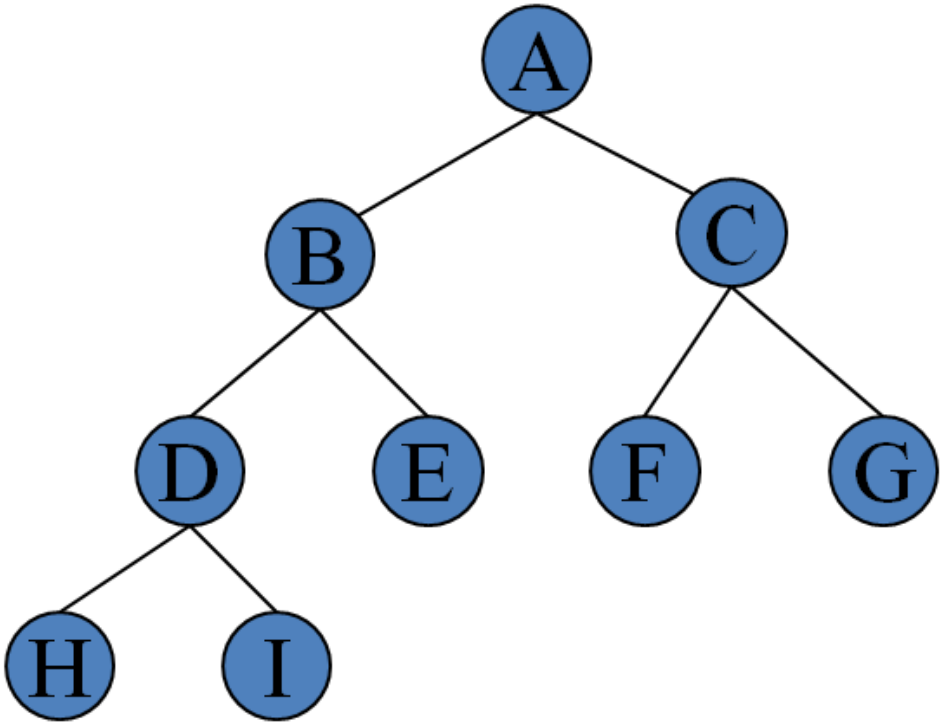


A->B->D->H->I->E->C->F->G

이진 트리의 순회

- **후위 순회의 구현**
  - 재귀를 사용하여 순서대로 모든 노드를 접근할 수 있음
  - 재귀를 통해 구현 시 다음과 같은 순서를 따름
    1. 왼쪽 서브 트리의 순회
    2. 오른쪽 서브 트리의 순회
    3. 루트 노드의 방문

이진 트리의 순회(후위순회)



H->I->D->E->B->F->G->C->A

제출

- **제출**
  - 개인실습 (#1)
    - 오늘 자정까지 제출 (~ 2020/4/17 23:59)
    - Tree
  - 과제
    - Tree Traversal(Inorder, Preorder, Postorder) 구현 (lab5.docx)
    - 다음 주 목요일 자정까지 제출 (~ 2020/4/23 23:59)

과제

- **Lab05.docx**
  - Inorder
    - 트리를 중위순회법으로 순회하며 데이터 출력
  - Preorder
    - 트리를 전위순회법으로 순회하며 데이터 출력
  - Postorder
    - 트리를 후위순회법으로 순회하며 데이터 출력

과제

- 함수
  - Tree_node* CreateNode(int key)
  - void Inorder (Tree_node* node)
  - void Preorder (Tree_node* node)
  - void Postorder (Tree_node* node)

과제

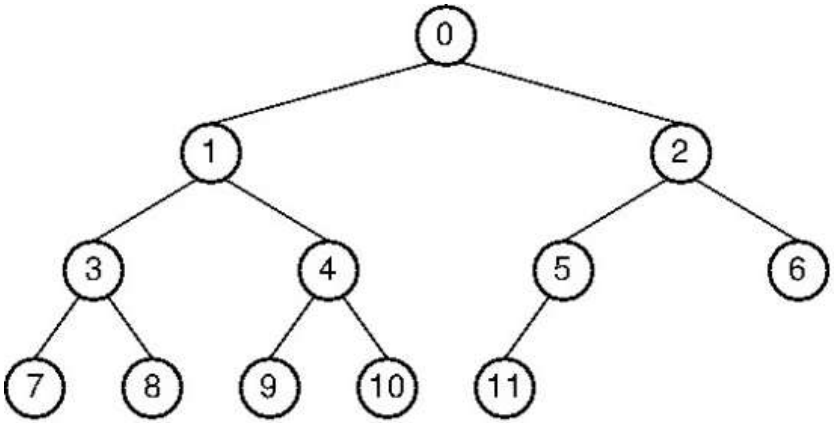
- 초기 선언

```
typedef struct Tree_node{
    int key;
    struct Tree_node *left;
    struct Tree_node *right;
}Tree_node;
```

  - Tree_node* CreateNode(int key)
    - 새로운 노드 할당
    - left, right NULL로 설정
    - parameter로 받은 key 값 저장
    - 생성된 노드 return

과제

- **프로그램**
  - 사용할 트리 구조



  - 결과

```
inorder : 7 3 8 1 9 4 10 0 11 5 2 6
preorder : 0 1 3 7 8 4 9 10 2 5 11 6
postorder : 7 8 3 9 10 4 1 11 5 6 2 0
```