트리

데이터 간 관계를 자식, 부모 관계로 표현하는 2차원적인 자료 구조. 자료의 삽입, 삭제가 빠르고 검색에 유리하다. 용량이 커져도 속도의 감소가 작아 대용량의 자료를 처리할 때 유용하다.

용어

용어	의미
노드	데이터를 가지고 있는 요소
루트	최상위 노드
자식	아래쪽 노드
부모	위쪽 노드
링크	부모, 자식 간 연결 관계
레벨	루트까지의 거리(링크 개수 + 1)
경로	특정 노드에서 다른 노드로 가는 길
차수	자식 노드의 개수
잎 노드	차수가 0인 노드

예제

tree_traverse.cpp

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

struct Node
{
    int data;
    Node *left;
    Node *right;
};
Node *Root;

void InitTree(int data)
{
    Root=(Node *)malloc(sizeof(Node));
    Root->data=data;
}

Node *AddChild(Node *Parent,int data,bool bLeft)
{
    Node *New;

    New=(Node *)malloc(sizeof(Node));
    New->data=data;
    New->left=NULL;
    New->right=NULL;
```

```
    if (bLeft) {
        Parent->left=New;
    } else {
        Parent->right=New;
    }
    return New;
}

void PreOrder(Node *R)
{
    printf("%d ",R->data);
    if (R->left) PreOrder(R->left);
    if (R->right) PreOrder(R->right);
}

void InOrder(Node *R)
{
    if (R->left) InOrder(R->left);
    printf("%d ",R->data);
    if (R->right) InOrder(R->right);
}

void PostOrder(Node *R)
{
    if (R->left) PostOrder(R->left);
    if (R->right) PostOrder(R->right);
    printf("%d ",R->data);
}

void FreeTree(Node *R)
{
    if (R->left) FreeTree(R->left);
    if (R->right) FreeTree(R->right);
    free(R);
}

void main()
{
    Node *Left,*Right;

    InitTree(1);
    Left=AddChild(Root,2,true);
    Right=AddChild(Root,3,false);
    AddChild(Left,4,true);
    AddChild(Left,5,false);
    AddChild(Right,6,true);

    PreOrder(Root);puts("");
    InOrder(Root);puts("");
    PostOrder(Root);puts("");
```

```
    FreeTree(Root);
}
```

참고

- 위키피디아
- 김상형 씨 홈페이지