Xilinx Zynq FPGA, TI DSP, MCU
기반의 프로그래밍 및 회로 설계
전문가 과정

13일차 숙제 – (자료구조)

강사 – **Innova Lee(**이상훈**)**
gcccompil3r@gmail.com

학생 – 안상재
sangjae2015@naver.com

*** 재귀함수를 사용하지 않는 이진 트리 구현**

1. 삽 입

```
void non_recur_tree_ins(tree **root, int data)
{
        tree **tmp = root;          // 재귀 함수가 아니기 때문에 노드를 이동시키기 위해 tmp라는 임시 저장 변
                                    수를 사용함. root가 이중 포인터이기 때문에 tmp도 이중포인터로 선언함.
        while(*tmp)                 // tmp가 0이 될때까지 while문 실행(트리의 밑바닥까지 가겠다는 의미)
        {
                if((*tmp)->data > data)
                        tmp = &(*tmp)->left;      // tmp를 실제 main문의 root와 동일하게 취급하기 위해, tmp
                                                  에 left 노드의 주소를 저장함.
                else if((*tmp)->data < data)
                        tmp = &(*tmp)->right;
        }
        *tmp = get_tree_node();             // 트리의 맨 밑바닥 까지 도달했으면 노드를 생성함.
        (*tmp)->data = data;
}
```

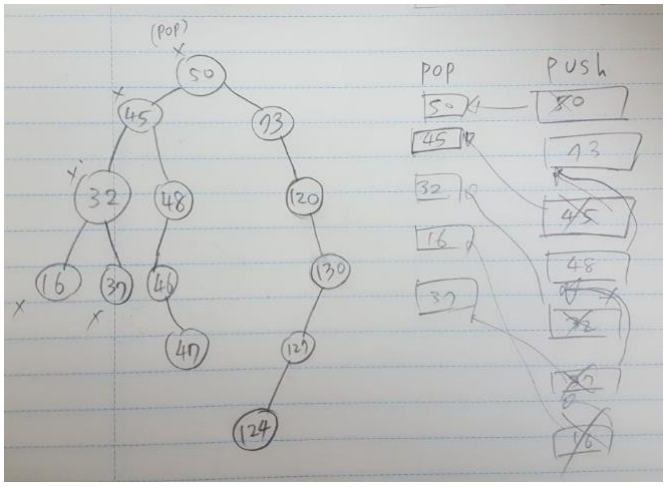
2. 삭 제

```
void non_recur_delete_tree(tree **root, int data)
{
        tree **tmp = root;          // 재귀 함수가 아니기 때문에 노드를 이동시키기 위해 tmp라는 임시 저장 변
                                    수를 사용함. root가 이중 포인터이기 때문에 tmp도 이중포인터로 선언함.
        int num;
        while(*tmp)
        {
                if((*tmp)->data > data)
                        tmp = &(*tmp)->left;              // tmp를 실제 main문의 root와 동일하게 취급하기
                                                          위해, tmp에 left 노드의 주소를 저장함.
                else if((*tmp)->data < data)
                        tmp = &(*tmp)->right;
                else if((*tmp)->left && (*tmp)->right)
                {
                        find_max(&(*tmp)->left, &num);    // 삭제하려는 노드의 왼쪽 서브 트리에서 최댓값을
                                                          찾음.
                        (*tmp)->data = num;
                        return;
                }
                else
                {
                        (*tmp) = chg_node(*tmp);          // 삭제하려는 노드의 왼쪽, 오른쪽 노드가 모두 없거나,
                                                        한 쪽만 있는 자식 노드를 삭제하려는 노드의 위치로 옮김.
```

```
                        return;
                }
        }
        printf("Not Found₩n");
}
```

3. 출 력

```
void print_tree(tree **root)
{
        tree **tmp = root;          // 이중 포인터 변수 root를 저장할 변수  tmp를 만듬.
        stack *top = NULL;          // stack 포인터 top 선언
        push(&top, *tmp);           // 처음에 트리의 루트 노드를 스택에 집어넣음.
         while(stack_is_not_empty(top)) // stack이 비어있지 않으면 while문 계속 실행
         {
        tree *t = (tree *)pop(&top);    // stack에서 노드를 하나 빼옴.
        tmp = &t;
        printf("data = %d, ", (*tmp)->data);  // 빼온 노드의 데이터 값 출력
        if((*tmp)->left)    // 왼쪽, 오른쪽 노드의 데이터값 디버깅 용 코드
                printf("left = %d, ", (*tmp)->left->data);
        else
                printf("left = NULL, ");
        if((*tmp)->right)
                printf("right = %d₩n", (*tmp)->right->data);
        else
                printf("right = NULL₩n");
        push(&top, (*tmp)->right); // 오른쪽 노드를 먼저 넣고 왼쪽 노드를 넣어서 왼쪽
                                   노드가 stack에서 pop되면, 그 다음 오른쪽 노드가 pop되게 함.
        push(&top, (*tmp)->left);
           }
}
```



출력함수 그림

=> 재귀호출의 구조가 스택의 구조와 동일함. 재귀함수로 한번 들어갈 때에는 노드 하나가 스택에 push되는 것이고, 재귀함수가 반환되는 것은 스택에 있는 노드가 pop되는 것이다.

4. 스택 상태 검사 함수

```
bool stack_is_not_empty(stack *top)          // 스택이 비어있는 상태인지 확인하는 함수
{
        if(top != NULL)
                return true;
        else
                return false;
}
```

=> bool 은 true 와 false 를 나타낼 수 있는 논리형 변수이다. 헤더 파일 stdbool.h 안에 정의 되어 있다.