TI DSP, MCU 및 Xilinx Zynq FPGA 프로그램 전문가 과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - 은태영
zero_bird@naver.com

AVL 개요

❖ AVL tree 는 균형 트리로서, 검색 시간의 단축을 목적으로 만들어졌다.

❖ 균형 트리는 왼쪽과 오른쪽의 레벨이 균형을 이루는 트리이다.

❖ 균형을 이루기 위하여 위해 기존 tree 에서 추가적으로 레벨 값 이 존재한다.

❖ 기준 노드의 하위에 존재하는 두 개의 노드는 레벨 값 은 2 이상의 차이가 없도록 한다.

❖ 두 노드 사이의 레벨 값 차이가 2 이상 날 경우, 노드를 이동시켜 균형을 맞춘다.

❖ AVL 의 한계는 검색 시간은 빠르지만, 데이터의 추가 및 삭제를 할 때 오래 걸리는 단점이 있다.

AVL_구조체

tewill@tewill-B85M-D3H: ~/my_proj

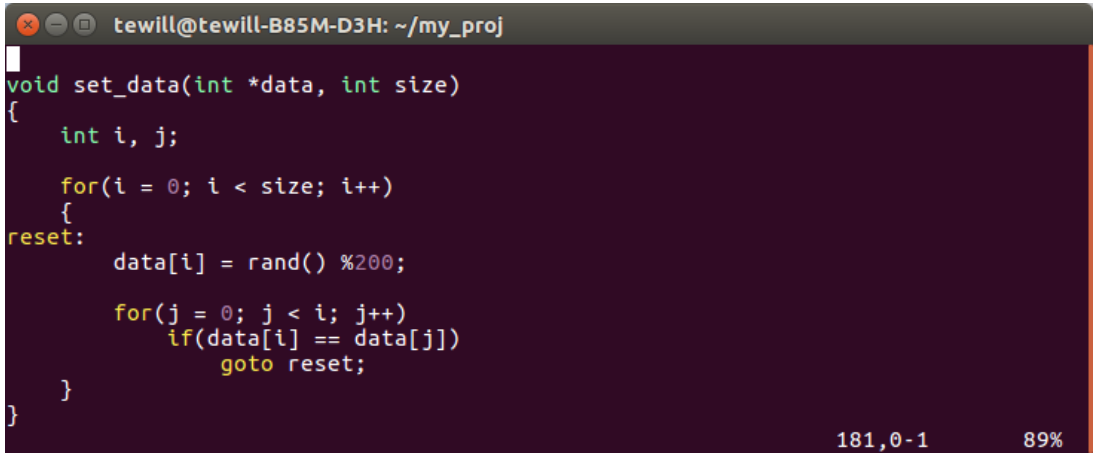
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <string.h>

enum
{
    RR, RL, LL, LR
};

struct avl_node{
    int data;
    int level;
    struct avl_node *left;
    struct avl_node *right;
};

typedef struct avl_node avl;


struct stack_node{
    avl *data;
    struct stack_node *link;
};

typedef struct stack_node stack;
                                                    1,1           Top
```

- enum 을 통하여 노드의 이동 경우의 수 4가지를 표현했다.
- 기본적으로 AVL tree 에 사용되는 노드를 정의하였다.
- 재귀 함수를 사용하지 않기 위하여, 추가적으로 스택을 만들 stack_node 를 정의하였다.

AVL

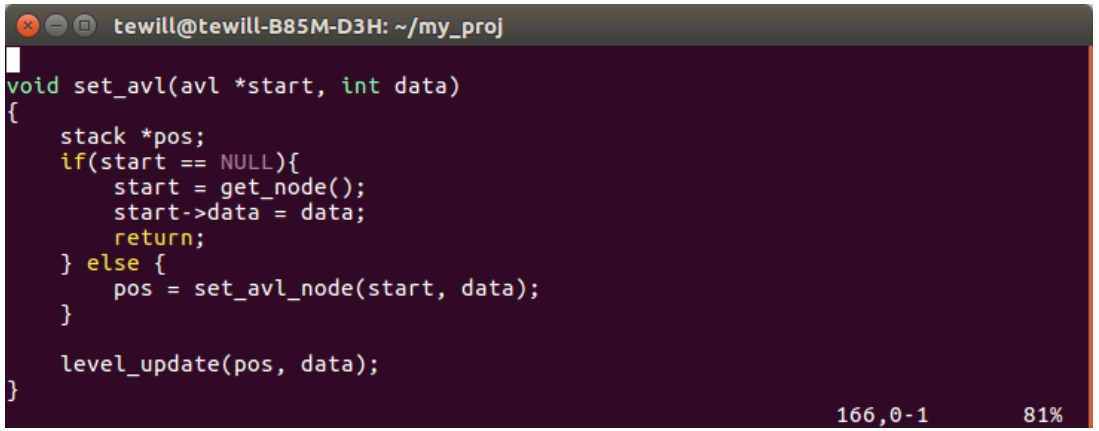
- ❖ 데이터를 받을 배열과, 배열의 길이를 저장할 size, for 문에서 사용할 I 를 정의해 준다.
- ❖ AVL 의 시작 위치를 정의하고 초기화 한다.
- ❖ 데이터를 랜덤하게 넣기 위하여 srand 를 사용하고, set_data 함수를 이용한다.
- ❖ 그 후, 해당 배열의 데이터를 for 문과 set_avl 함수를 이용하여 AVL tree 를 만든다.

AVL



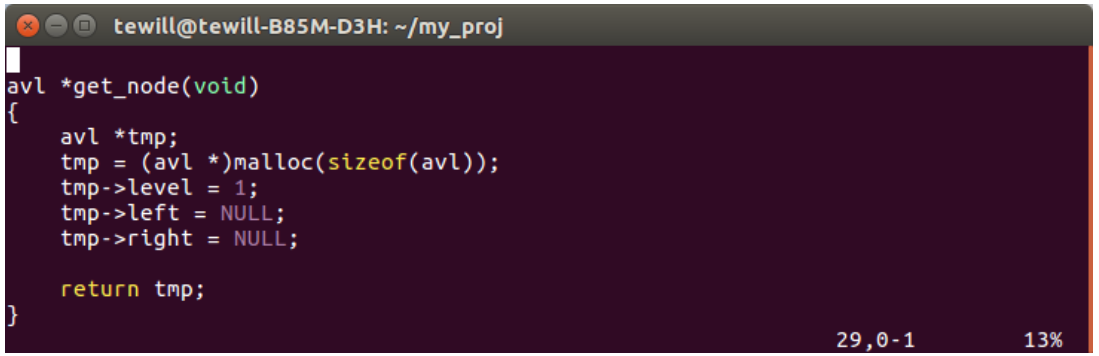
- ❖ 랜덤을 이용하여, 0~199의 숫자로 데이터 값을 집어 넣는다.
- ❖ 이때, for 문을 통하여 중복되는 값이 있는지 체크 후, 중복된 값이 있을 경우 goto 를 이용하여 값을 다시 받도록 한다.

AVL



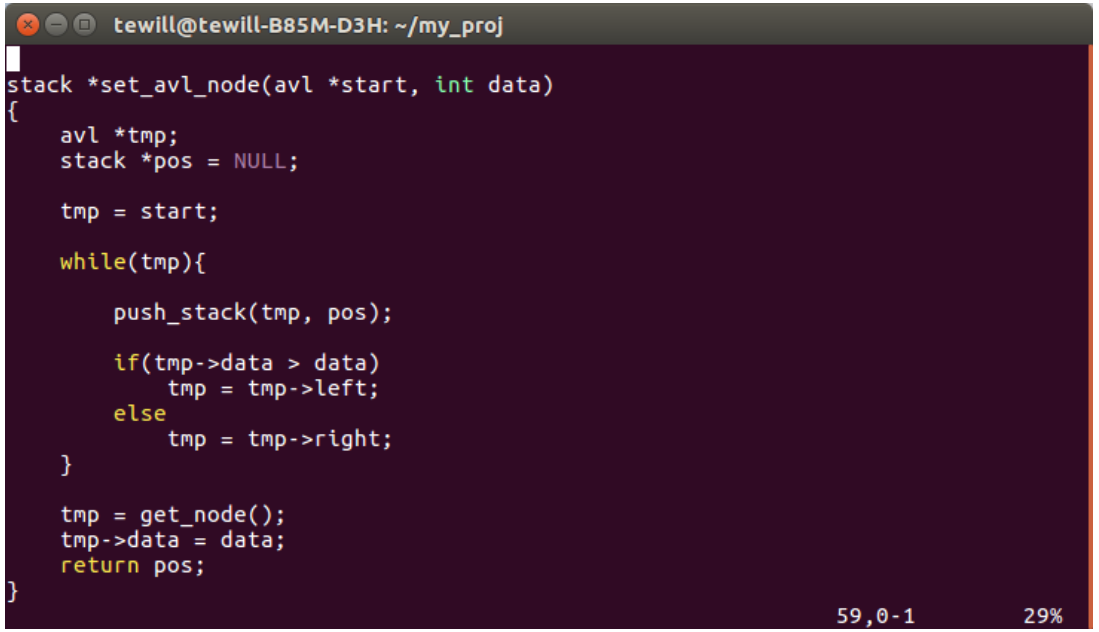
- ❖ AVL tree 를 처음 만드는지 체크한다.
- ❖ 처음일 경우 바로 get_node 를 통해 시작 주소를 입력한다.
- ❖ 그 후, data 를 입력해주고 리턴 시킨다.
- ❖ 처음 입력한 경우가 아닐 경우, set_avl_node 를 이용하여 데이터를 입력하고, 탐색한 순서, stack 의 주소를 받아온다.
- ❖ 받아온 stack을 이용하여 level_update 함수를 호출한다.

AVL



- 기본적인 AVL tree 노드의 생성 함수이다.
- malloc 을 통하여 메모리 공간을 생성 후, level, left, right 값들을 초기화 해 준다.
- 그 후, 해당 주소를 리턴한다.

AVL



- ❖ AVL tree 가 처음이 아닐 경우 호출되는 함수이다.
- ❖ AVL tree 의 시작 주소를 받아온다.
- ❖ 그 후, 반복문을 통하여 data 비교를 실시한다.
- ❖ 반복문이 시작될 때, push_stack 을 이용하여 stack 함수로 진행 순서를 저장한다.
- ❖ 비교 결과에 따라 left 나 right 로 이동한다.
- ❖ 이동 결과 tmp 가 NULL일 때 반복문을 나온다.
- ❖ 해당 공간에 get_node 를 통해 노드를 만든다.
- ❖ 진행 순서가 저장된 stack 을 리턴 한다.

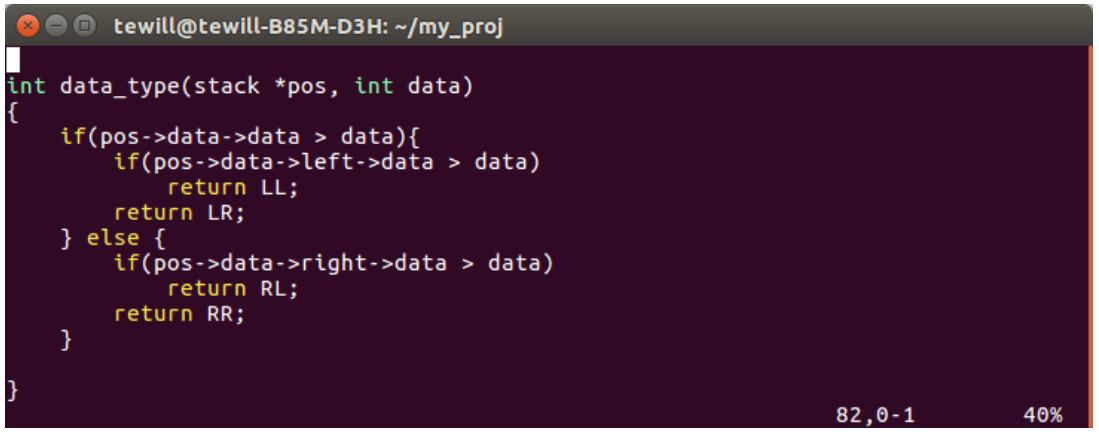
AVL

- 진행 순서를 저장하기 위한 stack 이다.
- stack 이 처음인지 아닌지를 판단하고, 결과에 따라 stack 을 올린다.

AVL

- 균형을 잡기 위한 level_update 함수이다.
- 받아온 stack 을 pop 해 가면서, 하위 노드의 레벨을 비교한다.
- 2 이상일 경우 반복문을 나와 rotation 하고, 아닐 경우 stack의 끝 부분까지 이동 후, 함수를 종료한다.
- rotation 이 끝난 다음, goto 를 이용하여 남은 stack 의 주소들을 계속 탐색한다.

AVL



- ❖ 하위 노드의 상태에 따라 노드의 이동 경우의 수는 총 4 가지 이다.
- ❖ 기준 노드에서 하위 노드의 상태를 탐색한다.
- ❖ enum 을 통해 정의한 경우의 수를 리턴한다.

AVL

tewill@tewill-B85M-D3H: ~/my_proj

```
void rotation(stack *pos, int data)
{

    avl *tmp = NULL;

    switch(data_type(pos, data))
    {
        case RL:
            tmp = pos->data->right;
            pos->data->right = tmp->left;
            tmp->left = tmp->left->right;
            pos->data->right->right = tmp;
        case RR:
            tmp = (avl *)malloc(sizeof(avl));
            memmove(tmp, pos->data, sizeof(avl));
            memmove(pos->data, pos->data->right, sizeof(avl));
            tmp->right = pos->data->left;
            pos->data->left = tmp;
            tmp->level -= 2;
            break;
        case LR:
            tmp = pos->data->left;
            pos->data->left = tmp->right;
            tmp->right = tmp->right->left;
            pos->data->left->left = tmp;
        case LL:
            tmp = (avl *)malloc(sizeof(avl));
            memmove(tmp, pos->data, sizeof(avl));
            memmove(pos->data, pos->data->left, sizeof(avl));
            tmp->left = pos->data->right;
            pos->data->right = tmp;
            tmp->level -= 2;
            break;
    }
}
                                                    97,0-1        53%
```

- ❖ switch 를 통해, 경우의 수에 따라 이동시킨다.
- ❖ RL 이나 LR 의 경우, RR 이나 LL 을 한 번 더 실행 해야 하기 때문에 break 를 없이 진행시킨다.