TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/3/8
수업일수	11 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

1.소스코드 해석

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

typedef enum __rot{
    RR,
    RL,
    LL,
    LR
} rot;

typedef struct __avl_tree{
    int lev;
    int data;
    struct __avl_tree *left;
    struct __avl_tree *right;
} avl;

bool is_dup(int *arr, int cur_idx){
    int i, tmp = arr[cur_idx];

    for(i = 0; i < cur_idx; i++)
        if(tmp == arr[i])
            return true;

    return false;
}

void init_rand_arr(int *arr, int size){
    int i;
    for(i = 0; i < size; i++){
        redo:
        //arr[i] = rand() % 15 + 1;
        arr[i] = rand() % 100 + 1;

        if(is_dup(arr, i)){
            printf("%d dup! redo rand()₩n", arr[i]);
            goto redo;
        }
    }
}

void print_arr(int *arr, int size){
    int i;

    for(i = 0; i < size; i++)
        printf("arr[%d] = %d₩n", i, arr[i]);
}
```

```
avl *get_avl_node(void){
    avl *tmp;
    tmp = (avl *)malloc(sizeof(avl));
    tmp->lev = 1;
    tmp->left = NULL;
    tmp->right = NULL;
    return tmp;
}

void print_tree(avl *root){
    if(root){
        printf("data = %d, lev = %d, ", root->data, root->lev);

        if(root->left)
            printf("left = %d, ", root->left->data);
        else
            printf("left = NULL, ");
        if(root->right)
            printf("right = %d₩n", root->right->data);
        else
            printf("right = NULL₩n");

        print_tree(root->left);
        print_tree(root->right);
    }
}
```

```
int update_level(avl *root){
    int left = root->left ? root->left->lev : 0;
    int right = root->right ? root->right->lev : 0;
    if(left > right)
        return left + 1;

    return right + 1;
}

int rotation_check(avl *root){
    int left = root->left ? root->left->lev : 0;
    int right = root->right ? root->right->lev : 0;

    return right - left;
}

int kinds_of_rot(avl *root, int data){
    printf("data = %d₩n", data);

    // for RR and RL
    if(rotation_check(root) > 1){
        if(root->right->data > data)
            return RL;

        return RR;
    }
    // for LL and LR
    else if(rotation_check(root) < -1){
```

```
            if(root->left->data < data)
                return LR;

            return LL;
        }
}

avl *rr_rot(avl *parent, avl *child){
    parent->right = child->left;
    child->left = parent;
    parent->lev = update_level(parent);
    child->lev = update_level(child);
    return child;
}

avl *ll_rot(avl *parent, avl *child){
    parent->left = child->right;
    child->right = parent;
    parent->lev = update_level(parent);
    child->lev = update_level(child);
    return child;
}

avl *rl_rot(avl *parent, avl *child){
    child = ll_rot(child, child->left);
    return rr_rot(parent, child);
}

avl *lr_rot(avl *parent, avl *child){
    child = rr_rot(child, child->right);
    return ll_rot(parent, child);
}

//void rotation(avl *root, int ret)
avl *rotation(avl *root, int ret){
    switch(ret){
        case RL:
            printf("RL Rotation₩n");
            return rl_rot(root, root->right);
        case RR:
            printf("RR Rotation₩n");
            return rr_rot(root, root->right);
        case LR:
            printf("LR Rotation₩n");
            return lr_rot(root, root->left);
        case LL:
            printf("LL Rotation₩n");
            return ll_rot(root, root->left);
    }
}

void avl_ins(avl **root, int data){
    if(!(*root)){
        (*root) = get_avl_node();
        (*root)->data = data;
        return;
```

```
    }

    if((*root)->data > data)
        avl_ins(&(*root)->left, data);
    else if((*root)->data < data)
        avl_ins(&(*root)->right, data);

    //update_level(root);
    (*root)->lev = update_level(*root);

    if(abs(rotation_check(*root)) > 1){
        printf("Insert Rotation!₩n");
        *root = rotation(*root, kinds_of_rot(*root, data));
        //rotation(*root, kinds_of_rot(*root, data));
    }
}

avl *chg_node(avl *root){
    avl *tmp = root;
    if(!root->right)
        root = root->left;
    else if(!root->left)
        root = root->right;
    free(tmp);
    return root;
}

avl *find_max(avl *root, int *data){
    if(root->right)
        root->right = find_max(root->right, data);
    else
    {
        *data = root->data;
        root = chg_node(root);
    }

    return root;
}

void avl_del(avl **root, int data){
    if(*root == NULL)
    {
        printf("There are no data that you find %d₩n", data);
        return;
    }
    else if((*root)->data > data)
        avl_del(&(*root)->left, data);
    else if((*root)->data < data)
        avl_del(&(*root)->right, data);
    else if((*root)->left && (*root)->right)
        (*root)->left = find_max((*root)->left, &(*root)->data);
    else
    {
        *root = chg_node(*root);
        return;
    }
```

```
    (*root)->lev = update_level(*root);

    if(abs(rotation_check(*root)) > 1)
    {
        printf("Delete Rotation!₩n");
        *root = rotation(*root, kinds_of_rot(*root, data));
        //rotation(*root, kinds_of_rot(*root, data));
    }
}

int main(void){
    int i;
    avl *root = NULL;
    int arr[16] = {0};
    int size = sizeof(arr) / sizeof(int) - 1;

    srand(time(NULL));
    init_rand_arr(arr, size);
    print_arr(arr, size);

    for(i = 0; i < size; i++)
        avl_ins(&root, arr[i]);

    print_tree(root);

    printf("₩nAfter Delete₩n");
    avl_del(&root, arr[3]);
    avl_del(&root, arr[6]);
    avl_del(&root, arr[9]);

    print_tree(root);

    return 0;
}
```

- 소스 해석

main함수를 보게 되면 init_rad_arr이라는 함수는 랜덤한 값을 배열arr에 넣는 역하을 하고 print_arr은 init_rad_arr에서 입력된 배열의 값들을 printf를 사용해 출력하는 역할을 한다. 그리고 다음으로 for문을 이용하여 위에서 만든 배열들을 avl_ins함수를 통해 배치하게 된다. avl_ins를 살펴보면

```
void avl_ins(avl **root, int data){
    if(!(*root)){
        (*root) = get_avl_node();
        (*root)->data = data;
        return;
    }

    if((*root)->data > data)
        avl_ins(&(*root)->left, data);
    else if((*root)->data < data)
        avl_ins(&(*root)->right, data);

    //update_level(root);
    (*root)->lev = update_level(*root);

    if(abs(rotation_check(*root)) > 1){
        printf("Insert Rotation!₩n");
        *root = rotation(*root, kinds_of_rot(*root, data));
        //rotation(*root, kinds_of_rot(*root, data));
    }
}
```

의 코드에서 첫번째 if문과 두번째 if else까지는 일반 이진트리에서 보았던 것 처럼 값을 배치하는 역할을 하고

```
(*root)->lev = update_level(*root);
```

에서는 각 데이터 공간의 레벨 값들을 관리한다.
마지막 if문 if(abs(rotation_check(*root)) > 1) 에서는 rotation을 하기 위한 체크를 하게 된다. 먼저 update_level(*root)를 살펴보면

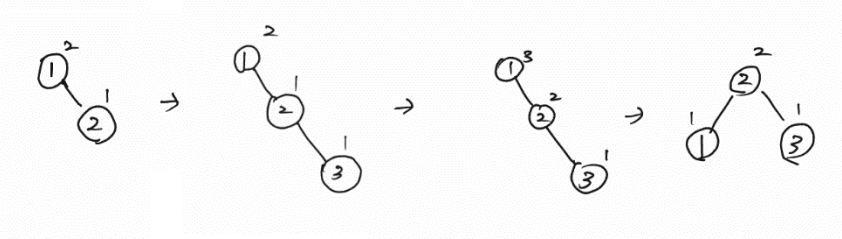
```
int update_level(avl *root){
    int left = root->left ? root->left->lev : 0;
    int right = root->right ? root->right->lev : 0;
    if(left > right)
        return left + 1;

    return right + 1;
}
```

변수 left(right)에 root의 left(right)에 값이 있는지 없는지를 판별하여 있다면 root->left->lev의 값을 변수 left(right)에 넣고 없으면 0을 넣는다. 그리고 만약 left의 값이 right보다 크다면 left+1의 값을 리턴하고 right가 left보다 크다면 right+1을 해주어 리턴한다. 이 함수는 *root->data값과 data값이 같을 때 수행하고 해당 (*root)->lev의 값을 갱신해 준다.

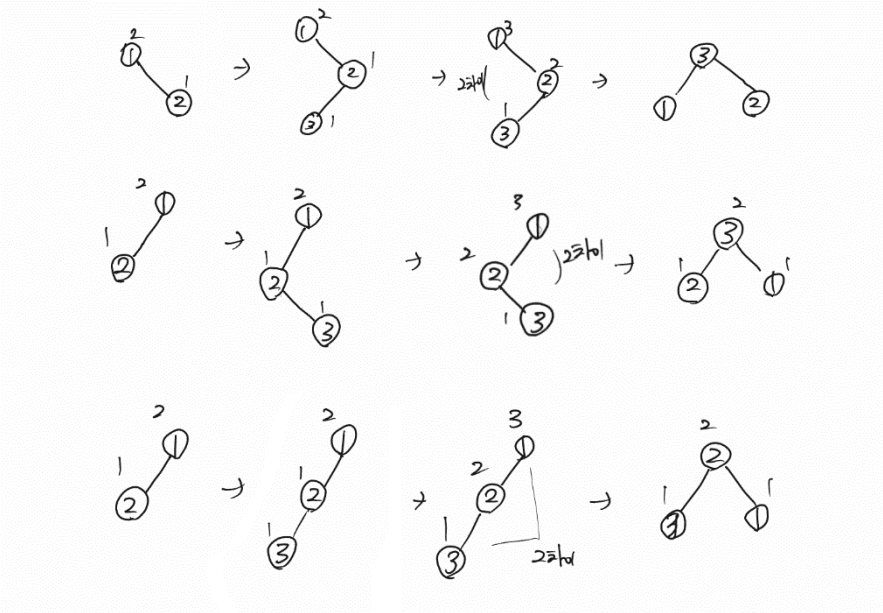
다음은 if (abs(rotation_check(*root))>1) 함수의 동작이다.
이 함수는 abs라는 함수가 rotation_check라는 함수를 인자로 사용하는데 현재 이 소스에는 abs가 나와있지 않는다. 추측하기로 root 데이터 값의 양쪽 레벨이 2이상 차이 날 때 다시 자리 배치를 하게 끔 하는 함수로 추측된다. 그리고 해당 *root에 rotation의 리턴 값을 넣게 되는데 rotation 함수의 동작은 이러하다.

```
typedef enum __rot{
    RR,
    RL,
    LL,
    LR
} rot;
```

Main 함수를 시작하기 앞서 enum을 이용해 각각에 0,1,2,3의 값을 부여한다.
그리고 이 값들은 rotation 함수에서 switch문을 이용하여 각 경우에 위의 값들을 알맞게 넣어준다.
저 RR과 RL, LR, LL은 avl tree가 이진 트리로 형태로 변환될 때 겪을 수 있는 데이터의 경우들이다.
밑의 사진은 RR에 해당하는 형태의 데이터이다.



1번을 기준으로 왼쪽과 오른쪽의 레벨 값을 확인하여 왼쪽은 값이 없으니 0, 오른쪽은 레벨 값이 1이 되어 레벨 차가 1이지만 3번이 2의 아래로 들어오게 되면서 1번과 2번의 레벨 값이 증가하게 되는데 그리하면 1번의 양 쪽 레벨의 차이가 2가 되어 이진 트리로 구성하게 위치가 바뀐다.
위의 그림 외에도 아래 그림처럼 3가지의 경우가 더 있는데 이름 RR, RL, LR, LL이라고 이름을 붙여 각 경우일 때 마다 소스코드를 다르게 적용한다.



```
avl *rotation(avl *root, int ret){
    switch(ret){
        case RL:
            printf("RL Rotation₩n");
            return rl_rot(root, root->right);
        case RR:
            printf("RR Rotation₩n");
            return rr_rot(root, root->right);
        case LR:
            printf("LR Rotation₩n");
            return lr_rot(root, root->left);
        case LL:
            printf("LL Rotation₩n");
            return ll_rot(root, root->left);
    }
}
```

rotation함수에서는 kinds_of_rot이라는 함수에서 반환된 리턴값을 인자로 사용하게 된다. 아래 kinds_of_rot를 보게 되면 rotation_check 에서부터 리턴된 right – left의 값이 1일 때 와 -1일 때를 구분하고 입력된 데이터와 그의 오른쪽(왼쪽)에 있는 데이터 값을 비교하여 right의 값이 커 레벨 차가 양수이고 오른쪽 데이터가 입력된 데이터보다 크면 RL, 작으면 RR이다.
반대로 right – left가 음수일 때 데이터의 왼쪽 데이터가 입력된 데이터보다 값이 크면 LL, 작으면 LR이 리턴되어 각 리턴값이 해당되는 문장을 수행한다.

```
int kinds_of_rot(avl *root, int data){
    printf("data = %d₩n", data);

    // for RR and RL
    if(rotation_check(root) > 1){
        if(root->right->data > data)
            return RL;

        return RR;
    }
    // for LL and LR
    else if(rotation_check(root) < -1){
        if(root->left->data < data)
            return LR;

        return LL;
    }
}
```

```
avl *rr_rot(avl *parent, avl *child){
    parent->right = child->left;
    child->left = parent;
    parent->lev = update_level(parent);
    child->lev = update_level(child);
    return child;
}
```

case문으로 들어가 수행되는 함수들이 다를 텐데 그 중 하나만을 분석해 보자면 avl *rr_rot(avl *parent, avl *child)라는 함수는 (root, root->right)를 인자로 보내고 parent와 child라고 받았다. Parent->right에 child->left값을 대입하고 child->left에 parent의 값을 넣는다. Parent->lev와 child->lev 모두 update_level함수의 리턴 값을 레벨 값에 적용하고 child값을 반환하면 이 child 값이 rotation함수를 불렀던 avl_ins의 root 값에 들어가게 된다.