Xilinx Zynq FPGA,TI DSP,
MCU 기반의
프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 정한별
hanbulkr@gmail.com

<avl_ins 재귀함수 없이 하기>

```
#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;

typedef struct __stack
{
	void *data;
	struct __stack *link;
} stack;

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;
}

stack *get_stack_node(void)
{
        stack *tmp;
        tmp = (stack *)malloc(sizeof(stack));
        tmp->link = NULL;
        return tmp;
}

void *pop(stack **top)
{
        stack *tmp = *top;
        void *data = NULL;

        if(*top == NULL)
        {
                printf("stack is empty!\n");
                return NULL;
        }

        data = (*top)->data;
```

```
	*top = (*top)->link;
	free(tmp);

	//return (*top)->data;
	return data;
}

void push(stack **top, void *data)
{
	if(data == NULL)
		return;

	stack *tmp = *top;
	*top = get_stack_node();
	(*top)->data = malloc(sizeof(void *));
	(*top)->data = data;
	(*top)->link = tmp;
}

bool stack_is_not_empty(stack *top)
{
	if(top != NULL)
		return true;
	else
		return false;
}

void print_tree(avl **root)
{
	avl **tmp = root;
	stack *top = NULL;

	push(&top, *tmp);

	while(stack_is_not_empty(top))
	{
		avl *t = (avl *)pop(&top);
		tmp = &t;

		printf("data = %d, lev = %d, ", (*tmp)->data, (*tmp)->lev);

		if((*tmp)->left)
			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);
                push(&top, (*tmp)->left);
        }
}

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)
{
        // for RR and RL
        //if(rotation_check(root) > 1)
        if(rotation_check(root) > 1)
        {
                //if(root->right->data > data)
                if(rotation_check(root->right) < 0)
                        return RL;
                return RR;
        }
        // for LL and LR
        //else if(rotation_check(root) > 1)
        else if(rotation_check(root) < -1)
        {
                //if(root->left->data < data)
                if(rotation_check(root->left) > 0)
                        return LR;
                return LL;
        }
}

avl *rr_rot(avl *parent, avl *child)
```

```
{
	//parent->right = child->left ? child->left : child->right;
	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 : child->left;
	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);
	//child = ll_rot(child, child->left);
	return rr_rot(parent, child);
}

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

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)
{
		int cnt = 0;
		avl **tmp = root;
		stack *top = NULL;
		//push(&top, *tmp);
		while(*tmp)
		{
				printf("Save Stack: %d, data = %d\n", ++cnt, data);
				//push(&top, *tmp);
				push(&top, tmp);

				if((*tmp)->data > data)
						tmp = &(*tmp)->left;
				else if((*tmp)->data < data)
						tmp = &(*tmp)->right;
		}

		*tmp = get_avl_node();
		(*tmp)->data = data;

		while(stack_is_not_empty(top))
		{
				printf("Extract Stack: %d, data = %d\n", --cnt, data);
				avl **t = (avl **)pop(&top);
				(*t)->lev = update_level(*t);
				if(abs(rotation_check(*t)) > 1)
				{
						printf("Insert Rotation\n");
						// Need to change here with pointer of pointer
						//*tmp = rotation(*tmp, kinds_of_rot(*tmp, data));
						//*root = rotation(*tmp, kinds_of_rot(*tmp, data));
						/* It's just same as else. */
#if 0
						if((*root) == (*t))
								*root = rotation(*t, kinds_of_rot(*t, data));
						else
								*t = rotation(*t, kinds_of_rot(*t, data));
#endif
						*t = rotation(*t, kinds_of_rot(*t, data));
				}
		}

#if 0
		//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), data);
    }
#endif
}

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;
}

#if 0
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;
}
#endif

void find_max(avl **root, int *data)
{
    avl **tmp = root;

    while(*tmp)
    {
        if((*tmp)->right)
            tmp = &(*tmp)->right;
        else
        {
            *data = (*tmp)->data;
            *tmp = chg_node(*tmp);
            break;
        }
    }
}

void avl_del(avl **root, int data)
{
    int cnt = 0, num, i;
    avl **tmp = root;
    stack *top = NULL;

    while(*tmp)
    {
        printf("Save Stack: %d, data = %d\n", ++cnt, data);
        //printf("tmp = 0x%x, data = %d\n", tmp, (*tmp)->data);
        //push(&top, *tmp);
        push(&top, tmp);

        if((*tmp)->data > data)
            tmp = &(*tmp)->left;
        else if((*tmp)->data < data)
            tmp = &(*tmp)->right;
        else if((*tmp)->left && (*tmp)->right)
        {
            find_max(&(*tmp)->left, &num);
            (*tmp)->data = num;
            goto lets_rot;
        }
        else
        {
            int counter = cnt;

            (*tmp) = chg_node(*tmp);

            for(i = 0; i < counter; i++)
            {
                printf("Extract Stack: %d, data = %d\n", --cnt, data);
                pop(&top);
            }
            //goto lets_rot;
            return;
        }
    }

    if(*tmp == NULL)
    {
        printf("There are no data that you find %d\n", data);
```

```
		for(i = 0; i < cnt; i++)
		{
			printf("Extract Stack: %d, data = %d\n", --cnt, data);
			pop(&top);
		}

		return;
	}

lets_rot:
	while(stack_is_not_empty(top))
	{
		avl **t = (avl **)pop(&top);
		printf("Extract Stack: %d, data = %d\n", --cnt, data);
		//printf("*t = 0x%x, data = %d\n", *t, (*t)->data);

		(*t)->lev = update_level(*t);

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

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

	//int data[] = {100, 50, 200, 25, 75, 80};
	int data[] = {100, 50, 200, 25, 75, 70};

	srand(time(NULL));

	init_rand_arr(arr, size);
	print_arr(arr, size);

#if 1
	for(i = 0; i < size; i++)
		avl_ins(&root, arr[i]);

	print_tree(&root);
#endif
```

```
#if 1
        printf("\nAfter Delete\n");
        avl_del(&root, arr[3]);
        avl_del(&root, arr[6]);
        avl_del(&root, arr[9]);

        print_tree(&root);
#endif

#if 0
        printf("\nDebug AVL\n");

        for(i = 0; i < 6; i++)
        {
                avl_ins(&test, data[i]);
                print_tree(&test);
        }

        printf("\nFinal Result\n");
        print_tree(&test);
#endif

        return 0;
}
```