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

#12

강사 : Innova Lee(이 상훈)

학생 : 김 시윤

1. 배운내용 복습

RB tree

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

#define		BLACK 0
#define		RED		1

typedef struct __rb_node
{
	int data;
	int color;

	struct __rb_node *left;
	struct __rb_node *right;
	struct __rb_node *parent;
} rb_node;

typedef struct __rb_tree
{
	struct __rb_node *root;
	struct __rb_node *nil;
} rb_tree;
```

```
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() % 200 + 1;

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

```
void rb_left_rotate(rb_tree **tree, rb_node *x)
{
        rb_node *y;
        rb_node *nil = (*tree)->nil;

        y = x->right;
        x->right = y->left;

        if(y->left != nil)
                y->left->parent = x;

        y->parent = x->parent;

        if(x == x->parent->left)
                x->parent->left = y;
        else
                x->parent->right = y;

        y->left = x;
        x->parent = y;
}

void rb_right_rotate(rb_tree **tree, rb_node *y)
{
        rb_node *x;
        rb_node *nil = (*tree)->nil;

        x = y->left;
        y->left = x->right;

        if(nil != x->right)
                x->right->parent = y;

        x->parent = y->parent;

        if(y->parent->left == y)
                y->parent->left = x;
        else
                y->parent->right = x;

        x->right = y;
        y->parent = x;
}

void rb_tree_ins_helper(rb_tree **tree, rb_node *z)
{
        rb_node *x;
        rb_node *y;
        rb_node *nil = (*tree)->nil;

        z->left = z->right = nil;
        y = (*tree)->root;
        x = (*tree)->root->left;

        while(x != nil)
        {
                y = x;

                if(x->data > z->data)
                        x = x->left;
```

```
                else
                        x = x->right;
        }

        z->parent = y;

        if(((*tree)->root == y) || (y->data > z->data))
                y->left = z;
        else
                y->right = z;
}

rb_node *rb_tree_ins(rb_tree **tree, int data)
{
        rb_node *x;
        rb_node *y;
        rb_node *tmp;

        x = (rb_node *)malloc(sizeof(rb_node));
        x->data = data;

        rb_tree_ins_helper(tree, x);

        tmp = x;
        x->color = RED;

        while(x->parent->color)
        {
                if(x->parent == x->parent->parent->left)
                {
                        y = x->parent->parent->right;

                        if(y->color)
                        {
                                x->parent->color = BLACK;
                                y->color = BLACK;
                                x->parent->parent->color = RED;
                                x = x->parent->parent;
                        }
                        else
                        {
                                if(x->parent->right == x)
                                {
                                        x = x->parent;
                                        rb_left_rotate(tree, x);
                                }

                                x->parent->color = BLACK;
                                x->parent->parent->color = RED;

                                rb_right_rotate(tree, x->parent->parent);
                        }
                }
                else
                {
                        y = x->parent->parent->left;

                        if(y->color)
                        {
                                x->parent->color = BLACK;
```

```
                        y->color = BLACK;
                        x->parent->parent->color = RED;
                        x = x->parent->parent;
                }
                else
                {
                        if(x->parent->left == x)
                        {
                                x = x->parent;
                                rb_right_rotate(tree, x);
                        }

                        x->parent->color = BLACK;
                        x->parent->parent->color = RED;

                        rb_left_rotate(tree, x->parent->parent);
                }
        }
    }

    (*tree)->root->left->color = BLACK;

    return tmp;
}

rb_tree *rb_tree_create(void)
{
    rb_tree *rbt;
    rb_node *tmp;

    rbt = (rb_tree *)malloc(sizeof(rb_tree));

    tmp = rbt->nil = (rb_node *)malloc(sizeof(rb_node));
    tmp->parent = tmp->left = tmp->right = tmp;
    tmp->color = BLACK;
    tmp->data = 0;

    tmp = rbt->root = (rb_node *)malloc(sizeof(rb_node));
    tmp->parent = tmp->left = tmp->right = rbt->nil;
    tmp->color = BLACK;
    tmp->data = 0;

    return rbt;
}

void rb_tree_preorder_print(rb_tree *tree, rb_node *x)
{
    rb_node *nil = tree->nil;
    rb_node *root = tree->root;

    if(x != tree->nil)
    {
        printf("data = %4i, ", x->data);

        if(x->left == nil)
            printf("left = NULL, ");
        else
            printf("left = %4i, ", x->left->data);

        if(x->right == nil)
```

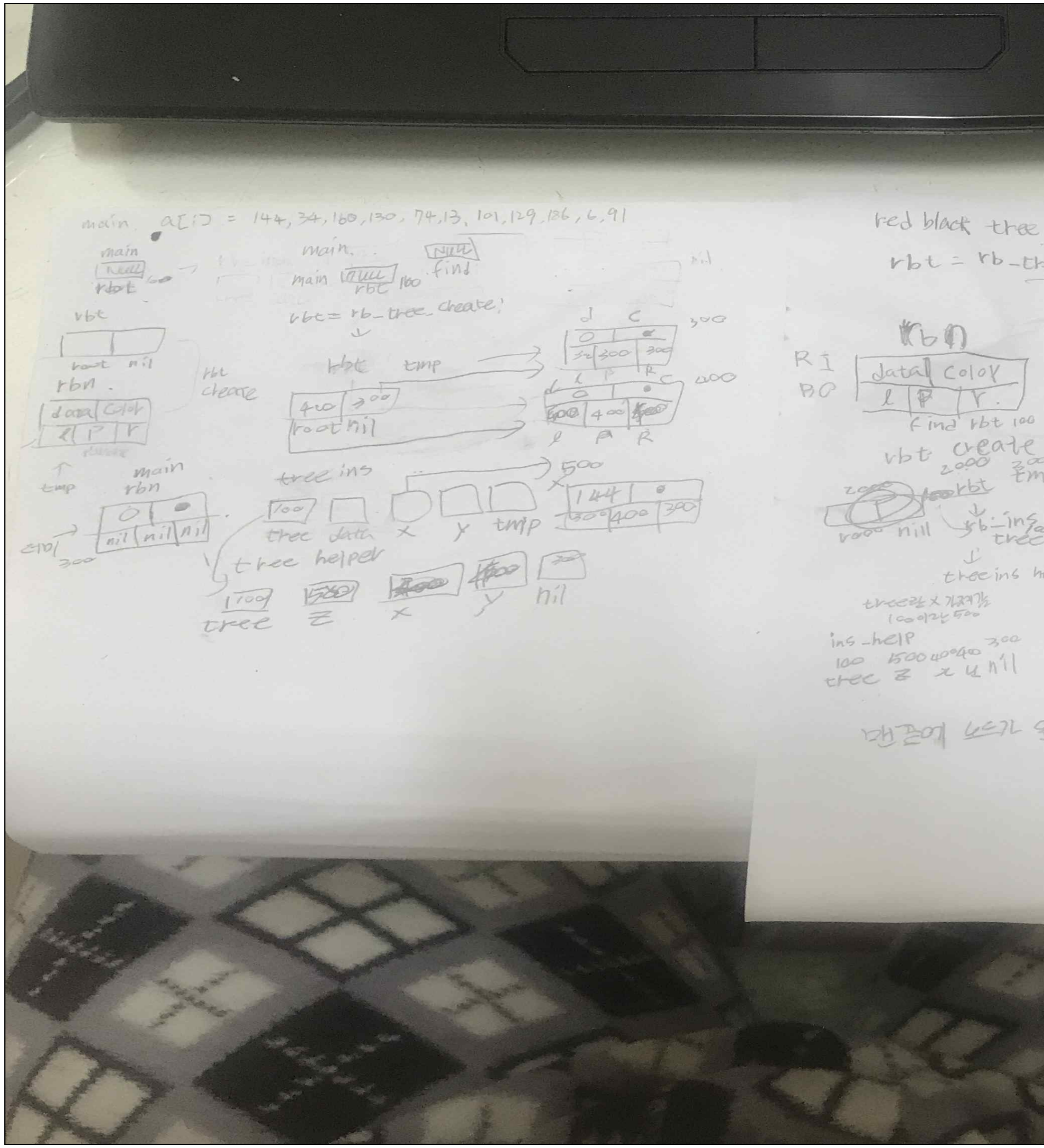
```
                printf("right = NULL, ");
        else
                printf("right = %4i, ", x->right->data);

        printf("color = %4i₩n", x->color);

        rb_tree_preorder_print(tree, x->left);
        rb_tree_preorder_print(tree, x->right);
    }
}

void rb_tree_print(rb_tree *tree)
{
    rb_tree_preorder_print(tree, tree->root->left);
}

int data_test(int n1, int n2)
{
    if(n1 > n2)
        return 1;
    else if(n1 < n2)
        return -1;
    else
        return 0;
}

rb_node *rb_tree_find(rb_tree *tree, int data)
{
    int tmp;
    rb_node *x = tree->root->left;
    rb_node *nil = tree->nil;

    if(x == nil)
        return 0;

    tmp = data_test(x->data, data);

    while(tmp != 0)
    {
        if(x->data > data)
            x = x->left;
        else
            x = x->right;

        if(x == nil)
            return 0;

        tmp = data_test(x->data, data);
    }

    return x;
}

rb_node *rb_tree_successor(rb_tree *tree, rb_node *x)
{
    rb_node *y;
    rb_node *nil = tree->nil;
    rb_node *root = tree->root;
```

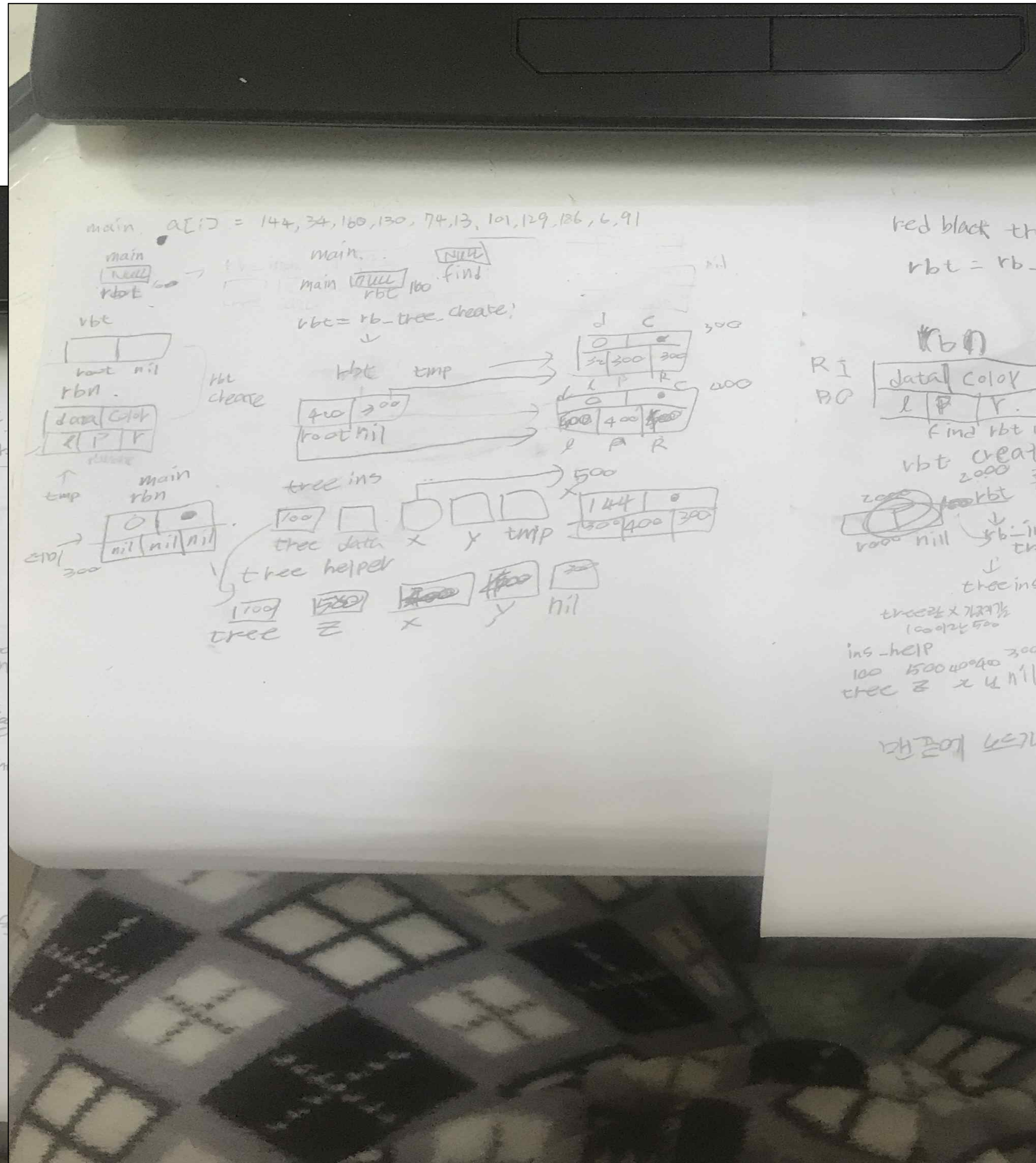
```
        if(nil != (y = x->right))
        {
                while(y->left != nil)
                        y = y->left;

                return y;
        }
        else
        {
                y = x->parent;

                while(y->right == x)
                {
                        x = y;
                        y = y->parent;
                }

                if(y == root)
                        return nil;

                return y;
        }
}

void rb_tree_del_fixup(rb_tree *tree, rb_node *x)
{
        rb_node *root = tree->root->left;
        rb_node *w;

        while((!x->color) && (root != x))
        {
                if(x->parent->left == x)
                {
                        w = x->parent->right;

                        if(w->color)
                        {
                                w->color = BLACK;
                                x->parent->color = RED;
                                rb_left_rotate(&tree, x->parent);
                                w = x->parent->right;
                        }

                        if((!w->right->color) && (!w->left->color))
                        {
                                w->color = RED;
                                x = x->parent;
                        }
                        else
                        {
                                if(!w->right->color)
                                {
                                        w->left->color = BLACK;
                                        w->color = RED;
                                        rb_right_rotate(&tree, w);
                                        w = x->parent->right;
                                }

                                w->color = x->parent->color;
                                x->parent->color = BLACK;
```

```
                w->right->color = BLACK;
                rb_right_rotate(&tree, x->parent);
                x = root;
        }
}
else
{
        w = x->parent->left;

        if(w->color)
        {
                w->color = BLACK;
                x->parent->color = 1;
                rb_right_rotate(&tree, x->parent);
                w = x->parent->left;
        }

        if((!w->right->color) && (!w->left->color))
        {
                w->color = RED;
                x = x->parent;
        }
        else
        {
                if((!w->right->color) && (!w->left->color))
                {
                        w->right->color = BLACK;
                        w->color = RED;
                        rb_left_rotate(&tree, w);
                        w = x->parent->left;
                }

                w->color = x->parent->color;
                x->parent->color = BLACK;
                w->left->color = BLACK;
                rb_right_rotate(&tree, x->parent);
                x = root;
        }
    }
  }

  x->color = BLACK;
}

void rb_tree_del(rb_tree *tree, rb_node *z)
{
        rb_node *y;
        rb_node *x;
        rb_node *nil = tree->nil;
        rb_node *root = tree->root;

        y = ((z->left == nil) || (z->right == nil)) ?
                        z : rb_tree_successor(tree, z);
        x = (y->left == nil) ? y->right : y->left;

        if(root == (x->parent = y->parent))
                root->left = x;
        else
        {
                if(y == y->parent->left)
```

```
                y->parent->left = x;
        else
                y->parent->right = x;
}

if(y != z)
{
        if(!(y->color))
                rb_tree_del_fixup(tree, x);

        y->left = z->left;
        y->right = z->right;
        y->parent = z->parent;
        y->color = z->color;
        z->left->parent = z->right->parent = y;

        if(z->parent->left == z)
                z->parent->left = y;
        else
                z->parent->right = y;

        free(z);
}
else
{
        if(!(y->color))
                rb_tree_del_fixup(tree, x);

        free(y);
}
}

int main(void)
{
        int i, size;
        int data[21] = {0};

        rb_tree *rbt = NULL;
        rb_node *find = NULL;

        srand(time(NULL));

        size = sizeof(data) / sizeof(int) - 1;

        init_rand_arr(data, size);

        rbt = rb_tree_create();

        for(i = 0; i < size; i++)
                rb_tree_ins(&rbt, data[i]);

        rb_tree_print(rbt);

        find = rb_tree_find(rbt, data[5]);

        rb_tree_del(rbt, find);
        printf("₩nAfter Delete₩n");

        rb_tree_print(rbt);
```

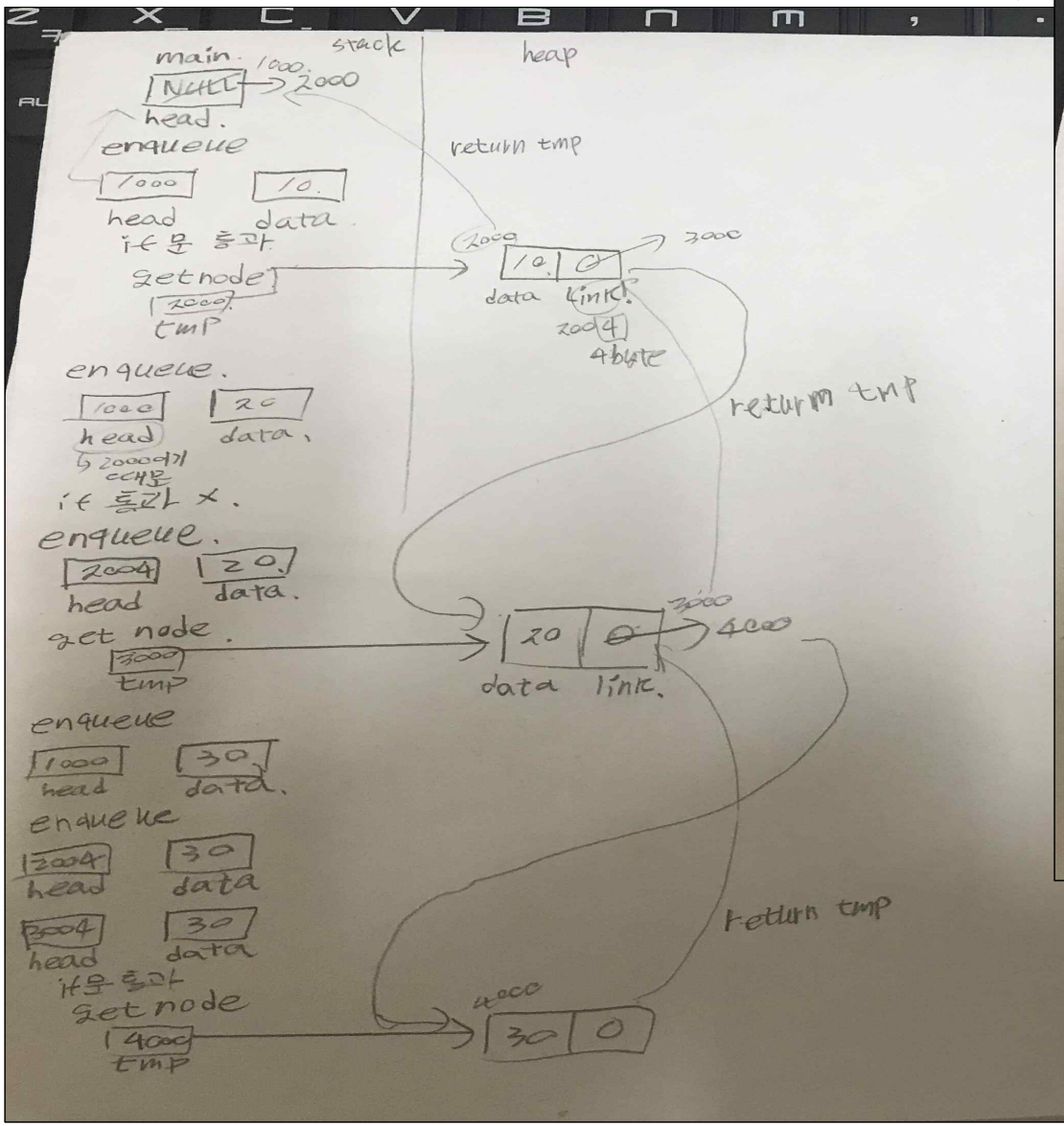
```
        return 0;
}
```

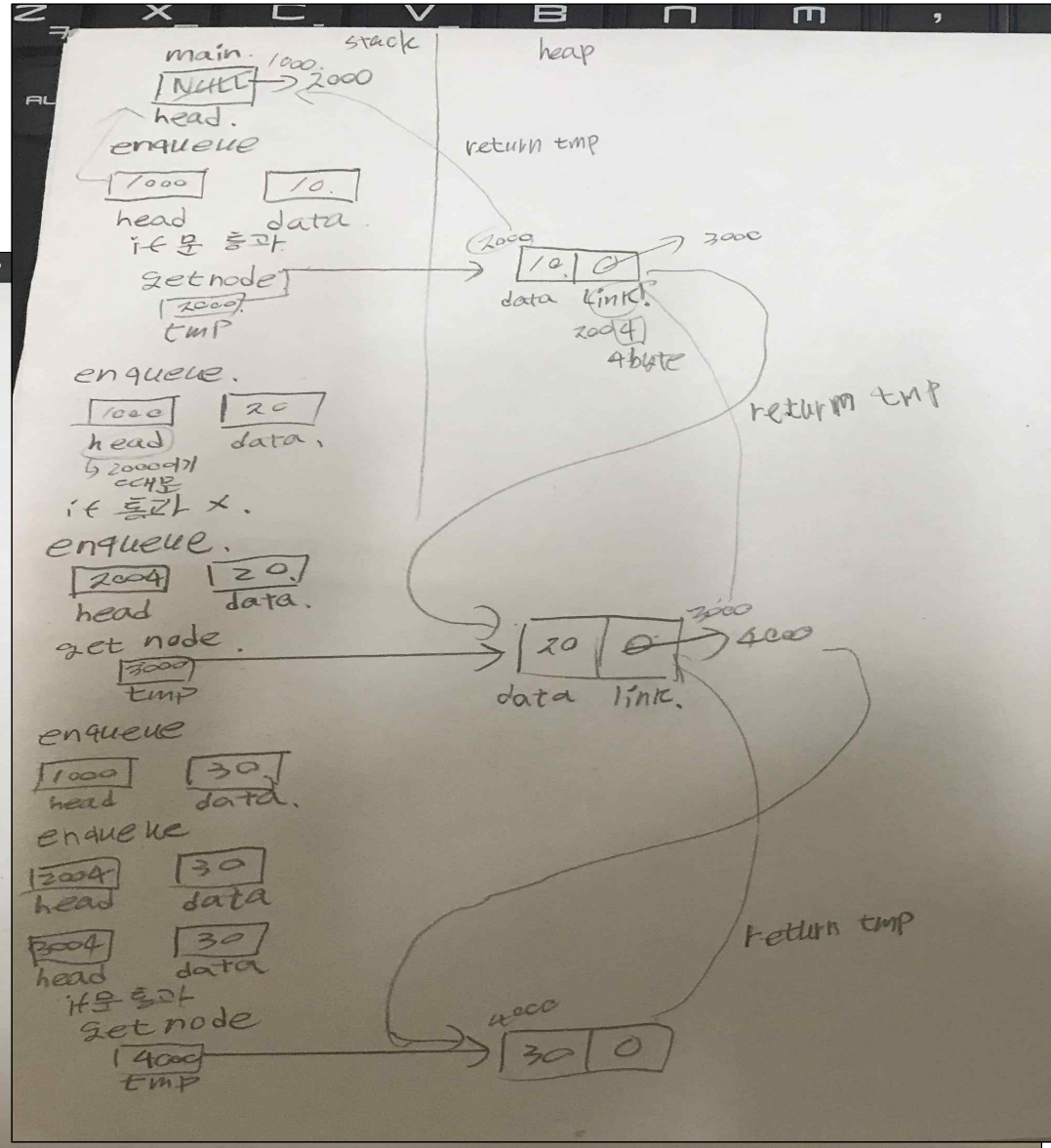




rb트리 정확한 구조를 이해하지못해 순서를 따라가지 못했다..

Queue , Delete Queue 복습





사전평가

1번

단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.

```
대소문자를 입력하시오  : a
입력한 값 소문자 -> 대문자 : A siyun@siyun-CR62-6M:~$ vi 1.c
siyun@siyun-CR62-6M:~$ gcc 1.c
siyun@siyun-CR62-6M:~$ ./a.out
대소문자를 입력하시오  : d
입력한 값 소문자 -> 대문자 : D
siyun@siyun-CR62-6M:~$ cat 1.c
#include <stdio.h>

char alpabet_converter(char al)
{

        char res;
        char res2;
        int a=al;
        if(a>=65 && a<=90){
        res=al+32;
        printf("입력한 값 대문자 ->소문자 : %c ",res);
        }
        else if(a>=97 && a<=122)
        {

        res2=al-32;
        printf("입력한 값 소문자 -> 대문자 : %c ",res2);

        }


        return 0;
}


int main(void)
{

        char al;
        char all;

        printf("대소문자를 입력하시오  : ");
        scanf("%c",&al);
        alpabet_converter(al);
        printf("\n");
return 0;
}
```

2.Stack 및 Queue 외에 Tree 라는 자료구조가 있다.
이 중에서 Tree 는 Stack 이나 Queue 와는 다르게 어떠한 이점이 있는가 ?

stack,queue 는 데이터를 일렬로 나열하지만 tree는 데이터를 나무형태로 나열하기 때문에 검색속도가 더 빠르다는 이점이 있다.

3.임의의 값 x가 있는데, 이를 4096 단위로 정렬하고 싶다면 어떻게 해야할까 ?
문제를 이해못함

4. int p[7] 와 int (*p)[7] 가 있다.
이 둘의 차이점에 대해 기술하시오.

int p[7] 은 7개의 값을 저장할수 있는 배열이고
int (*p)[7]은 int[7] (*p)로 7개있다는뜻이다.

5. 행렬이 있다고 가정하고
123 123
123 행렬을 149 형태로 만드시오
함수 supply를 사용

행렬의 덧셈을 이용한 함수를 작성하였다.

```c
#include <stdio.h>

void supply(int (*p)[3],int (*q)[3]){
        int i,j;
        for(i=0;i<2;i++){
        for(j=0;j<3;j++)
        {       printf("%d",p[i][j]);

        }
        printf("\n");
        }
        printf("\n");
for(i=0;i<2;i++){
        for(j=0;j<3;j++)
        {
                printf("%d",q[i][j]);
        }
        printf("\n");
        }
        printf("\n");
for(i=0;i<2;i++){
        for(j=0;j<3;j++)
        {       printf("%d",p[i][j]+q[i][j]);
        }
        printf("\n");
        }

}

int main(void)
{
        int a[2][3]={
                {1,2,3},{1,2,3}};
        int b[2][3]={{0,0,0},{0,2,6}};
        supply(a,b);
        return 0;
}
~
```

7. stack

9.
C 언어에서 중요시하는 메모리 구조에 대해 기술하시오.
(힌트: Stack, Heap, Data, Text 에 대해 기술하시오.)

stack 은 임시메모리로 레지스터에서 저장하기 어려운 값들을 임시로 저장해준다.

10.파이프라인이 깨지는 경우에 대해 기술하시오.
call 이나 jmp를 많이하게되면 쓸데없이 클럭을 버려야한다.
정해진 클럭에 너무 많은 call 이나 jmp 를하게 되면 깨진다.
11.
void (* signal(int signum, void (* handler)(int)))(int)라는 signal 함수의 프로토타입을 기술하시오.
프로토타입을 기술하라는 의미는 반환형(리턴 타입)과 함수의 이름, 그리고 인자(파라메터)가 무엇인지 기술하라는 뜻임.
void (* signal(int signum, void (* handler)(int)))(int)
void (*)(int) signal(int signum, void (* handler)(int))
리턴 : void (*)(int)
이름 : signal
인자 : int signum, void (* handler)(int)
12.goto 를 사용하는 이유에 대해 기술하시오.
파이프라인의 깨짐을 방지하기 위해, 또는 쓸데없는 클럭을 만들어내지 않기 위해서이다.

관없으며 둘 중 무엇인지 표기하시오)

14.

배열에 아래와 같은 정보들이 들어있다.

2400, 5000, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 234, 345, 26023, 346, 345, 234, 457, 3, 1224, 34, 646, 732, 5, 4467, 45, 623, 4, 356, 45, 6, 123, 3245, 6567, 234, 567, 6789, 123, 2334, 345, 4576, 678, 789, 1000, 978, 456, 234756 , 234 ,4564 ,3243, 876,645, 534, 423, 312, 756, 235 ,75678

여기서 가장 빈도수가 높은 3개의 숫자를 찾아 출력하시오!

19.
Stack 자료구조를 아래와 같은 포맷에 맞춰 구현해보시오.
(힌트: 이중 포인터)

ex)

```
int main(void)
{
	stack *top = NULL;
	push(&top, 1);
	push(&top, 2);
	printf("data = %d₩n", pop(&top));
}
```

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

```
struct node
{
        int data;
        struct node *link;
};
typedef struct node stack;

stack *get_node()
{
{
            stack *tmp;
            tmp=(stack *)malloc(sizeof(stack));
            tmp->link=EMPTY;
            return tmp;
}

void push(stack **top, int data)
{
            stack *tmp;
            tmp = *top;
            *top = get_node();
            (*top)->data = data;
            (*top)->link = tmp;
}
int pop(stack **top)
{
           stack *tmp;
           int num;
           tmp=*top;
           if(*top ==EMPTY)
           {
           printf("Stack is empty ~!!₩n");
           return 0;
           }
           num=tmp->data;
           *top = (*top)->link;
           free(tmp);
           return num;
           }


int main(void)
{
     stack *top = NULL;
     push(&top, 1);
     push(&top, 2);
     printf("data = %d₩n", pop(&top));
}
```

20.

Binary Tree 나 AVL Tree, Red-Black Tree 와 같이 Tree 계열의 자료구조를 재귀 호출 없이 구현하고자 한다.
이 경우 반드시 필요한 것은 무엇인가 ?

21.

다음 자료 구조를 C 로 구현하시오.

Binary Tree 를 구현하시오.
초기 데이터를 입력 받은 이후 다음 값이 들어갈 때 작으면 왼쪽 크면 오른쪽으로 보내는 방식으로 구현하시오.
삭제 구현이 가능하다면 삭제도 구현하시오.

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

#define EMPTY 0

struct node{
        int data;
        struct node *rlink;
        struct node *llink;
};
typedef struct node tree;

tree *get_node()
{
        tree *tmp;
        tmp = (tree *)malloc(sizeof(tree));

        tmp->llink=EMPTY;
        tmp->rlink=EMPTY;
        return tmp;
}



void insert(tree **root,int data)
{

        if(*root ==NULL)
        {
                *root = get_node();
                (*root)->data = data;

                return;
        }
        if((*root)->data > data)
        {
                insert(&(*root)->llink,data);
        }
        else
        {
                insert(&(*root)->rlink,data);}

}


void print_tree(tree *root)
{
```

```
        if(root)
        {
                printf("%d , ",root->data);
                print_tree(root->llink);
                print_tree(root->rlink);
        }
}

tree *chg_node(tree *root)
{
        tree *tmp=root;
        if(!root->llink)
        {
        root = root->rlink;

        }else if(!root->rlink)
        {
        root= root->llink;
        }

        free(tmp);
        return root;
}
tree *find_max(tree *root,int *data)
{
        if(root->rlink)
        {
        root->rlink = find_max(root->rlink,data);
        }
        else
        {
        *data = root->data;
        root = chg_node(root);
        }
        return root;
}
tree *detree(tree *root,int data)
{
        int num;
        if(root == NULL)
        {
                printf("There are no data that you delete\n");
                return NULL;
        }
        else if(root->data > data)
        {
                root->llink =  detree(root->llink,data);
        }
        else if(root->data<data)
        {
         root->rlink = detree(root->rlink,data);
        }
        else if(root->llink && root->rlink)
        {
                root->llink = find_max(root->llink,&num);
                root->data = num;
        }
        else
        {
                root=chg_node(root);
        }
        return root;
}
int main(void)
{       tree *root=EMPTY;
        int i=0;
```

```
        int a[20]={50,45,73,32,48,46,16,37,120,47,130,127,124};
        for(i=0;i<a[i];i++)
        {
                insert(&root,a[i]);
        }
                detree(root,45);
        print_tree(root);
        printf("\n");
    return 0;
}
```

22.

AVL 트리는 검색 속도가 빠르기로 유명하다.

Red-Black 트리도 검색 속도가 빠르지만 AVL 트리보다 느리다.

그런데 어째서 SNS 솔루션등에서는 AVL 트리가 아닌 Red-Black 트리를 사용할까 ?

avl tree 는 검색속도는 빠르지만 입력된 데이터를 저장하는 속도가 느리다 그이유는 벨런스를 맞추기 위해 데이터를 저장할때마다 이동이 많기 때문에다 하지만 레드블랙트리는 avl의 입력속도 단점을 개선하였기 때문에 많이 사용한다.

23.

AVL 트리를 C 언어로 구현하시오.

AVL 트리는 밸런스 트리로 데이터가 1, 2, 3, 4, 5, 6, 7, 8, 9 와 같이

순서대로 쌓이는 것을 방지하기 위해 만들어진 자료구조다.

```
#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;
}
```

30.리눅스 시스템에서 소프트웨어 인터럽트 번호는 0x80 에 해당한다. 소프트웨어 인터럽트가 무엇인지 기술하고 이것의 동작 메커니즘에 대해 기술하시오.

인터럽트란 급작스러운 사건 발생시 하던동작을 멈추고 급작스럽게 발생한 서건을 해결한 후 다시 원래 하던 동작을 이어서 진행하는 것을 말한다.

인터럽트와 비슷한 개념으로 폴링 이란것이 있는데 폴링은 하던 일을 끝내고 사건을 해결하는 반면 인터럽트는 하던도중 사건을 해결하고 다시 원래 상태로 돌아와 중간에 멈춘 작업을 이어서 한다.

우리가 흔히 알고있는 인터럽트의 예를 들자면
엘리베이터의 버튼을 누른 행위이다.
버튼을 누른 행위가 사건이고 그 사건을 처리하기 위해 엘리베이터가 올라가고 내려가고를 결정하는 것이 인터럽트 서비스루틴이라고 할 수있다.

인터럽트는 어떤 사건이 발생했을 경우를 설정해 그 사건이 발생하면 인터럽스 서비스루틴 함수로 진입하여 사건을 해결하거 다시 메인으로 리턴하는 메커니즘을 갖고있다.

여기서 mcu에서 서비스 루틴등의 동작을 인터럽트 레지스터를 이용한다.