TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – GJ (박현우)
uc820@naver.com

1. RB TREE의 사용이유와 규칙

✓ AVL보다 RB TREE를 사용하는 이유

검색속도는 약간 희생했지만, 삭제도 빠르고 , 입력도 속도가 빠르다.
즉, 빈번한 데이터 삽입과 삭제에는 효율적이다.
(대규모 서버 관리나 검색에 좋다.)

반면, AVL은 검색은 빠르나 삭제가 너무 많고 회전이 너무 많아서 속도가 점점 느려진다.

✓ 규칙

1. 루트 노드는 항상 검정!

2. 잎사귀 노드 어디를 가든지, 거치는 검정색의 개수는 서로 모두 같다.

3. 빨강이 연속해서 두개가 오면, 회전하거나 색상을 바꾼다.

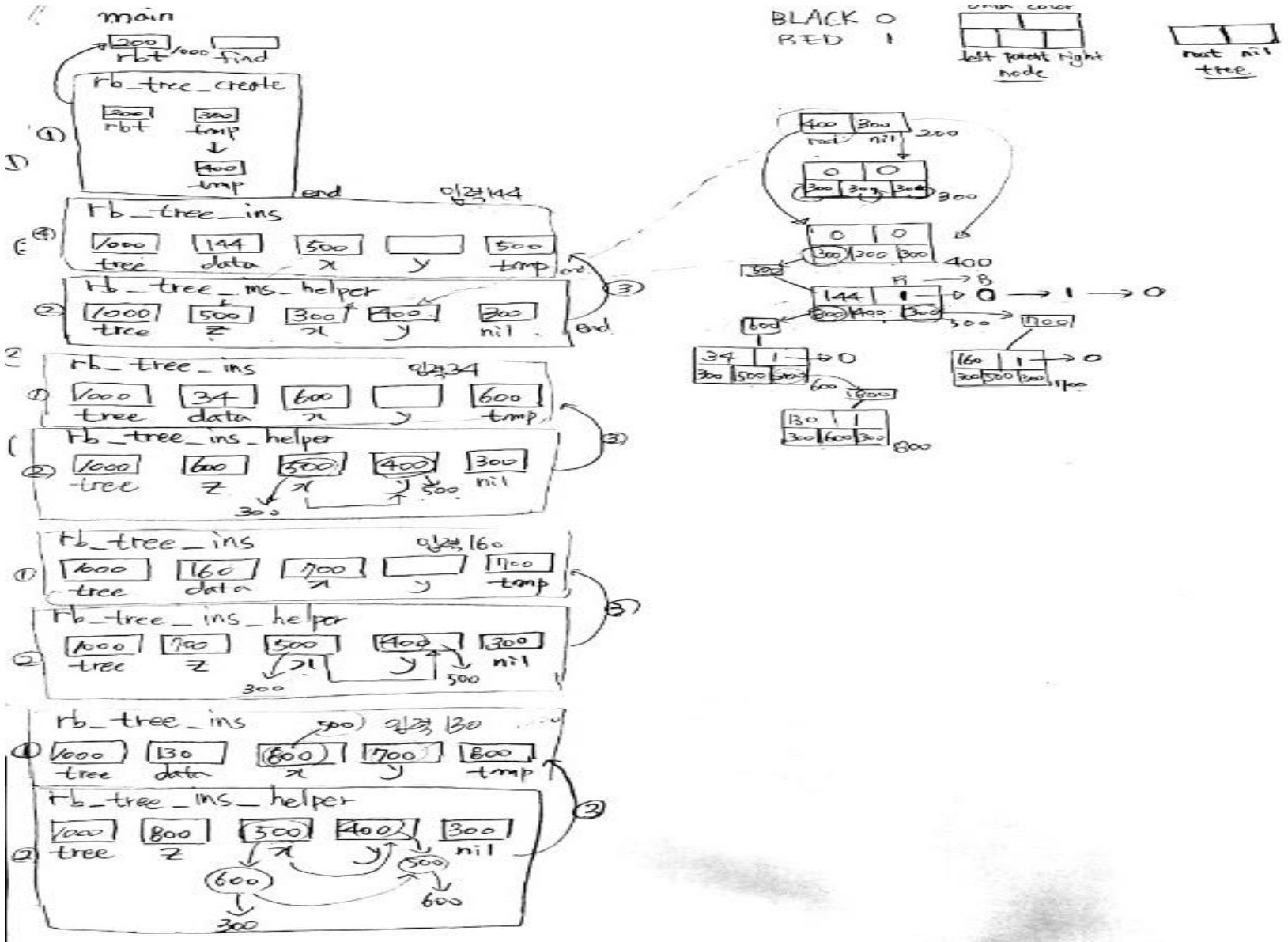
4. 현재 기준점에서 부모노드와 삼촌의 색상이 같으면 색상만 변경함.
   -> 할아버지가 빨간색이고 그 자식들은 검정색!

5. 3번 규칙을 만족하는데 4번이 만족되지 않으면 회전

6. 오른쪽 길이와 왼쪽 길이 는 2배 차이

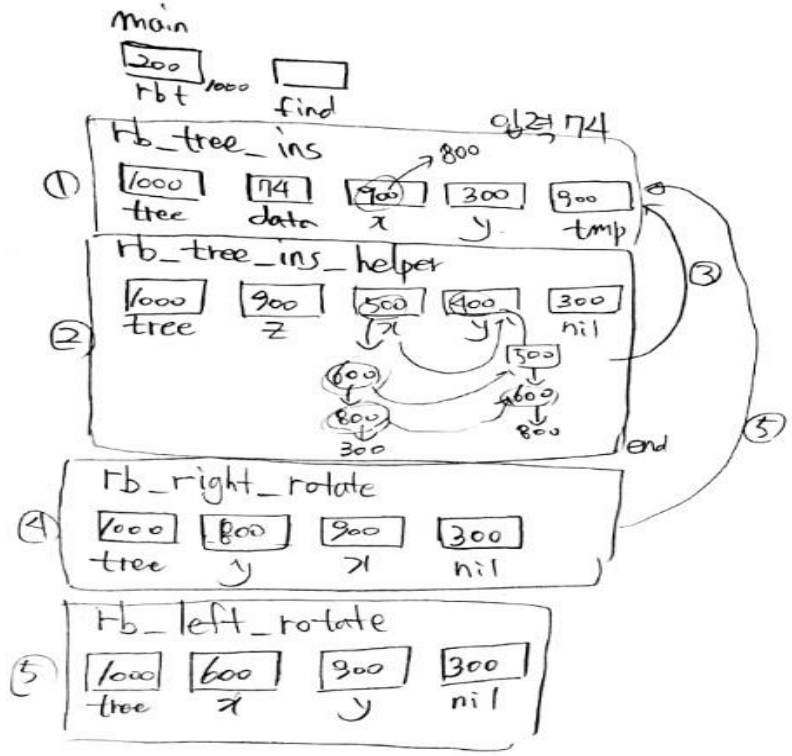
1. RB TREE 코드 이해 1(그림 그리기)

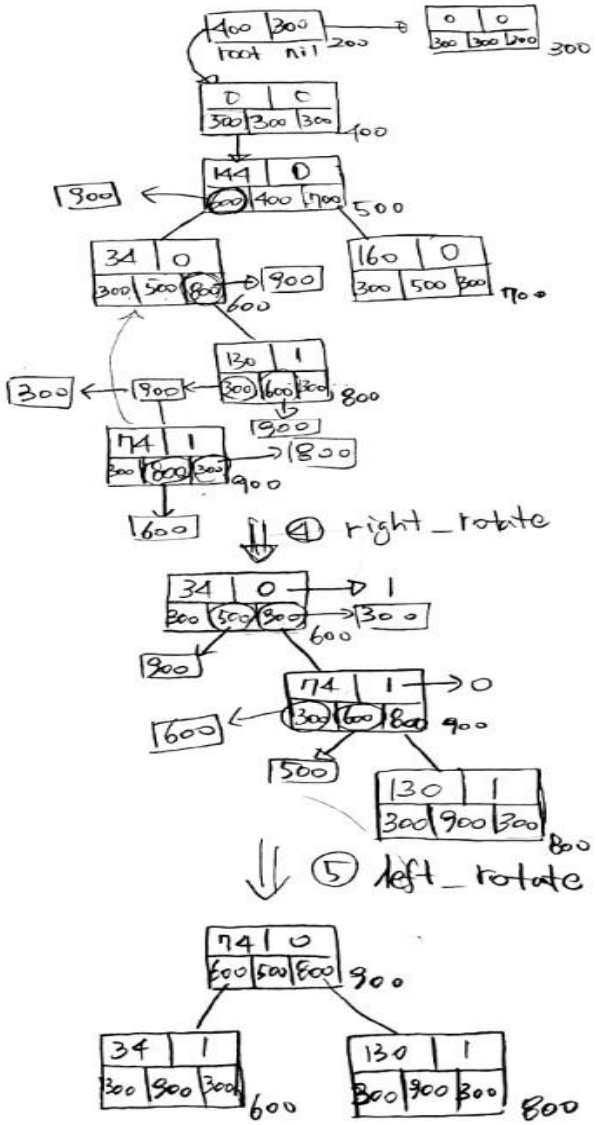
144 → 34 → 160 → 130 (1) → 74 → 13 → 101 → 129 → 186 → 6 → 91
순으로 데이터 입력 (그림에 빨간 표시까지 데이터 입력)



1. RB TREE 코드 이해 2(그림 그리기)

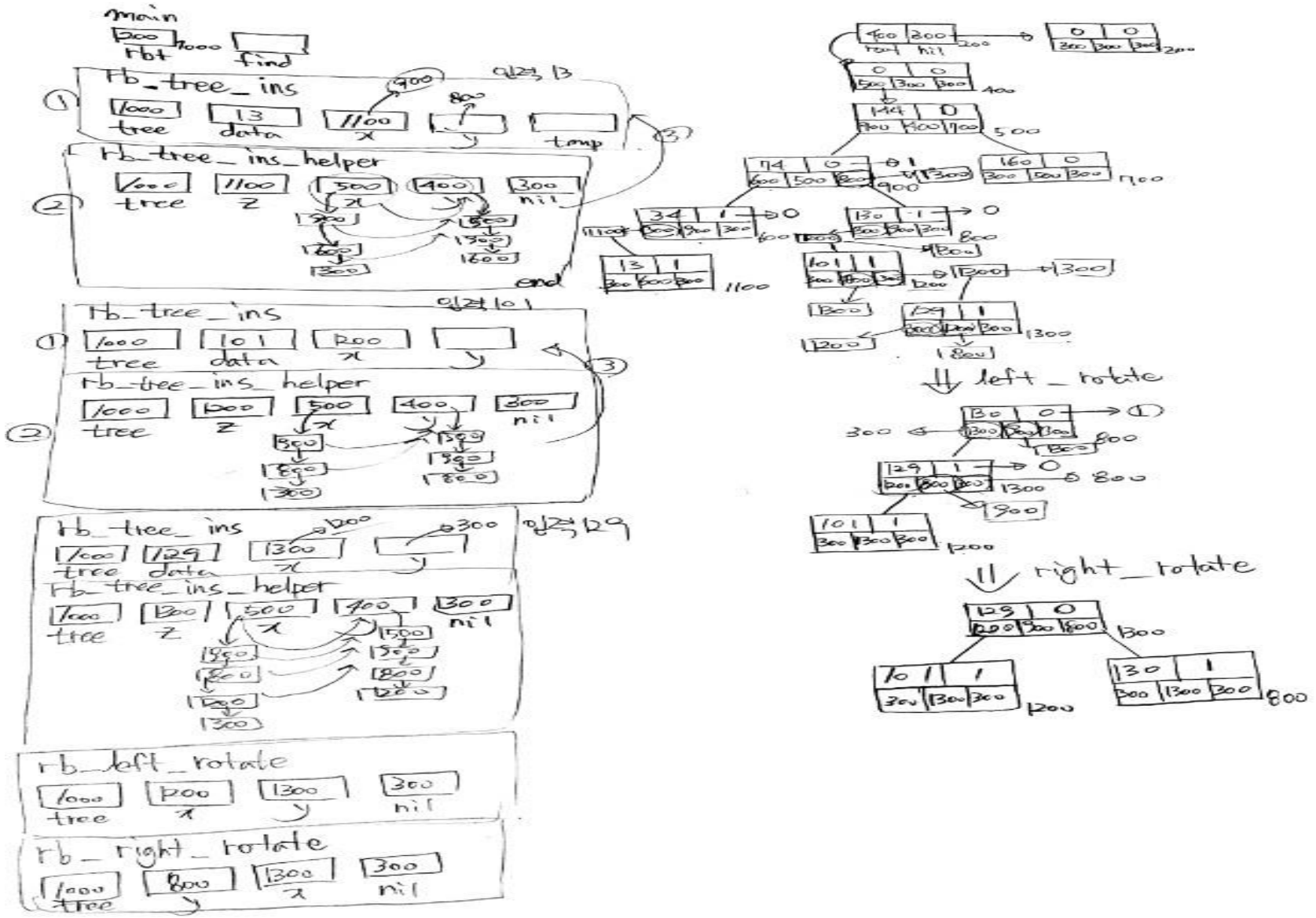
144 → 34 → 160 → 130 → 74 (2) → 13 → 101 → 129 → 186 → 6 → 91
순으로 데이터 입력 (그림에 빨간 표시까지 데이터 입력)





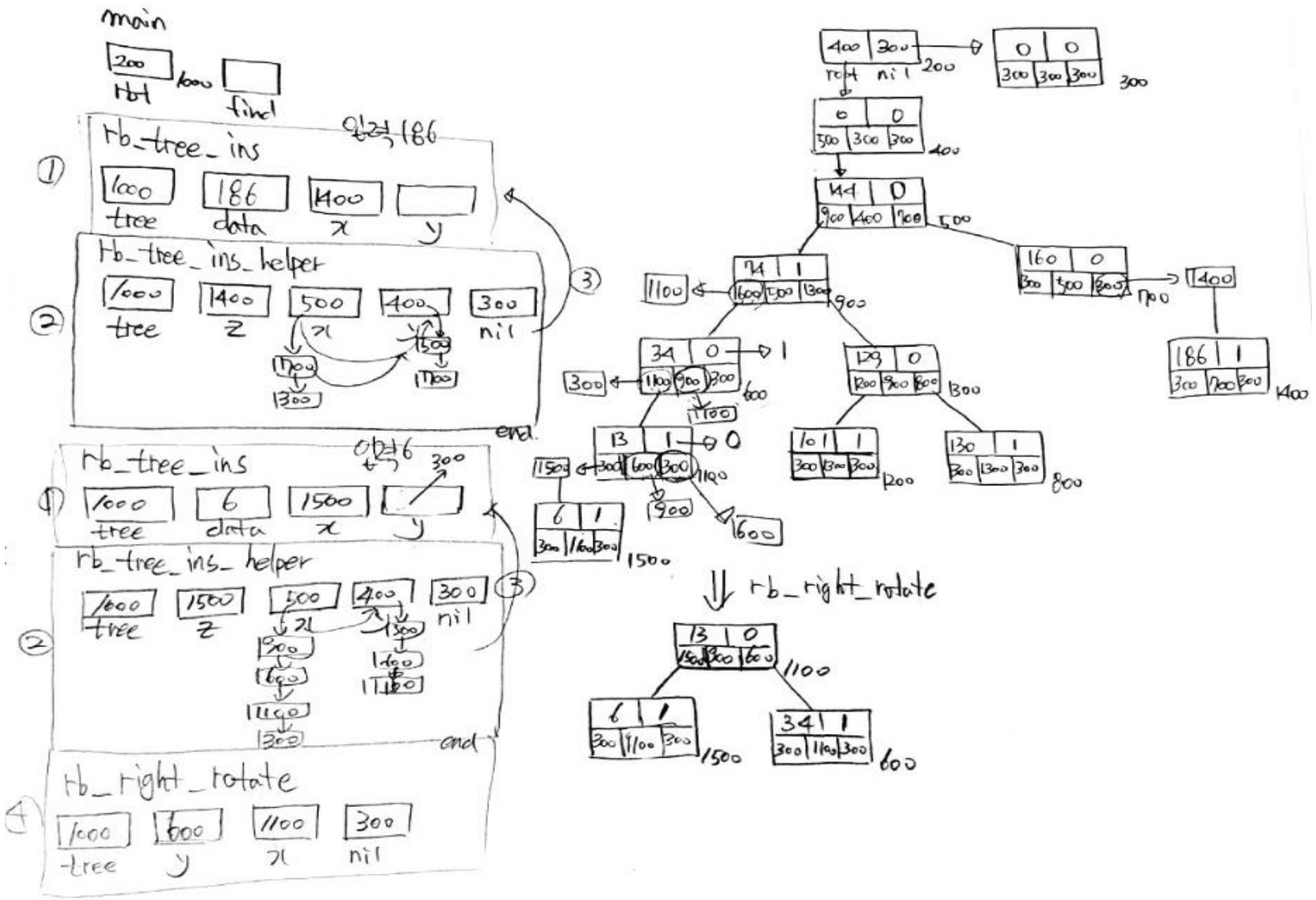
1. RB TREE 코드 이해 3(그림 그리기)

144 → 34 → 160 → 130 → 74 → 13 → 101 → 129 (3) → 186 → 6 → 91
순으로 데이터 입력 (그림에 빨간 표시까지 데이터 입력)



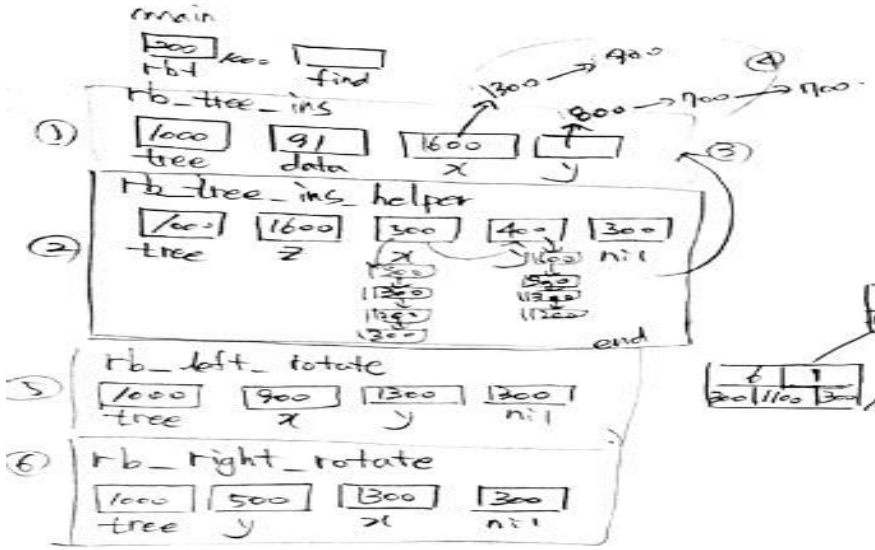
1. RB TREE 코드 이해 4(그림 그리기)

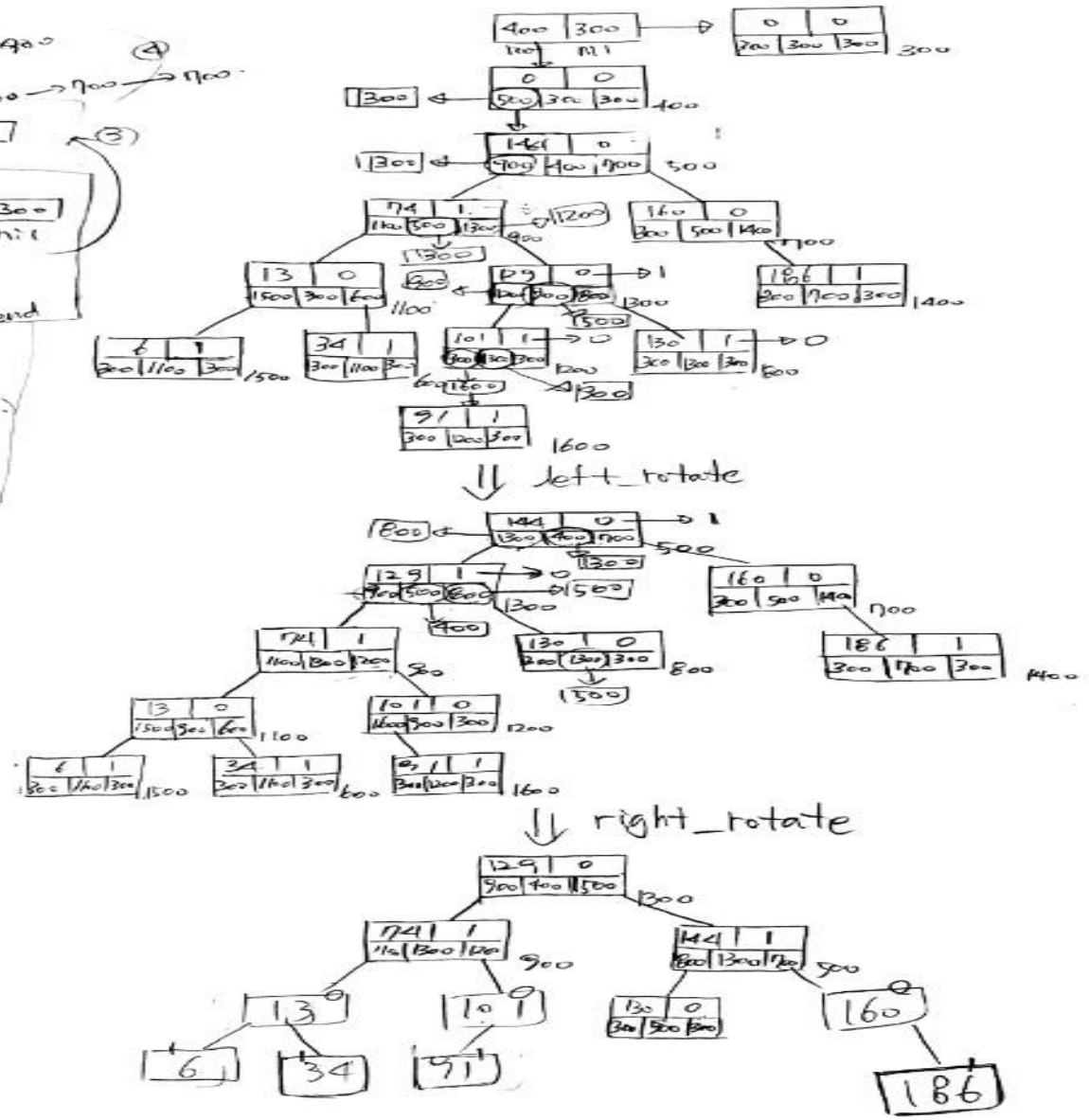
144 → 34 → 160 → 130 → 74 → 13 → 101 → 129 → 186 → 6 (4) → 91
순으로 데이터 입력 (그림에 빨간 표시까지 데이터 입력)



1. RB TREE 코드 이해 5(그림 그리기)

144 → 34 → 160 → 130 → 74 → 13 → 101 → 129 → 186 → 6 → 91 (5)
순으로 데이터 입력 (그림에 빨간 표시까지 데이터 입력)





2. 사전평가문제 1번

사전평가	현재문항 : 1 / 92
문제	단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.
내용	C 언어로 프로그램을 구현하시오.

코드

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

```
1  #include<stdio.h>
2
3  int main(void){
4
5      char capital;
6
7      char xor = 0x20;
8
9      printf("알파벳을 입력하시오 : ");
10
11     scanf("%c", &capital);
12
13     printf("변경 전 : %c \n", capital);
14
15     capital ^= xor;
16
17     printf("변경 후: %c \n", capital);
18 }
~
~
~
~
~
"1.c" 18L, 242C 저장 했습니다                    18,1          모두
```

실행

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

```
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$ ./a.out
알파벳을 입력하시오 : C
변경 전 : C
변경 후: c
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$ ./a.out
알파벳을 입력하시오 : a
변경 전 : a
변경 후: A
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$
```

2. 사전평가문제 2번

사전평가	현재문항 : 2 / 92
문제	Stack 및 Queue 외에 Tree 라는 자료구조가 있다.
내용	이 중에서 Tree 는 Stack 이나 Queue 와는 다르게 어떠한 이점이 있는가 ?

✓ 트리의 이점

stack 이나 queue같은 경우에는 선형구조라는 점 때문에 저장된 자료를 search하려면 순서대로 찾아야 한다.

이러한 점 때문에 자료가 맨 마지막에 위치해 있다면 자료를 찾는 시간이 굉장히 오래 걸리며 비효율적이다.

반면, 트리는 비선형구조를 취하고 있기 때문에 자료를 찾는 면에서 시간이 적게 걸리며 더 효율적인 search를 할 수 있다.

2. 사전평가문제 3번

사전평가	현재문항 : 3 / 92
문제	임의의 값 x가 있는데, 이를 4096 단위로 정렬하고 싶다면 어떻게 해야할까 ?
내용	C 언어로 프로그래밍 하시오.

코드

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

```
 3 int main(void){
 4
 5     long num;
 6
 7     int mask = 0xfff;
 8
 9
10     scanf("%d", &num);
11
12     while(1){
13
14         if(num & mask){
15             printf("%3x ",num & mask);
16             num >>= 12;
17         }else{
18             goto a;
19         }
20     }
21
22 a:
23     return 0;
24 }
~
"3.c" 24L, 209C 저장 했습니다                                5,1-4         바닥
```

실행

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

```
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$ ./a.out
1111111
447 10f d00 7ff hyunwoopark@hyunwoopark-P65-P67SG:~/hw$ ./a.out
1111111111111
1c7 621 eb3 7ff hyunwoopark@hyunwoopark-P65-P67SG:~/hw$ ./a.out
11111111111111111
1c7 c47 d2b 7ff hyunwoopark@hyunwoopark-P65-P67SG:~/hw$
```

2. 사전평가문제 4번

사전평가	현재문항 : 4 / 92
문제	int p[7] 와 int (*p)[7] 가 있다.
내용	이 둘의 차이점에 대해 기술하시오.

✓ int p[7] 와 int (*p)[7]의 차이

int p[7]은 int형 배열 7개를 가질 수 있는 변수 p라는 뜻이고,

int (*p)[7]은 int *[7] p로 달리 쓸 수 있으며 int형 7개를 붙여서 쓰는 포인터 변수 p라는 뜻이다.

즉, p는 a[][7]와 같은 배열을 가리킬 수 있는 변수이다.

2. 사전평가문제 5번

사전평가	현재문항 : 5 / 92
문제	다음을 C 언어로 구현하시오.
내용	이번 문제의 힌트를 제공하자면 함수 포인터와 관련된 문제다. 아래와 같은 행렬을 생각해보자! 1 2 3 1 2 3 sapply(arr, func) 이라는 함수를 만들어서 위의 행렬을 아래와 같이 바꿔보자! sapply 에 func 함수가 연산을 수행하는 함수로 만들어야 한다. 1 2 3 1 4 9

코드

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

```
 1 #include<stdio.h>
 2
 3 void add(int (*arr)[3]){
 4
 5     arr[1][1] +=2;
 6     arr[1][2] +=6;
 7
 8 }
 9
10 void sapply(int (*arr)[3], void(*func)(int (*arr)[3])){
11
12     func(arr);
13
14 }
15 void print_arr(int (*arr)[3]){
16
17     int i,j;
18     for(i=0; i<2;i++){
19
20         for(j=0; j<3;j++)
21             printf("%d ",arr[i][j]);
22
23         printf("\n");
24     }
25
26 }
27
28 int main(void){
29
30     int arr[2][3] = { {1,2,3}, {1,2,3} };
31
32     printf("함수 적용 전\n");
33     print_arr(arr);
34     sapply(arr,add);
35     printf("함수 적용 후\n");
36     print_arr(arr);
37     return 0;
38 }
~
~
~
                                                    36,11-14      모두
```

실행

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

```
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$ ./a.out
함수 적용 전
1 2 3
1 2 3
함수 적용 후
1 2 3
1 4 9
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$
```

2. 사전평가문제 6번

사전평가	현재문항 : 6 / 92
문제	다음 사항에 대해 아는대로 기술하시오.
내용	Intel Architecture 와 ARM Architecture 의 차이점은 무엇인가 ?

- ARM은 Intel에 비해 디자인적으로 가능한 단순하게 만들었고, low-power에서 작동하도록 설계되어 졌다. 그렇기 때문에, 핸드폰이나 태블릿과 같은 소형 기기에 적합하다.

- 반면, Intel은 ARM에 비해 복잡한 디자인이며, 더 다양한 기기와 상호작용할 수 있도록 설계되었다. 일반적으로는 데스트 탑이나 랩탑과 같은 큰 기기에 적합하다.

아래의 기사를 참고,

http://www.alphr.com/features/390064/arm-vs-intel-processors-what-s-the-difference

2. 사전평가문제 7, 9번

사전평가	현재문항 : 7 / 92
문제	이것이 없으면 C 언어의 함수를 호출할 수 없다.
내용	여기서 이야기하는 이것은 무엇일까 ?

- 기계어로 본다면 call 함수가 없으면 함수를 호출이 안 된다.

사전평가	현재문항 : 9 / 92
문제	다음 사항에 대해 기술하시오.
내용	Memory Hierarchy(메모리 계층 구조)에 대해 기술하시오.

- 레지스터
- 캐시
- 메모리
- 하드디스크

2. 사전평가문제 10번

사전평가	현재문항 : 10 / 92
문제	다음 질문에 대해 기술하시오.
내용	C 언어에서 중요시하는 메모리 구조에 대해 기술하시오. (힌트: Stack, Heap, Data, Text 에 대해 기술하시오.)

✓ stack은 지역 변수와 함수 호출시 변수가 저장되는 공간
또한, stack은 다른 영역과 달리 위에서 아래로 쌓이는 구조

✓ data 전역변수, 정적 변수가 저장되는 공간

✓ text 프로그램의 코드 부분이 저장되는 영역
순차처리

✓ heap
프로그래머가 동적 할당을 해서 생성한 변수가 저장되는 공간

2. 사전평가문제 11, 12번

사전평가	현재문항 : 11 / 92
문제	다음 사항에 대해 답하시오.
내용	파이프라인이 깨지는 경우에 대해 기술하시오.

- ✓ 파이프라인은 call 이나 jmp 등 분기 명령어가 너무 자주 호출될 때 cpu의 파이프라인에 치명적인 손실을 가져다 준다.
- ✓ cpu clock의 낭비로 인해 파이프라인이 깨질 수 있음.

사전평가	현재문항 : 12 / 92
문제	아래 질문에 대해 답하시오.
내용	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 (*)(int) signal(int signum, void (* handler)(int))
- ✓ 리턴 : void (*)(int)
- ✓ 함수 이름: signal
- ✓ 인자 : int signum, void (* handler)(int)

2. 사전평가문제 13번

사전평가	현재문항 : 13 / 92
문제	다음 질문에 대해 답하시오.
내용	goto 를 사용하는 이유에 대해 기술하시오.

예를 들어, for문을 중복해서 사용하고 if와 break로 for문에서 빠져 나올 때,
상당히 많은 불필요한 명령어들이 사용된다.

이러한 단점을 보완하기 위해 goto 문법을 사용하면 불필요한 코드를 사용하지 않고
한 번에 for문을 빠져나올 수 있다.

If와 break를 사용하면 기본적으로 mov, cmp, jmp를 해야한다.
하지만, goto는 jmp 하나로 끝난다.

또한, for문과 if,break를 여러 개 조합을 할수록 mov, cmp, jmp가 늘어난다.
여기서 문제는 jmp이다.

Call 이나 jmp를 cpu instruction 레벨에서 분기 명령어라고 하고
이들은 cpu 파이프라인에 치명적인 손실을 가져다 준다.

즉, 성능면으로만 보아도 goto가 월등히 좋다는 것을 알 수 있다.

2. 사전평가문제 19번

사전평가 현재문항 : 19 / 92

문제	아래 자료 구조를 C 언어로 구현해보시오.

내용	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));
	}

코드

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

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

typedef struct node{

    int data;
    struct node *link;
}stack;

stack *getnode(void){

    stack *tmp;
    tmp = (stack *)malloc(sizeof(stack));
    tmp->link = NULL;

    return tmp;
}

void push(stack **top, int data){

    stack *tmp = *top;

    *top = getnode();
    (*top)->link = tmp;
    (*top)->data = data;

}

int pop(stack **top){

    stack *tmp = *top;
    int data = (*top)->data;

    if(*top == NULL){
        printf("stack is empty\n");
        return 0;
    }
    (*top) = (*top)->link;

    free(tmp);


    return data;
}
int main(void){

    stack *top = NULL;
    push(&top,1);
    push(&top,2);

    printf("data = %d\n",pop(&top));
    return 0;
}
```

실행

```
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$ ./a.out
data = 2
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$
```

2. 사전평가문제 20번

사전평가	현재문항 : 20 / 92
문제	다음 질문에 답하시오.
내용	Binary Tree 나 AVL Tree, Red-Black Tree 와 같이 Tree 계열의 자료구조를 재귀 호출 없이 구현하고자 한다. 이 경우 반드시 필요한 것은 무엇인가 ?

✓ Stack이 필요하다.

2. 사전평가문제 21번

사전평가	현재문항 : 21 / 92
문제	다음 자료 구조를 C 로 구현하시오.
내용	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.

코드

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

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

typedef struct node{

    int data;
    struct node *left;
    struct node *right;
}tree;

tree *getnode(){

    tree *tmp;
    tmp = (tree *)malloc(sizeof(tree));
    tmp->left = NULL;
    tmp->right = NULL;
    return tmp;
}

void ins_tree(tree **root, int data){
    if(!(*root)){

        (*root) = getnode();
        (*root)->data = data;
        return;
    }

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

}

tree *chg_node(tree *root){

    tree *tmp = root;
    if( !(root->left) )
        root = root->right;
    if( !(root->right) )
        root = root->left;

    free(tmp);

    return root;
}

tree *find_max(tree *root, int *num){

    if( root->right){
        root = find_max(root->right, num);
        return root;
    }

    *num = root->data;
    root = chg_node(root);

    return root;
}
```

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

```c
    return root;
}

tree *del_tree(tree *root, int data){

    tree *tmp;
    int num;

    if( root->data > data)
        root = root->right;
    else if( root->data <data)
        root = root->left;
    else if( root->left && root->right){

        root->left->right = find_max( root->left, &num);
        root->data = num;

        printf("= %d \n",num);
    }else{

        root = chg_node(root);
    }


    return root;
}


void print_tree(tree **root){

    if( *root ){
        printf("%d\n", (*root)->data);
        print_tree( &(*root)->left);
        print_tree( &(*root)->right);
    }

}

int main(void){

    tree *root = NULL;
    int data[13] = {50, 45, 73, 32, 48, 46, 16, 37, 120, 47, 130, 127, 124};

    int i;
    for(i = 0; i< (int)(sizeof(data)/sizeof(int)) ;i++){

        ins_tree(&root,data[i]);
    }

    print_tree(&root);

    printf("after del ");
    root = del_tree(root, 50);

    print_tree(&root);
    return 0;
}
```

114,1 바닥

실행

```
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$ ./a.out
50
45
32
16
37
48
46
47
73
120
130
127
124
after del = 48
48
45
32
16
37
46
47
73
120
130
127
124
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$
```