EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv2 과정
[자료구조 프로그래밍]

제 1기

2021. 11. 19

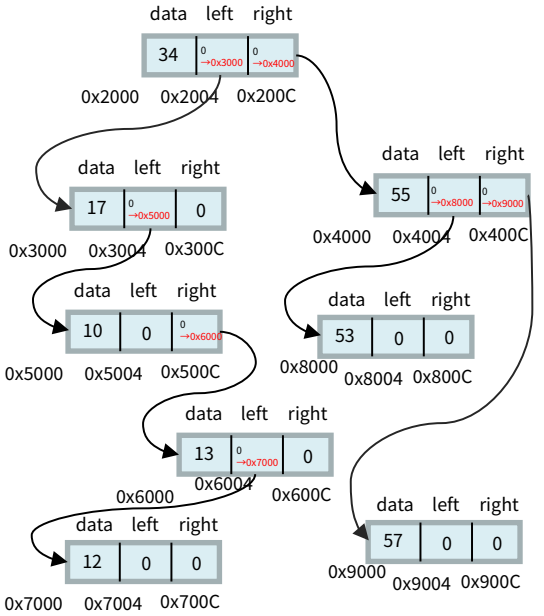
박태인

목차

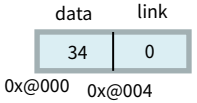
1) nr_print_tree 함수 (비재귀 print 함수) [bin_tree.c]

함수 구현 전략.

HEAP



STACK



- 새로운 stack을 생성한다.
- Stack의 data에 출력 할 data를 push
- 해당 data의 tree * 형 변환 후 포인팅 하는 변수(t)생성
- 다음 stack 의 data를 포인팅 후 pop
- 변수(t)의 data 출력 및 left, right 유무 확인 후 출력

전략 :

Stack에 data 정보를 저장한 후
Tree 형태로 형 변환해서
T로 저장한 후
Pop 하고
Tree t 를 정보를 출력하고
Push를 통해 왼쪽, 오른쪽을 다시
있는 것은 모두 출력 해놓고
다음 출력 데이터를
대상으로 linked 해놓는다.

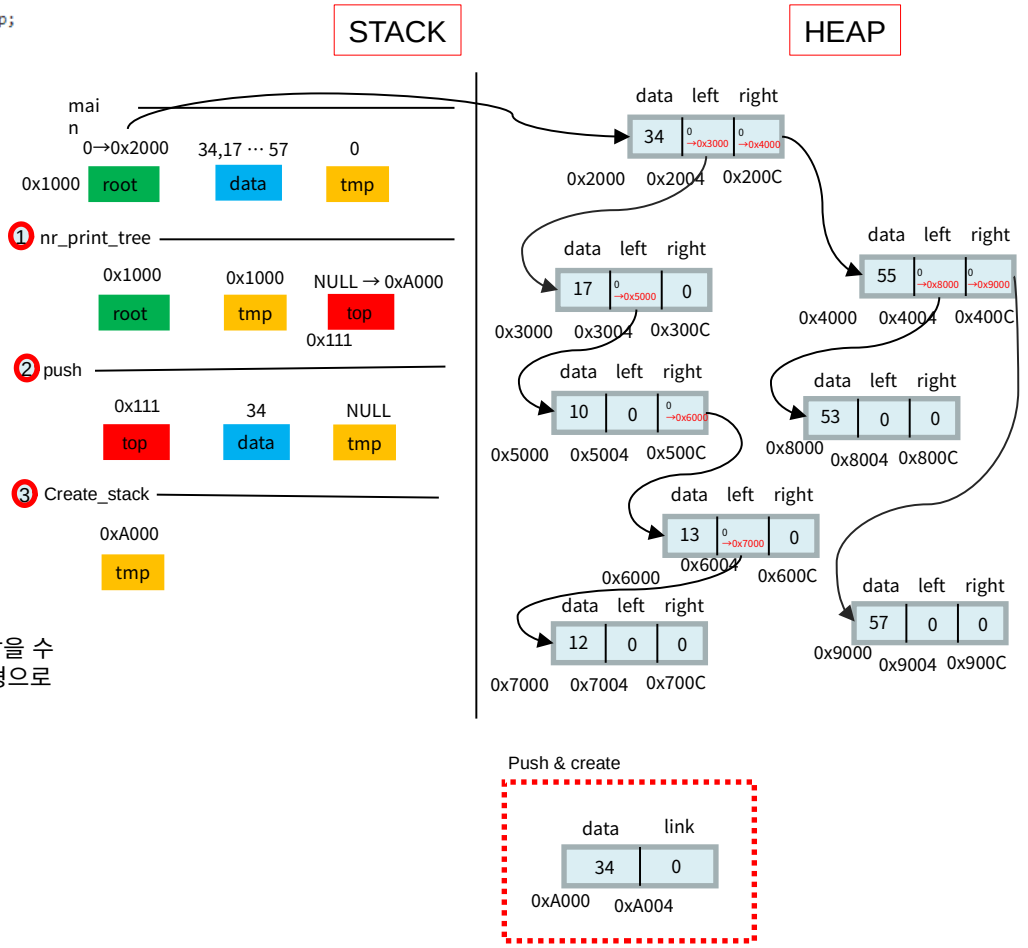
```
data = 34, left = 17, right = 55
data = 17, left = 10, right = NULL
data = 10, left = NULL, right = 13
data = 13, left = 12, right = NULL
data = 12, left = NULL, right = NULL
data = 55, left = 53, right = 57
data = 53, left = NULL, right = NULL
data = 57, left = NULL, right = NULL
tmp->data = 13
데이터를 찾을 수 없습니다!
12 삭제
data = 34, left = 17, right = 55
data = 17, left = 10, right = NULL
data = 10, left = NULL, right = 13
data = 13, left = NULL, right = NULL
data = 55, left = 53, right = 57
data = 53, left = NULL, right = NULL
data = 57, left = NULL, right = NULL
10 삭제
data = 34, left = 17, right = 55
data = 17, left = 13, right = NULL
data = 13, left = NULL, right = NULL
data = 55, left = 53, right = 57
data = 53, left = NULL, right = NULL
data = 57, left = NULL, right = NULL
55 삭제
data = 34, left = 17, right = 54
data = 17, left = 13, right = NULL
data = 13, left = NULL, right = NULL
data = 54, left = 53, right = 57
data = 53, left = NULL, right = NULL
data = 57, left = NULL, right = NULL
```

1-1) 비 재귀 nr_print_tree(&root)

```
① nr_print_tree(&root);   [main]

void nr_print_tree(tree **root)
{
        tree **tmp = root;
        stack *top = NULL;

    ② push(&top, *tmp);

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

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

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

                stack *tmp = *top;
            ③ *top = create_stack_node();
                //(*top)->data = malloc(sizeof(void *));
                (*top)->data = data;
                (*top)->link = tmp;
        }
```

어떤 형태의 데이터도 받을 수 있게 하기 위해 void * 형으로

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



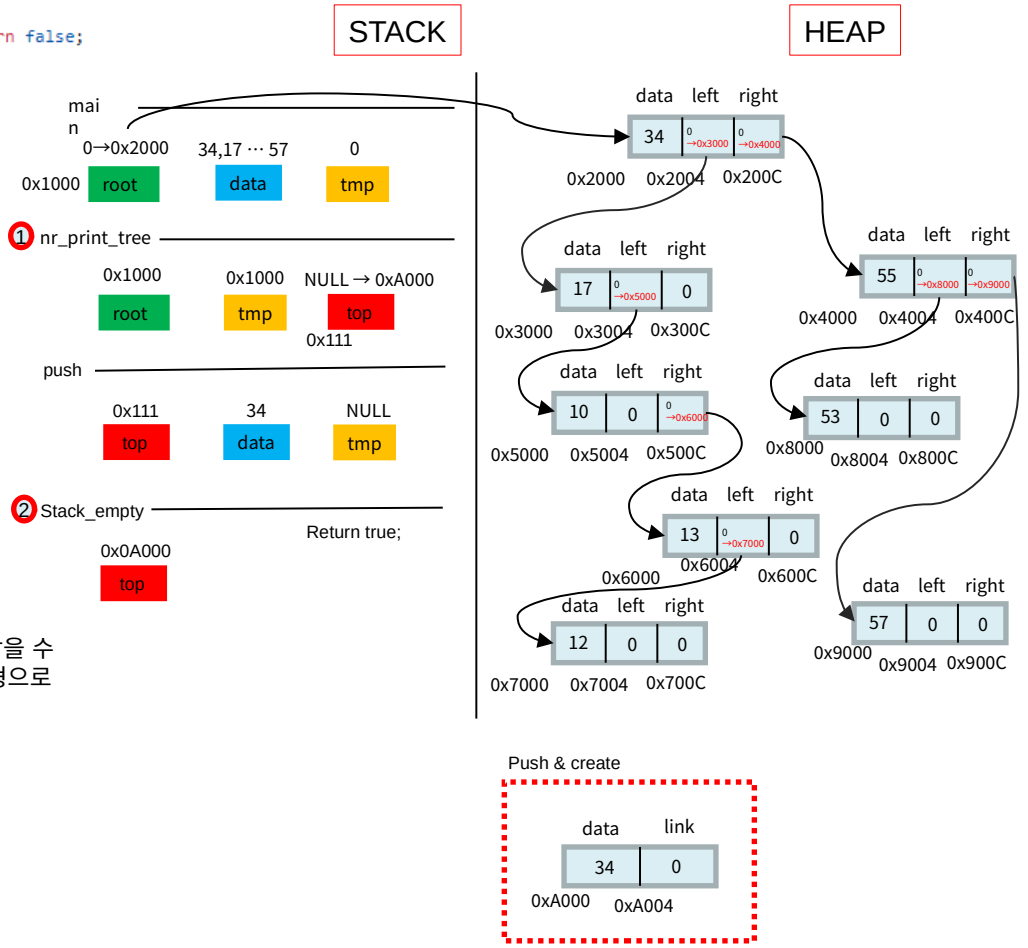
1-2) 비 재귀 nr_print_tree(&root)

① `nr_print_tree(&root);` [main]

```
void nr_print_tree(tree **root)
{
        tree **tmp = root;
        stack *top = NULL;

        push(&top, *tmp);
        ②
        while(stack_is_not_empty(top))
        {
                tree *t = (tree *)pop(&top);
                tmp = &t;

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

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

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

어떤 형태의 데이터도 받을 수 있게 하기 위해 void * 형으로

②
```
bool stack_is_not_empty(stack *top)
{
        if(top != NULL)
                return true;
        else
                return false;
}
```



1-3) 비 재귀 nr_print_tree(&root)

① `nr_print_tree(&root);` [main]

```
void nr_print_tree(tree **root)
{
        tree **tmp = root;
        stack *top = NULL;

        push(&top, *tmp);

        while(stack_is_not_empty(top))
        {
            ② tree *t = (tree *)pop(&top);
                tmp = &t;

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

                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);
        }
}
```

Data 34 인쇄

Left 17 인쇄

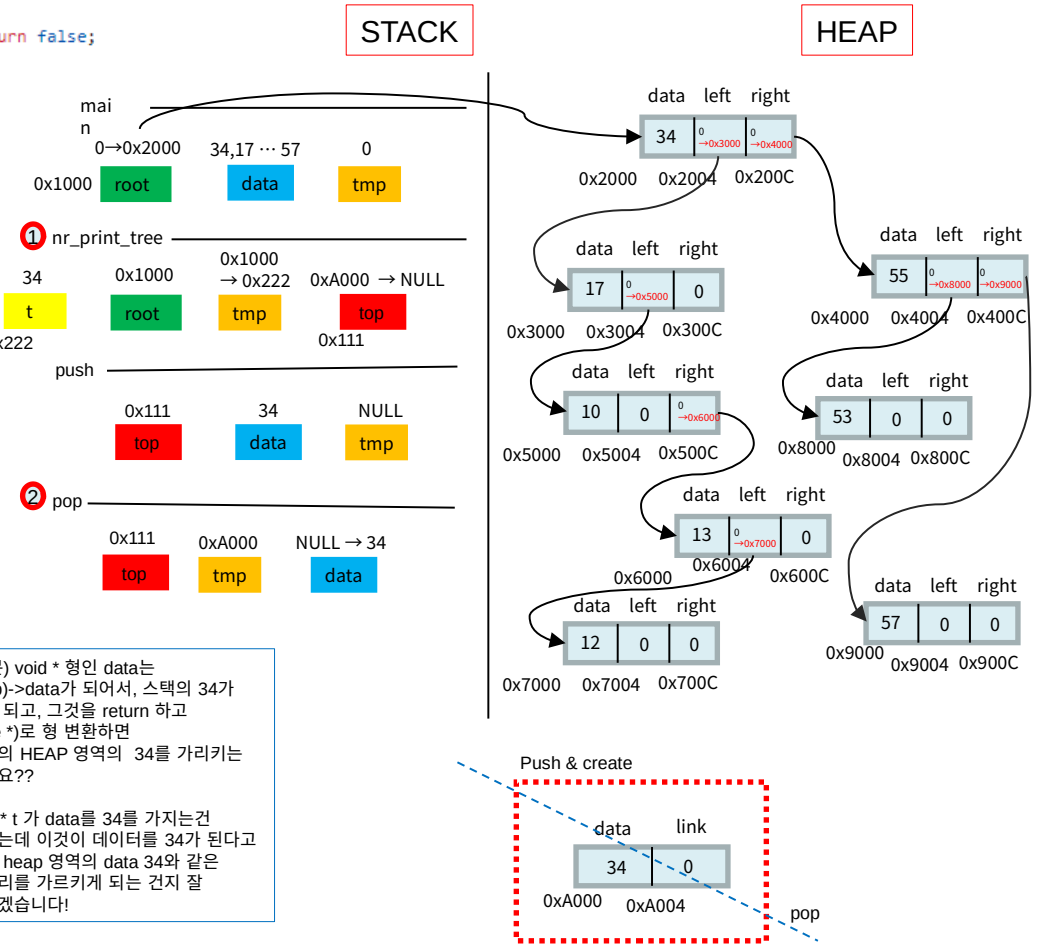
Right 55 인쇄

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

```
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 data;
}
```

(질문) void * 형인 data는
(*top)->data가 되어서, 스택의 34가
입력 되고, 그것을 return 하고
(tree *)로 형 변환하면
왼쪽의 HEAP 영역의 34를 가리키는
건가요??

tree * t 가 data를 34를 가지는건
알겠는데 이것이 데이터를 34가 된다고
해서 heap 영역의 data 34와 같은
메모리를 가르키게 되는 건지 잘
모르겠습니다!



1-4) 비 재귀 nr_print_tree(&root)

① `nr_print_tree(&root);` [main]

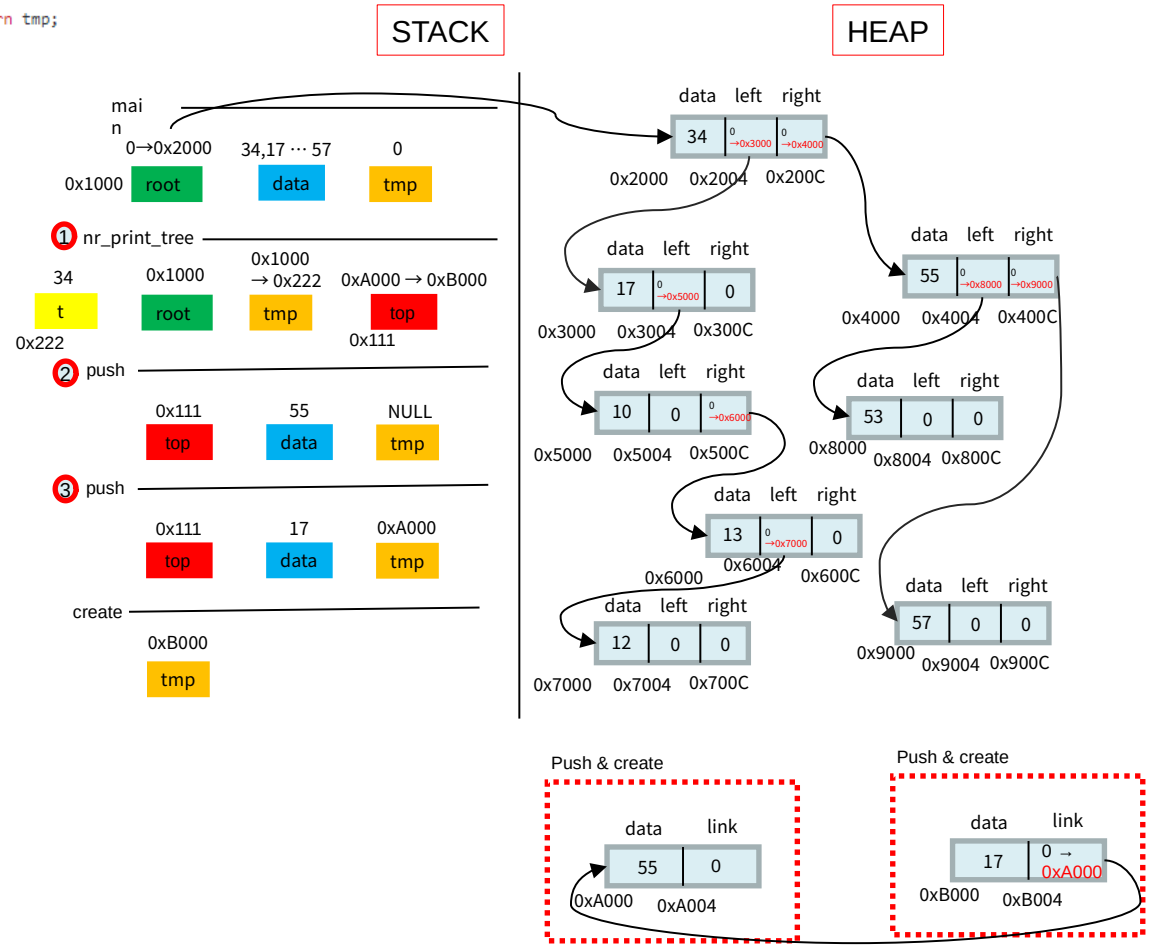
```
void nr_print_tree(tree **root)
{
        tree **tmp = root;
        stack *top = NULL;

        push(&top, *tmp);

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

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

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

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

Data 34 인쇄

Left 17 인쇄

Right 55 인쇄

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



1-5) 비 재귀 nr_print_tree(&root)

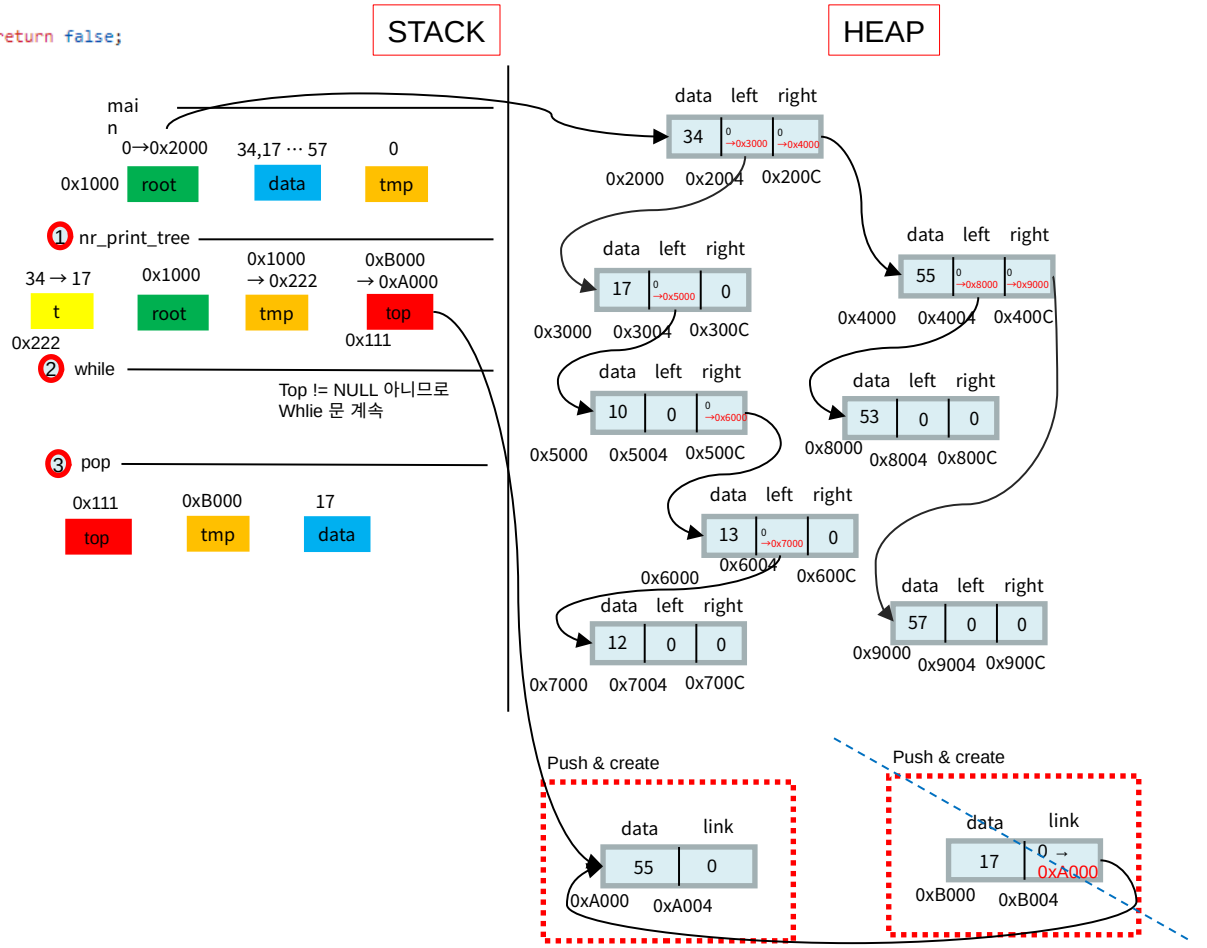
① `nr_print_tree(&root);` [main]

```
void nr_print_tree(tree **root)
{
        tree **tmp = root;
        stack *top = NULL;

        push(&top, *tmp);

        while(stack_is_not_empty(top))
        {
③           tree *t = (tree *)pop(&top);
            tmp = &t;
                                        Data 17 인쇄
            printf("data = %d, ", (*tmp)->data);

            if((*tmp)->left)            Left 10 인쇄
                    printf("left = %d, ", (*tmp)->left->data);
            else
                    printf("left = NULL, ");

            if((*tmp)->right)           NULL 인쇄
                    printf("right = %d\n", (*tmp)->right->data);
            else
                    printf("right = NULL\n");

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

```
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 data;
}
```

②

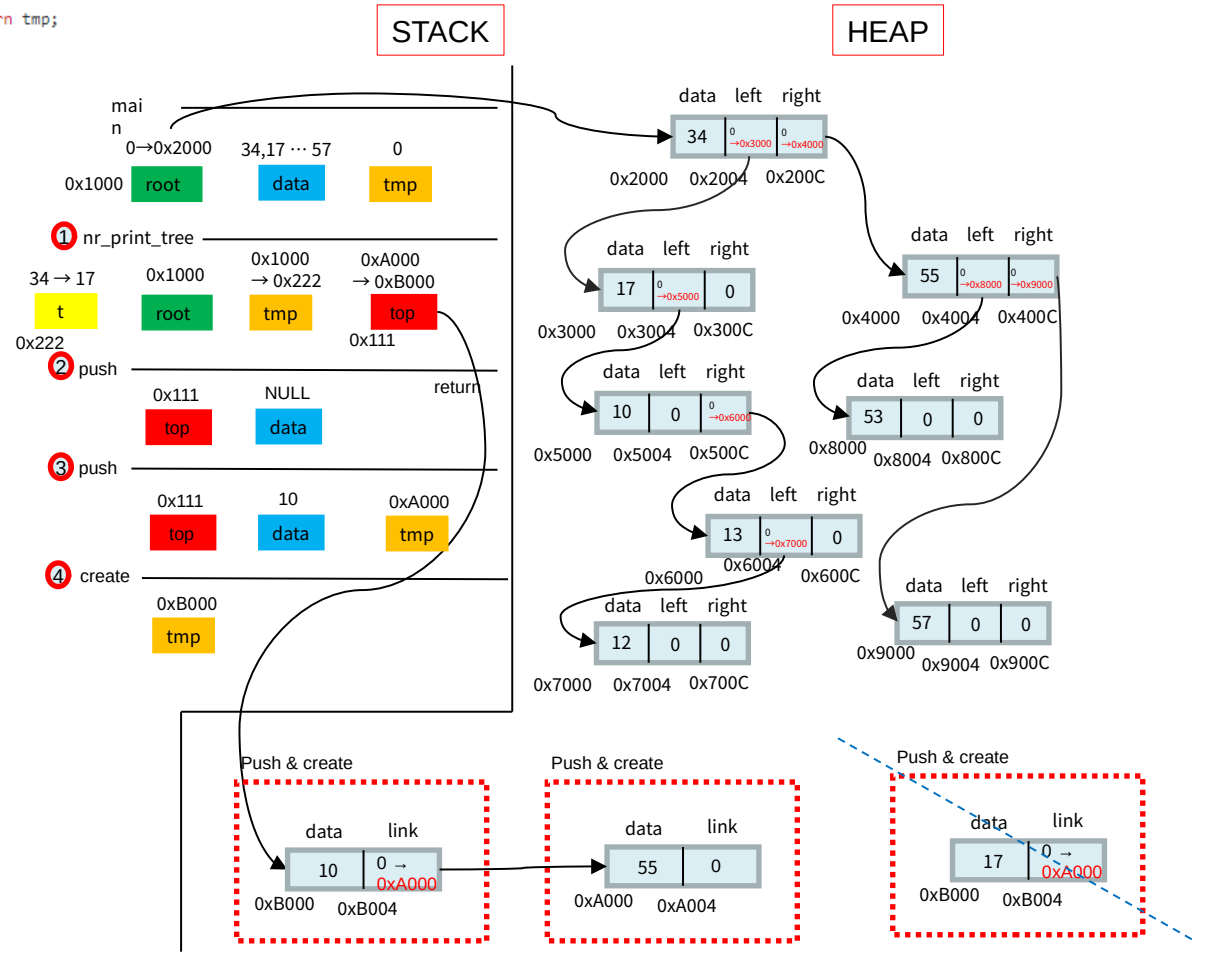
```
bool stack_is_not_empty(stack *top)
{
        if(top != NULL)
                return true;
        else
                return false;
}
```



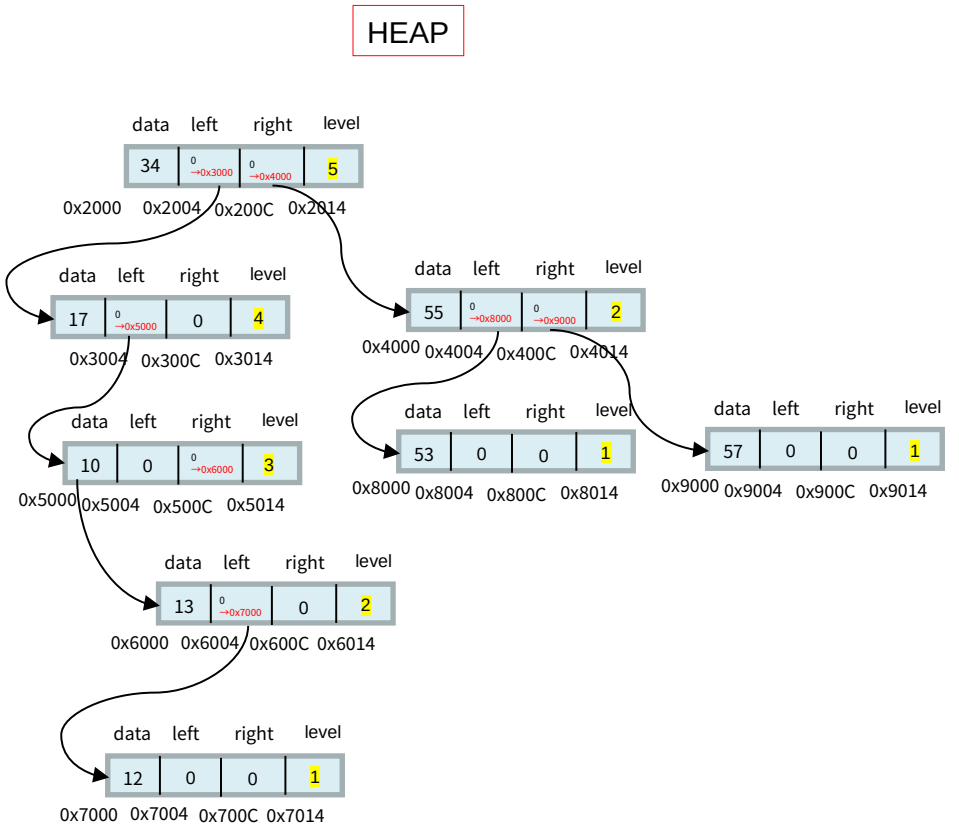
1-5) 비 재귀 nr_print_tree(&root)

```
① nr_print_tree(&root);   [main]

void nr_print_tree(tree **root)
{
        tree **tmp = root;
        stack *top = NULL;

        push(&top, *tmp);

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

                printf("data = %d, ", (*tmp)->data);        Data 17 인쇄

                if((*tmp)->left)                            Left 10 인쇄
                        printf("left = %d, ", (*tmp)->left->data);
                else
                        printf("left = NULL, ");

                if((*tmp)->right)                           NULL 인쇄
                        printf("right = %d\n", (*tmp)->right->data);
                else
                        printf("right = NULL\n");

            ② push(&top, (*tmp)->right);
            ③ push(&top, (*tmp)->left);
        }
}
        void push(stack **top, void *data)
        {
                if(data == NULL)
                        return;

            ④ stack *tmp = *top;
               *top = create_stack_node();
               //(*top)->data = malloc(sizeof(void *));
               (*top)->data = data;
               (*top)->link = tmp;
        }
```

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



2) Bin_tree level 체크 (참고, AVL 트리의 경우 rotation 이전에 한번 rotation 이후에 한번 더)




```
data = 34,      level = 5,      left = 17,      right = 55
data = 17,      level = 4,      left = 10,      right = NULL
data = 10,      level = 3,      left = NULL,    right = 13
data = 13,      level = 2,      left = 12,      right = NULL
data = 12,      level = 1,      left = NULL,    right = NULL
data = 55,      level = 2,      left = 53,      right = 57
data = 53,      level = 1,      left = NULL,    right = NULL
data = 57,      level = 1,      left = NULL,    right = NULL
```

전략 :

Node 생성시 level 표시를 할 수 있도록 만든다.

Level 표시는 트리가 연결 되고 난 후 tree 의 좌,우에 데이터 값이 존재 하면 cnt를 활용해 레벨의 높이를 가늠하고,

레벨의 숫자가 현재 저장된
레벨 정보보다 높을 시 갱신하여
트리의 레벨 표시를 업데이트 하는 방법을 사용.

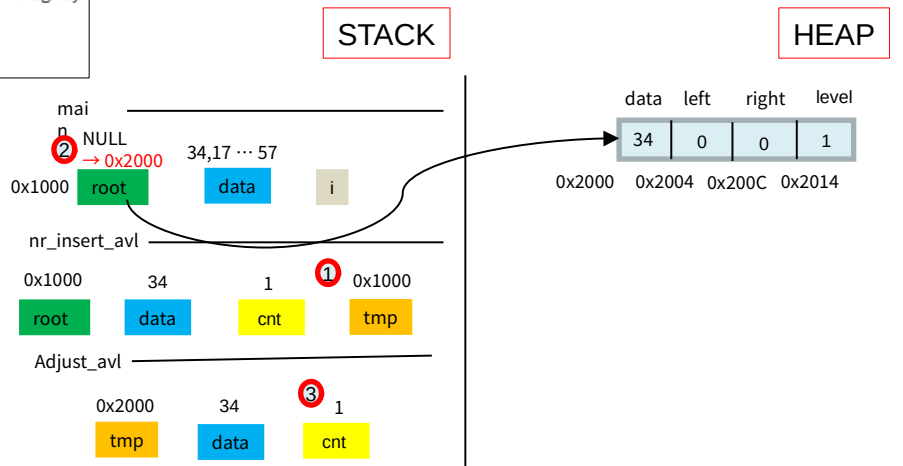
2-1) Bin_tree level 체크

```c
int main(void)
{
	int i;
	avl *root = NULL;
	int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };

	for (i = 0; i < 8; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	int level;
};
```

```c
void nr_insert_avl_data(avl **root, int data)
{
	avl **tmp = root;
	int cnt = 1;

	while(*tmp)
	{
		cnt++;

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

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

	adjust_avl_level(root, data, cnt);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->level = 1;

	return tmp;
}
```




```c
void adjust_avl_level(avl **root, int data, int cnt)
{
	while(*root)
	{
		if ((*root)->level < cnt)
			(*root)->level = cnt;

		cnt--;

		if((*root)->data > data)
			root = &(*root)->left;
		else if((*root)->data < data)
			root = &(*root)->right;
		else
			break;
	}
}
```

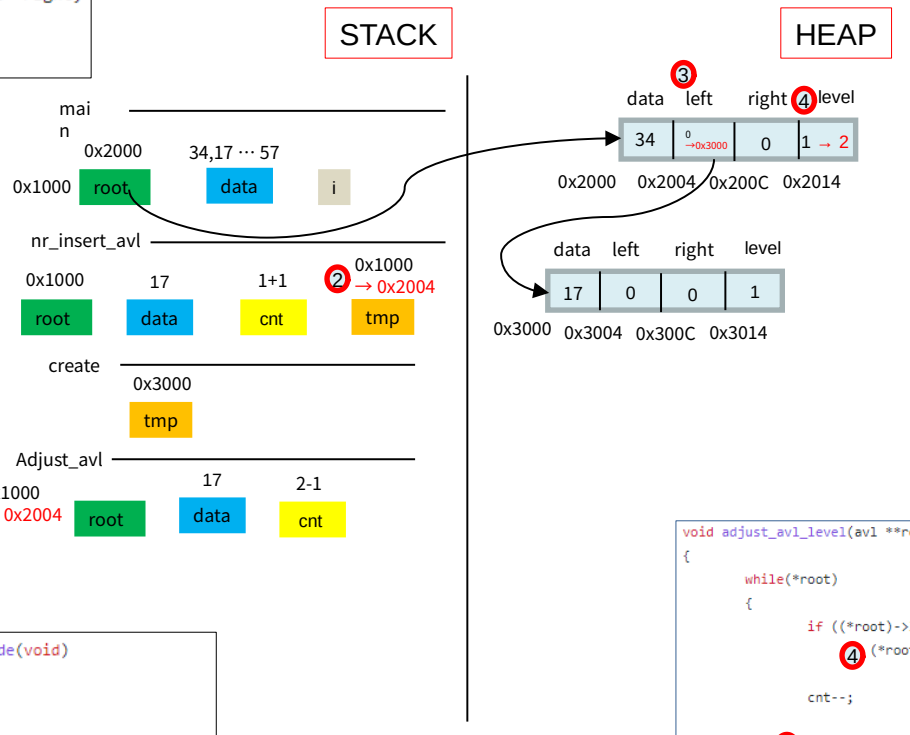
2-2) Bin_tree level 체크

```c
int main(void)
{
	int i;
	avl *root = NULL;
	int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };

	for (i = 0; i < 8; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	int level;
};
```

```c
void nr_insert_avl_data(avl **root, int data)
{
	① avl **tmp = root;
	int cnt = 1;

	while(*tmp)
	{
		cnt++;

		② if((*tmp)->data > data)
			tmp = &(*tmp)->left;
		else if((*tmp)->data < data)
			tmp = &(*tmp)->right;
	}

	③ *tmp = create_avl_node();
	(*tmp)->data = data;

	adjust_avl_level(root, data, cnt);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->level = 1;

	return tmp;
}
```




```c
void adjust_avl_level(avl **root, int data, int cnt)
{
	while(*root)
	{
		if ((*root)->level < cnt)
			④ (*root)->level = cnt;

		cnt--;

		⑤ if((*root)->data > data)
			root = &(*root)->left;
		else if((*root)->data < data)
			root = &(*root)->right;
		else
			break;
	}
}
```

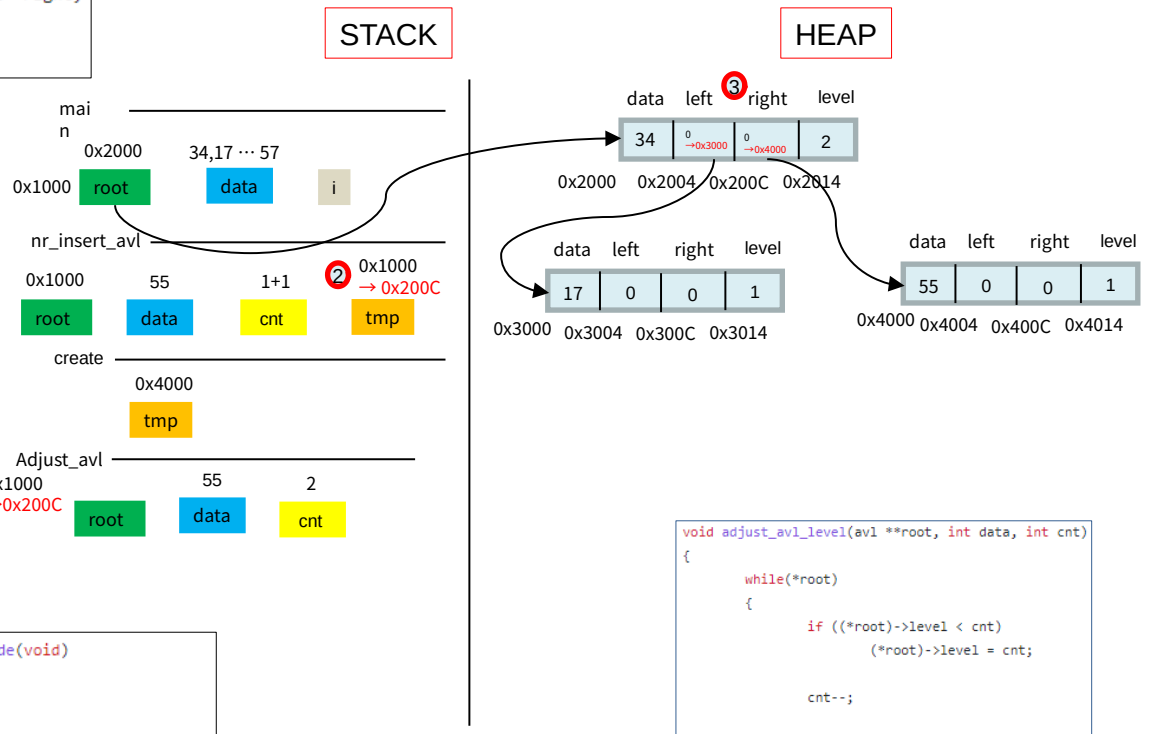
2-3) Bin_tree level 체크

```c
int main(void)
{
	int i;
	avl *root = NULL;
	int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };

	for (i = 0; i < 8; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	int level;
};
```

```c
void nr_insert_avl_data(avl **root, int data)
{
	① avl **tmp = root;
	int cnt = 1;

	while(*tmp)
	{
		cnt++;

		if((*tmp)->data > data)
			tmp = &(*tmp)->left;
		② else if((*tmp)->data < data)
			tmp = &(*tmp)->right;
	}

	③ *tmp = create_avl_node();
	(*tmp)->data = data;

	adjust_avl_level(root, data, cnt);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->level = 1;

	return tmp;
}
```




```c
void adjust_avl_level(avl **root, int data, int cnt)
{
	while(*root)
	{
		if ((*root)->level < cnt)
			(*root)->level = cnt;

		cnt--;

		if((*root)->data > data)
			root = &(*root)->left;
		④ else if((*root)->data < data)
			root = &(*root)->right;
		else
			break;
	}
}
```

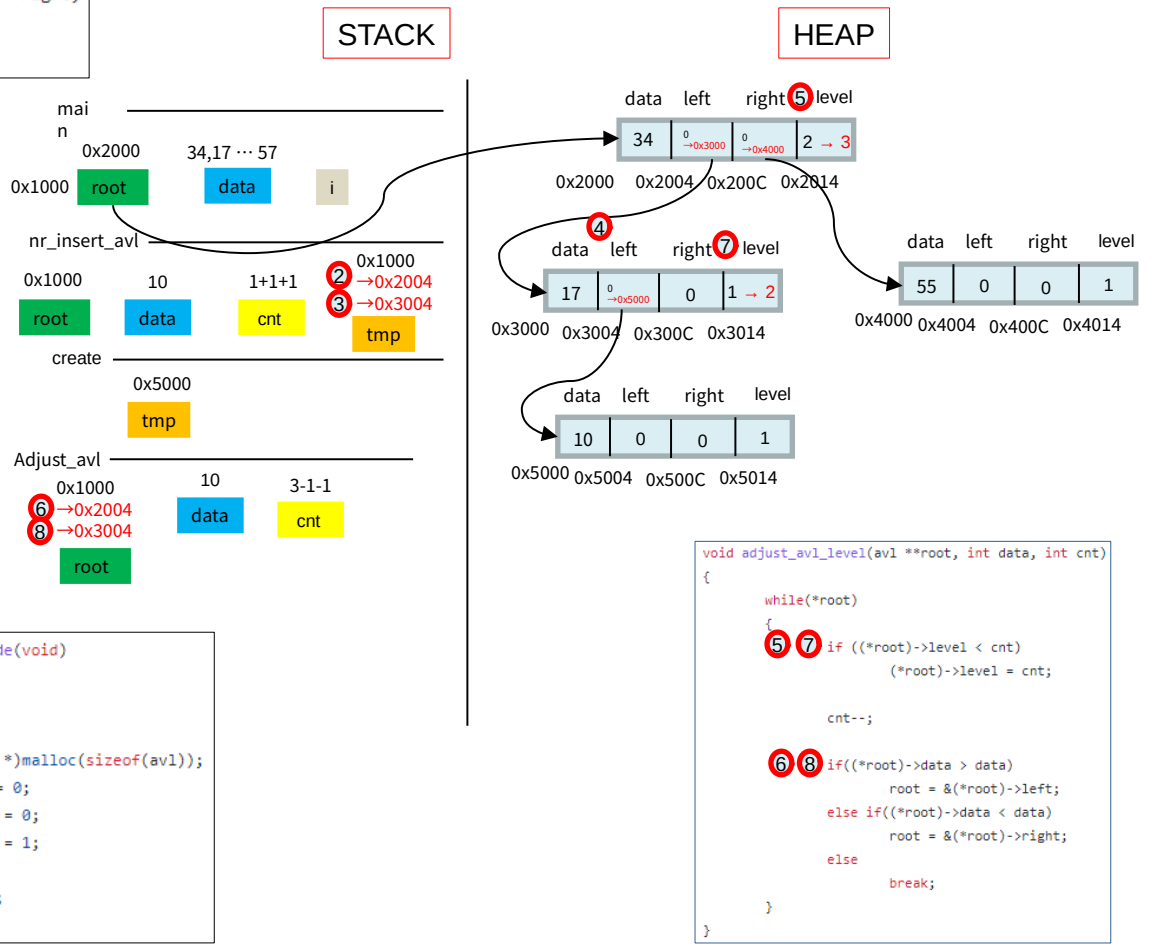
2-4) Bin_tree level 체크

```c
int main(void)
{
	int i;
	avl *root = NULL;
	int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };

	for (i = 0; i < 8; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	int level;
};
```

```c
void nr_insert_avl_data(avl **root, int data)
{
	① avl **tmp = root;
	int cnt = 1;

	while(*tmp)
	{
		cnt++;

		② ③ if((*tmp)->data > data)
			tmp = &(*tmp)->left;
		else if((*tmp)->data < data)
			tmp = &(*tmp)->right;
	}

	④ *tmp = create_avl_node();
	(*tmp)->data = data;

	adjust_avl_level(root, data, cnt);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->level = 1;

	return tmp;
}
```




```c
void adjust_avl_level(avl **root, int data, int cnt)
{
	while(*root)
	{
		⑤ ⑦ if ((*root)->level < cnt)
			(*root)->level = cnt;

		cnt--;

		⑥ ⑧ if((*root)->data > data)
			root = &(*root)->left;
		else if((*root)->data < data)
			root = &(*root)->right;
		else
			break;
	}
}
```

2-5) Bin_tree level 체크

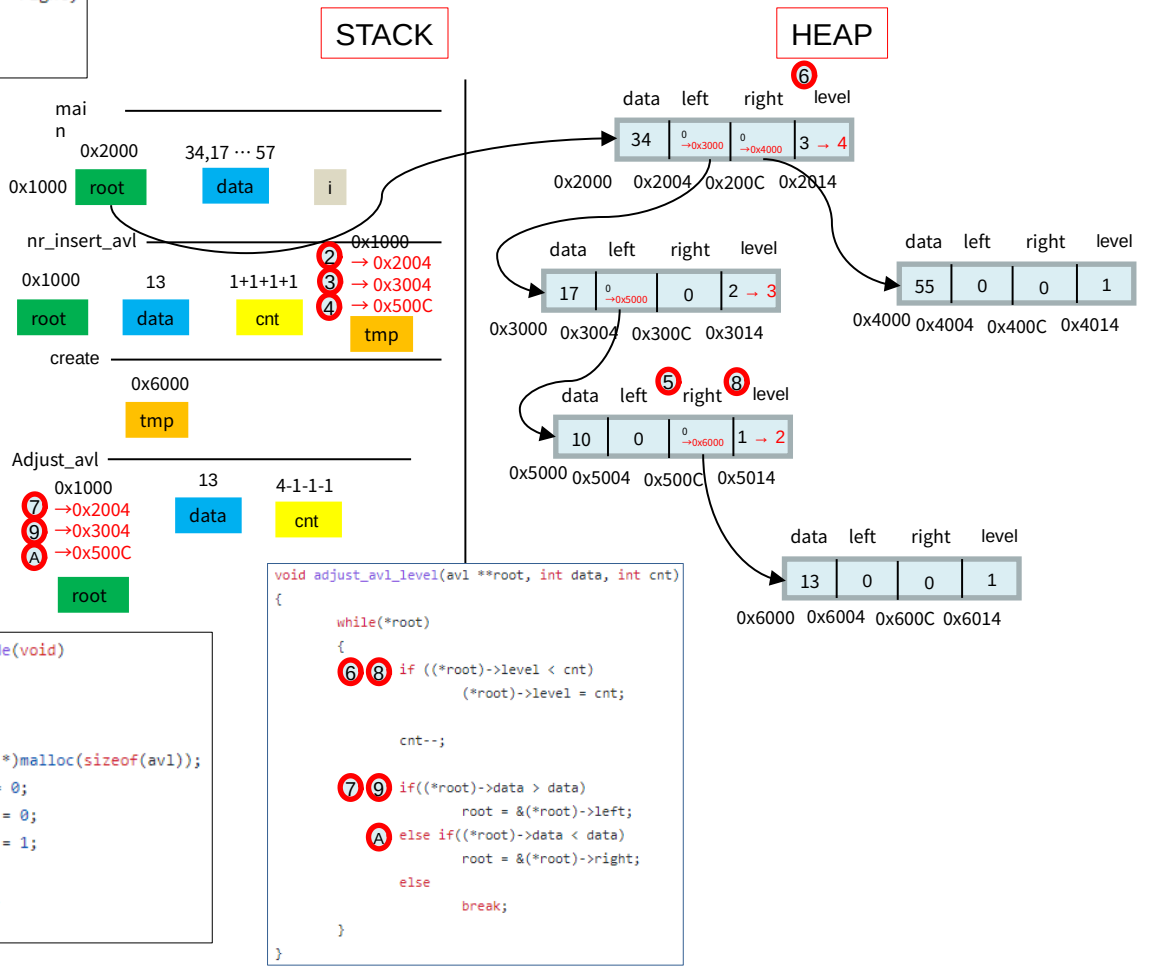
```c
int main(void)
{
	int i;
	avl *root = NULL;
	int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };

	for (i = 0; i < 8; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	int level;
};
```

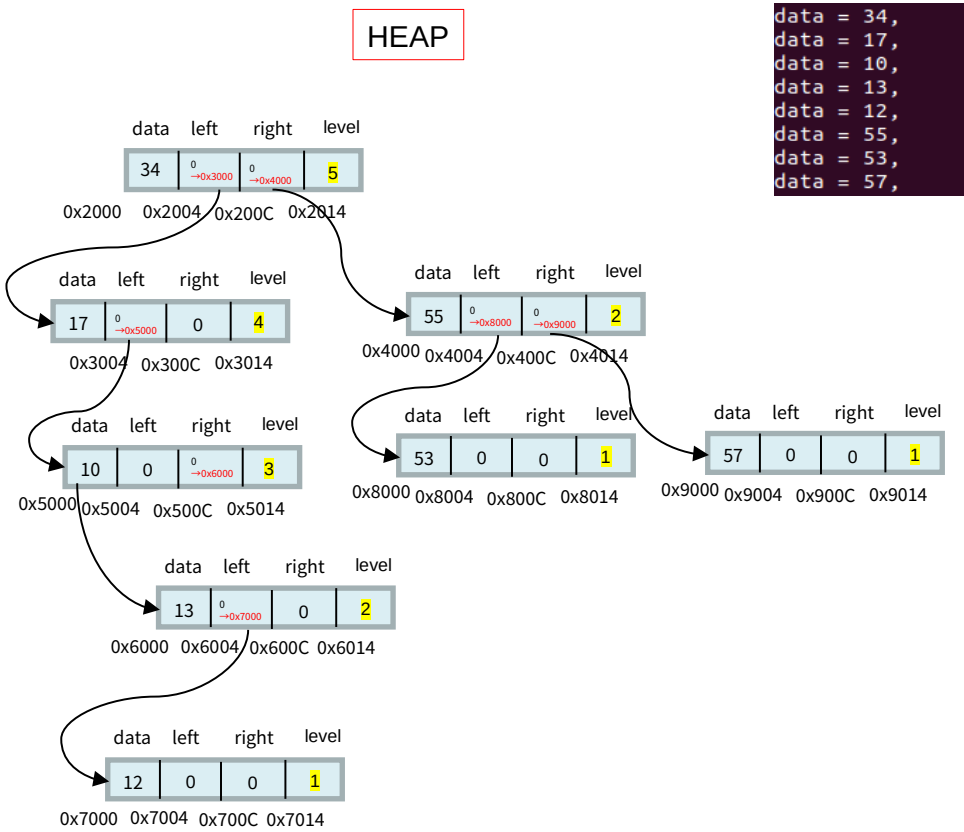
```c
void nr_insert_avl_data(avl **root, int data)
{
	1 avl **tmp = root;
	  int cnt = 1;

	  while(*tmp)
	  {
		cnt++;

	    2 3 if((*tmp)->data > data)
			tmp = &(*tmp)->left;
		4 else if((*tmp)->data < data)
			tmp = &(*tmp)->right;
	  }

	5 *tmp = create_avl_node();
	  (*tmp)->data = data;

	  adjust_avl_level(root, data, cnt);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->level = 1;

	return tmp;
}
```

```c
void adjust_avl_level(avl **root, int data, int cnt)
{
	while(*root)
	{
	6 8 if ((*root)->level < cnt)
			(*root)->level = cnt;

		cnt--;

	7 9 if((*root)->data > data)
			root = &(*root)->left;
	  A else if((*root)->data < data)
			root = &(*root)->right;
		else
			break;
	}
}
```



3) Parent를 활용한 양방향 순회방식 레벨 체크[Lv_par_tree.c]




```
data = 34,      level = 5,      parent = NULL   left = 17,      right = 55
data = 17,      level = 4,      parent = 34     left = 10,      right = NULL
data = 10,      level = 3,      parent = 17     left = NULL,    right = 13
data = 13,      level = 2,      parent = 10     left = 12,      right = NULL
data = 12,      level = 1,      parent = 13     left = NULL,    right = NULL
data = 55,      level = 2,      parent = 34     left = 53,      right = 57
data = 53,      level = 1,      parent = 55     left = NULL,    right = NULL
data = 57,      level = 1,      parent = 55     left = NULL,    right = NULL
```

전략 :

레벨링시 앞 노드의 주소 정보를 가지고 있는 Parent를 활용하여

Parent를 거칠 때 마다 상위로 갈 수록 레벨링을 +1씩 증가

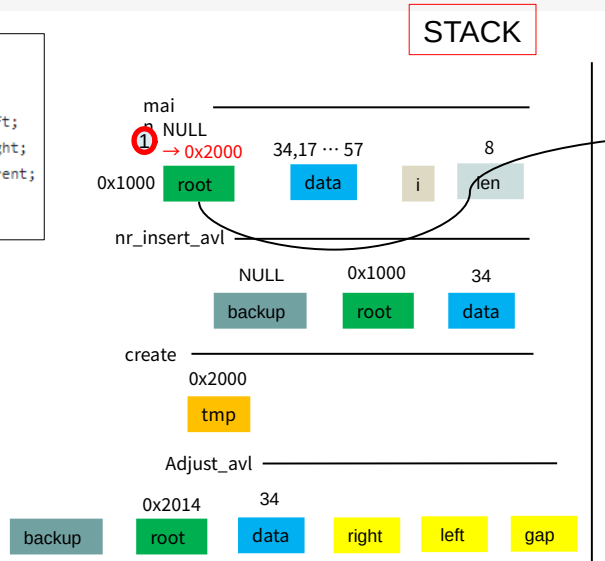
3-1) Bin_tree level 체크(Parent 활용 양방향 순회 방식)

```c
int main(void)
{
	int i;
	avl *root = NULL;
	//int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };
	//int data[] = { 34, 17, 55, 10, 13 };
	int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	int len = sizeof(data) / sizeof(int);

	for (i = 0; i < len; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	struct _avl *parent;
	int level;
};
```

```c
void nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;

	while(*root)
	{
		backup = *root;
		if((*root)->data > data)
			root = &(*root)->left;
		else if((*root)->data < data)
			root = &(*root)->right;
	}

	① *root = create_avl_node();
	(*root)->data = data;
	(*root)->parent = backup;

	② adjust_avl_level(&(*root)->parent, data);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->parent = 0;
	tmp->level = 1;

	return tmp;
}
```



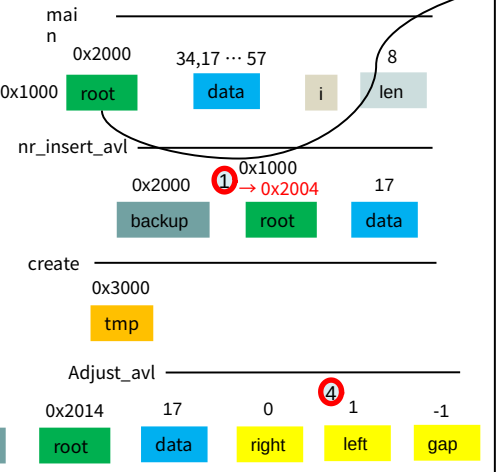

```c
void adjust_avl_level(avl **root, int data)
{
	int right, left, gap;
	avl **backup;

	② while(*root)
	{
		if ((*root)->right)
			right = (*root)->right->level;
		else
			right = 0;

		if ((*root)->left)
			left = (*root)->left->level;
		else
			left = 0;

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, decision_rotation(backup, right - left, data));

		if (right > left)
			(*root)->level = right + 1;
		else
			(*root)->level = left + 1;

		backup = root;
		root = &(*root)->parent;
	}
}
```

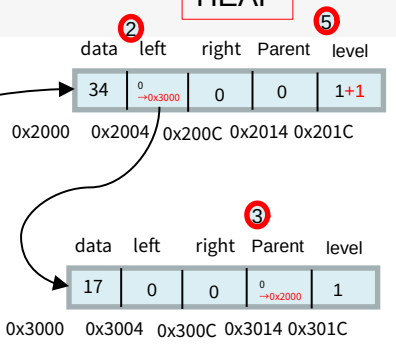
Parent가 NULL 이기에 while 문 들어가지 않음

3-2) Bin_tree level 체크(Parent 활용 양방향 순회 방식)

```c
int main(void)
{
	int i;
	avl *root = NULL;
	//int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };
	//int data[] = { 34, 17, 55, 10, 13 };
	int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	int len = sizeof(data) / sizeof(int);

	for (i = 0; i < len; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	struct _avl *parent;
	int level;
};
```






```c
void nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;

	while(*root)
	{
		backup = *root;
		if((*root)->data > data)
			1 root = &(*root)->left;
		else if((*root)->data < data)
			root = &(*root)->right;
	}

	2 *root = create_avl_node();
	(*root)->data = data;
	3 (*root)->parent = backup;

	adjust_avl_level(&(*root)->parent, data);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->parent = 0;
	tmp->level = 1;

	return tmp;
}
```

```c
void adjust_avl_level(avl **root, int data)
{
	int right, left, gap;
	avl **backup;

	while(*root)
	{
		if ((*root)->right)
			right = (*root)->right->level;
		else
			right = 0;

		4 if ((*root)->left)
			left = (*root)->left->level;
		else
			left = 0;

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, decision_rotation(backup, right - left, data));

		if (right > left)
			(*root)->level = right + 1;
		5 else
			(*root)->level = left + 1;

		backup = root;
		root = &(*root)->parent;
	}
}
```

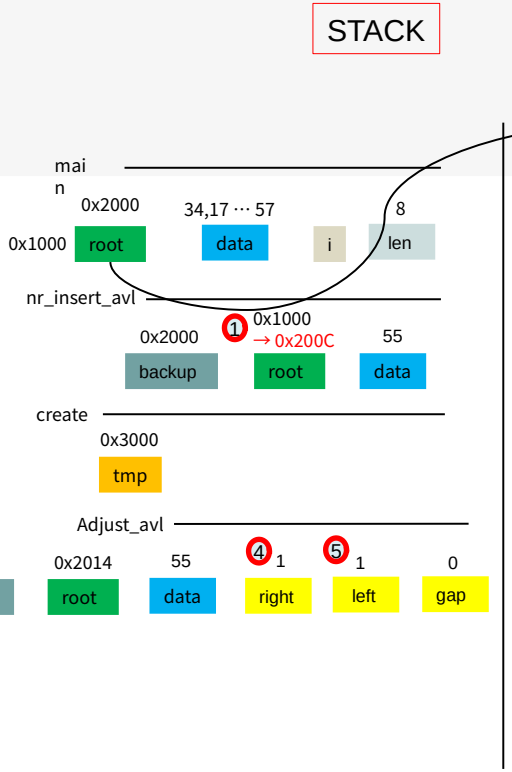
3-3) Bin_tree level 체크(Parent 활용 양방향 순회 방식)

```c
int main(void)
{
	int i;
	avl *root = NULL;
	//int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };
	//int data[] = { 34, 17, 55, 10, 13 };
	int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	int len = sizeof(data) / sizeof(int);

	for (i = 0; i < len; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	struct _avl *parent;
	int level;
};
```

```c
void nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;

	while(*root)
	{
		backup = *root;
		if((*root)->data > data)
			root = &(*root)->left;
		else if((*root)->data < data)
		    ① root = &(*root)->right;
	}

    ② *root = create_avl_node();
	(*root)->data = data;
    ③ (*root)->parent = backup;

	adjust_avl_level(&(*root)->parent, data);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->parent = 0;
	tmp->level = 1;

	return tmp;
}
```




```c
void adjust_avl_level(avl **root, int data)
{
	int right, left, gap;
	avl **backup;

	while(*root)
	{
		if ((*root)->right)
		    ④ right = (*root)->right->level;
		else
			right = 0;

		if ((*root)->left)
		    ⑤ left = (*root)->left->level;
		else
			left = 0;

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, decision_rotation(backup, right - left, data));

		if (right > left)
			(*root)->level = right + 1;
		else
		    ⑥ (*root)->level = left + 1;

		backup = root;
		root = &(*root)->parent;
	}
}
```

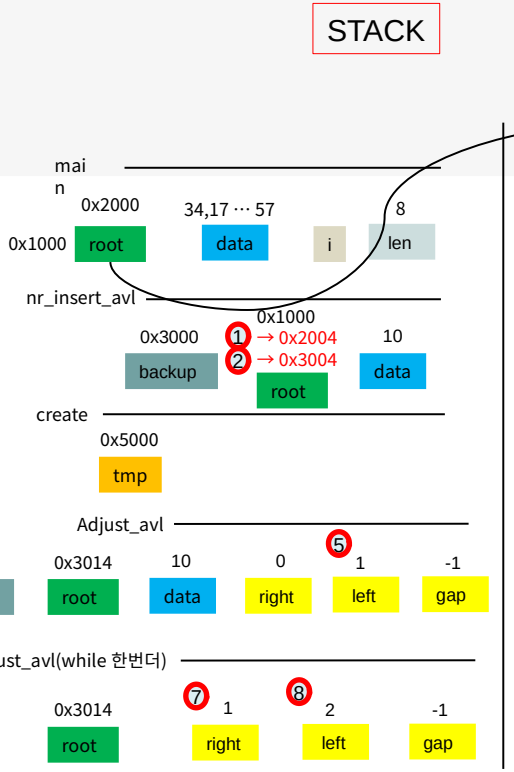
3-4) Bin_tree level 체크(Parent 활용 양방향 순회 방식)

```c
int main(void)
{
	int i;
	avl *root = NULL;
	//int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };
	//int data[] = { 34, 17, 55, 10, 13 };
	int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	int len = sizeof(data) / sizeof(int);

	for (i = 0; i < len; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	struct _avl *parent;
	int level;
};
```

```c
void nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;

	while(*root)
	{
		backup = *root;
		if((*root)->data > data)
			root = &(*root)->left;
		else if((*root)->data < data)
			root = &(*root)->right;
	}

	*root = create_avl_node();
	(*root)->data = data;
	(*root)->parent = backup;

	adjust_avl_level(&(*root)->parent, data);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->parent = 0;
	tmp->level = 1;

	return tmp;
}
```




```c
void adjust_avl_level(avl **root, int data)
{
	int right, left, gap;
	avl **backup;

	while(*root)
	{
		if ((*root)->right)
			right = (*root)->right->level;
		else
			right = 0;

		if ((*root)->left)
			left = (*root)->left->level;
		else
			left = 0;

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, decision_rotation(backup, right - left, data));

		if (right > left)
			(*root)->level = right + 1;
		else
			(*root)->level = left + 1;

		backup = root;
		root = &(*root)->parent;
	}
}
```

3-5) Bin_tree level 체크(Parent 활용 양방향 순회 방식)

```c
int main(void)
{
	int i;
	avl *root = NULL;
	//int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };
	//int data[] = { 34, 17, 55, 10, 13 };
	int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	int len = sizeof(data) / sizeof(int);

	for (i = 0; i < len; i++)
		nr_insert_avl_data(&root, data[i]);

	print_tree(root);

	return 0;
}
```

```c
struct _avl
{
	int data;
	struct _avl *left;
	struct _avl *right;
	struct _avl *parent;
	int level;
};
```

```c
void nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;

	while(*root)
	{
		backup = *root;
		if((*root)->data > data)
		 (1) (2) root = &(*root)->left;
		else if((*root)->data < data)
		     (3) root = &(*root)->right;
	}

    (4) *root = create_avl_node();
	(*root)->data = data;
    (5) (*root)->parent = backup;

	adjust_avl_level(&(*root)->parent, data);
}
```

```c
avl *create_avl_node(void)
{
	avl *tmp;

	tmp = (avl *)malloc(sizeof(avl));
	tmp->left = 0;
	tmp->right = 0;
	tmp->parent = 0;
	tmp->level = 1;

	return tmp;
}
```

STACK

main

0x2000 root | 34,17 ··· 57 data | i | 8 len

nr_insert_avl

0x1000 backup | 0x2000 data | 13

① → 0x2004
② → 0x3004
③ → 0x500C

root

create

0x6000 tmp

Adjust_avl

0x6014 backup	0x5014 root	13 data	6 1 right	0 left	1 gap

Adjust_avl(while 한번더)

0x5014 backup	0x3014 root	0 right	8 2 left	9 -2 gap

Adjust_avl(while 한번더)

0x3014 backup	0x2014 root	B 1 right	C 3 left	9 -2 gap

HEAP

	data	left	right	Parent	d level
0x2000	34	0 →0x3000	0 →0x4000	0	3+1

	data	left	right	Parent	B level
0x4000	55	0	0	0 →0x2000	1

	data	left	right	Parent	A C level
0x3000	17	0 →0x4000	0	0 →0x2000	2+1

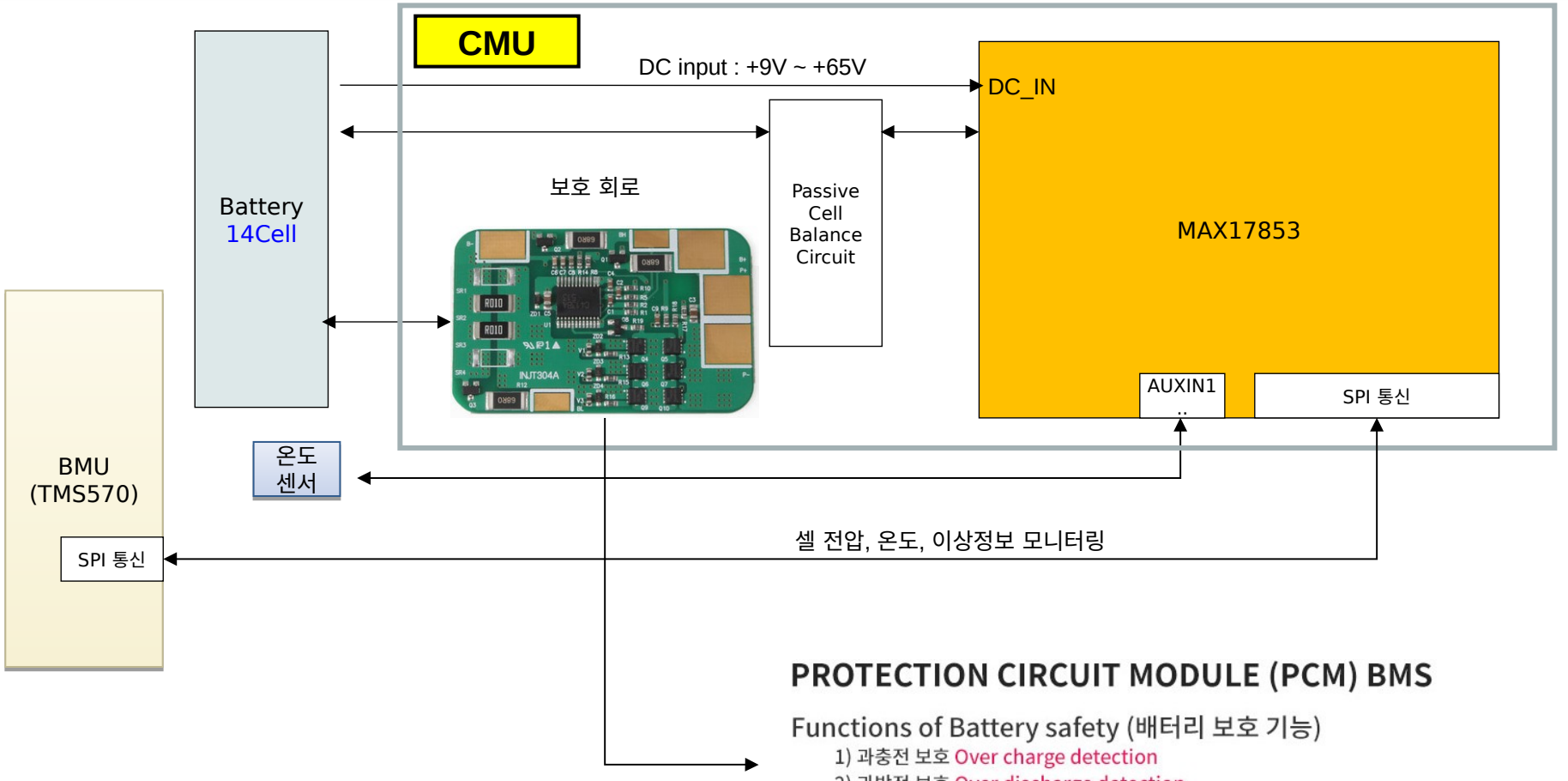
	data	left	4 right	Parent	7 8 level
0x5000	10	0	0 →0x6000	0 →0x3000	1 +1

	data	left	right	5 Parent	6 level
0x6000	13	0	0	0 →0x5000	1

```c
void adjust_avl_level(avl **root, int data)
{
	int right, left, gap;
	avl **backup;

	while(*root)
	{
		if ((*root)->right)
		    (6) right = (*root)->right->level; (B)
		else
			right = 0;

		if ((*root)->left)
		    (8) left = (*root)->left->level; (C)
		else
			left = 0;

		gap = right - left;

		(9) if (ABS(gap) > 1)
			rotation(root, decision_rotation(backup, right - left, data));

		if (right > left)
		    (7) (*root)->level = right + 1;
		else
		    (A) (*root)->level = left + 1; (d)

		backup = root;
		root = &(*root)->parent;
	}
}
```

회전 필요

BMS Block Diagram



PROTECTION CIRCUIT MODULE (PCM) BMS

Functions of Battery safety (배터리 보호 기능)

1) 과충전 보호 Over charge detection
2) 과방전 보호 Over discharge detection
3) 과전류 보호 Over current detection
4) 쇼트(단락) 보호 Short detection
5) 셀 발란싱 기능 Cell Balance Function

- 보호 회로 분석 필요.
- 분석 후 필요한 만큼 확장 설계

https://lunavolt.com/product/%EA%B5%AD%EC%82%B0-3%EC%85%80-111v3s-12v-15a-%EB%A6%AC%ED%8A%AC%EC%9D%B4%EC%98%A8-%EB%B3%B4%ED%98%B8%ED%9A%8C%EB%A1%9C-bms-%EC%85%80%EB%B0%9C%EB%9E%80%EC%8B%B1/108/