EDDI
Electronic Design
Development Institute

에디로봇아카데미
임베디드 마스터 Lv2 과정
[자료구조 프로그래밍]

제 1기

2021. 12. 03

박태인

목차

1) AVL 트리 LR 회전 이해하기 [lv_par_rot_tree.c]
2) AVL 트리 insert 완성본 이해하기 [nr_avl_insert.c]
   ㄴ init_data 에 난수 생성을 하지 않고, 데이터를 직접 넣어서 rotation 의 원리를 파악해 보자.
   ㄴ 이거 2번 일단은 최종 insert가 있으니까 그 내용을 한번 참고 해보자고

- 추가 도전 미션 –

1) 이중 연결 리스트 구현해 보기
2) print_data(node. 30) 함수 구현해 보기
   ㄴ 연결리스트 되어 있는 정보들을 30회 print 한다는 의미의 함수

[현상황]

1번의 LR회전을 하기위한 판단 방법이 맞는지 잘 모르겠습니다.
그와 함께 회전에서 헤매고 있는 것 같습니다..

이를 극복 하고 나면 추후 2번의 insert.c의 다른 회전들과 함께 더 이해해 나가야 할 것 같습니다.

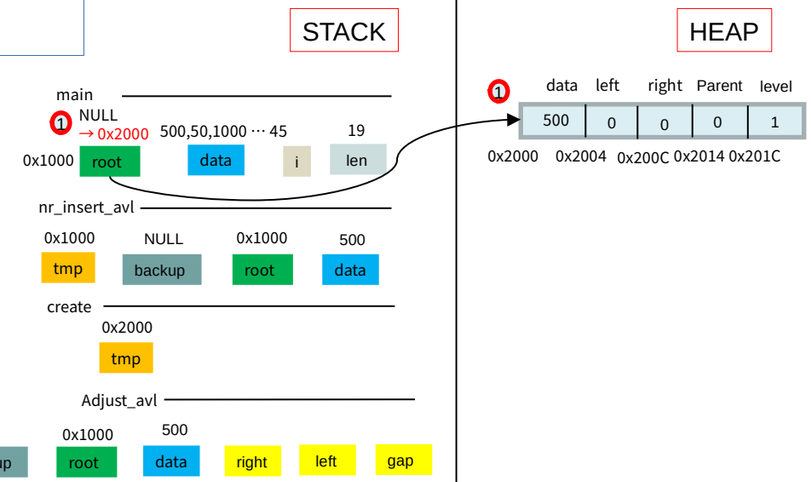
추가 도전 미션은 그 이후가 될 것 같습니다.

4-1) AVL 트리 LR 회전 이해하기

```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 };

        // 제어노드가 root인 복잡한 LR
        //int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
        // 단순한 LR
        //int data[] = { 5, 3, 4 };
        // 제어노드가 root가 아닌 복잡한 LR
        int data[] = { 500, 50, 1000, 100, 25, 750, 1250, 75, 125, 37, 12, 625, 875, 1125, 1375, 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
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 nr_insert_avl_data(avl **root, int data)
{
        avl *backup = NULL;
#else
        avl **tmp = root;

        while (*tmp)
        {
                //backup = *root;
                backup = *tmp;

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

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

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



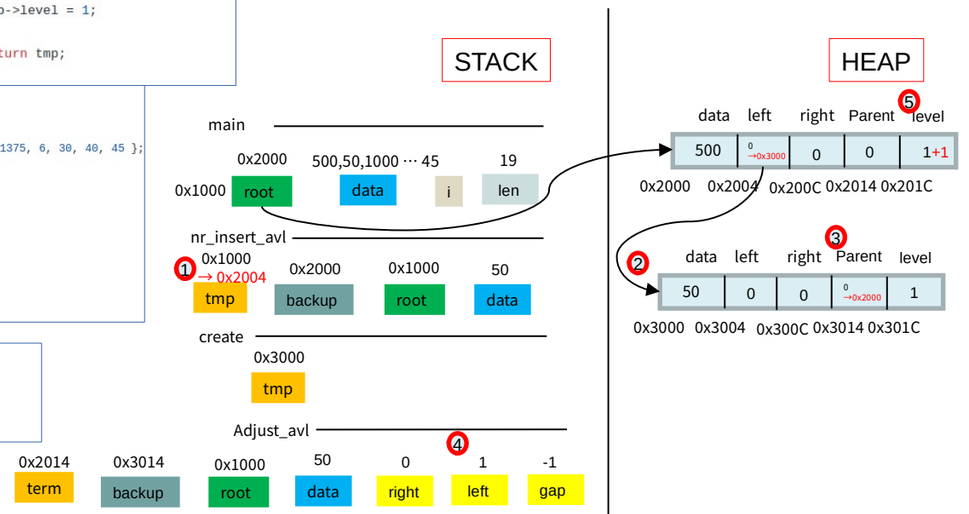

```c
void adjust_avl_level(avl **root, avl **term_node, int data)
{
        int right, left, gap;
        avl **backup;

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

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

                gap = right - left;

                if (ABS(gap) > 1)
                        rotation(root, term_node, decision_rotation(backup, right - left, data));
                else if (right > left)
                        (*term_node)->level = right + 1;
                else
                        (*term_node)->level = left + 1;

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

4-2) AVL 트리 LR 회전 이해하기

```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 };

	// 제어노드가 root인 복잡한 LR
	//int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	// 단순한 LR
	//int data[] = { 5, 3, 4 };
	// 제어노드가 root가 아닌 복잡한 LR
	int data[] = { 500, 50, 1000, 100, 25, 750, 1250, 75, 125, 37, 12, 625, 875, 1125, 1375, 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
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 nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;

#else
	avl **tmp = root;

	while (*tmp)
	{
		//backup = *root;
		backup = *tmp;

		if ((*tmp)->data > data)
			① tmp = &(*tmp)->left;
		else if ((*tmp)->data < data)
			tmp = &(*tmp)->right;
	}

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

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



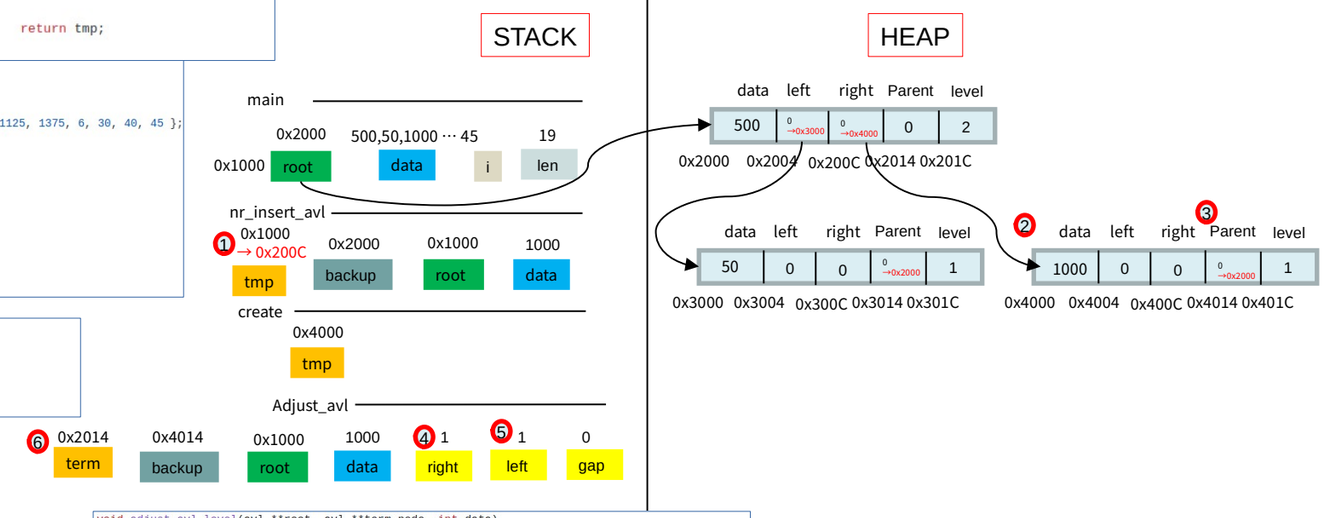

```c
void adjust_avl_level(avl **root, avl **term_node, int data)
{
	int right, left, gap;
	avl **backup;

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

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

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, term_node, decision_rotation(backup, right - left, data));
		else if (right > left)
			(*term_node)->level = right + 1;
		else
			⑤ (*term_node)->level = left + 1;

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

4-3) AVL 트리 LR 회전 이해하기

```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 };

	// 제어노드가 root인 복잡한 LR
	//int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	// 단순한 LR
	//int data[] = { 5, 3, 4 };
	// 제어노드가 root가 아닌 복잡한 LR
	int data[] = { 500, 50, 1000, 100, 25, 750, 1250, 75, 125, 37, 12, 625, 875, 1125, 1375, 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
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 nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;
#else
	avl **tmp = root;

	while (*tmp)
	{
		//backup = *root;
		backup = *tmp;

		if ((*tmp)->data > data)
			tmp = &(*tmp)->left;
		else if ((*tmp)->data < data)
		① tmp = &(*tmp)->right;
	}

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

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



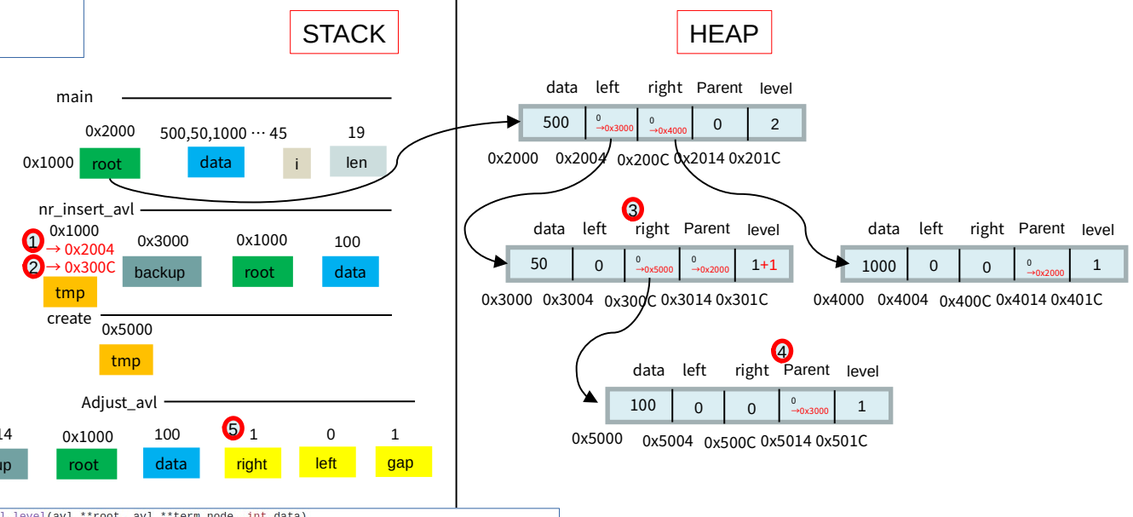

```c
void adjust_avl_level(avl **root, avl **term_node, int data)
{
	int right, left, gap;
	avl **backup;

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

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

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, term_node, decision_rotation(backup, right - left, data));
		else if (right > left)
			(*term_node)->level = right + 1;
		else
			(*term_node)->level = left + 1;

		backup = term_node;
	⑥ term_node = &(*term_node)->parent;
	}
}
```

4-4) AVL 트리 LR 회전 이해하기

```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 };

	// 제어노드가 root인 복잡한 LR
	//int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	// 단순한 LR
	//int data[] = { 5, 3, 4 };
	// 제어노드가 root가 아닌 복잡한 LR
	int data[] = { 500, 50, 1000, 100, 25, 750, 1250, 75, 125, 37, 12, 625, 875, 1125, 1375, 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
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 nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;
#else
	avl **tmp = root;

	while (*tmp)
	{
		//backup = *root;
		backup = *tmp;

		if ((*tmp)->data > data)
			① tmp = &(*tmp)->left;
		else if ((*tmp)->data < data)
			② tmp = &(*tmp)->right;
	}

	③ *tmp = create_avl_node();
	(*tmp)->data = data;
	④ (*tmp)->parent = backup;

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



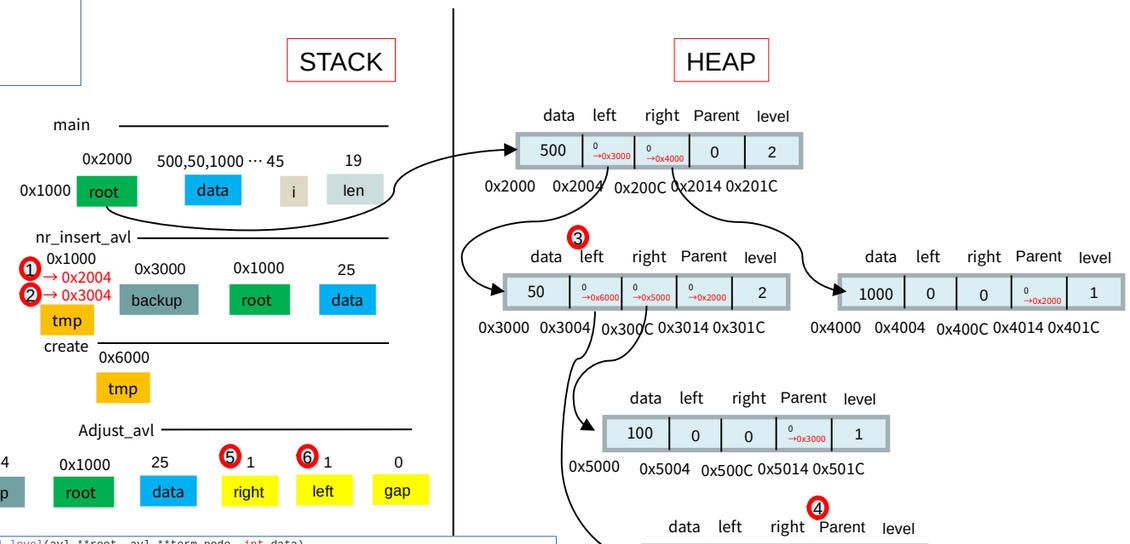

```c
void adjust_avl_level(avl **root, avl **term_node, int data)
{
	int right, left, gap;
	avl **backup;

	while(*term_node)
	{
		if ((*term_node)->right)
			⑤ right = (*term_node)->right->level;
		else
			right = 0;

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

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, term_node, decision_rotation(backup, right - left, data));
		else if (right > left)
			⑥ (*term_node)->level = right + 1;
		else
			(*term_node)->level = left + 1;

		backup = term_node;
		⑦ term_node = &(*term_node)->parent;
	}
}
```

4-5) AVL 트리 LR 회전 이해하기

```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 };

	// 제어노드가 root인 복잡한 LR
	//int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	// 단순한 LR
	//int data[] = { 5, 3, 4 };
	// 제어노드가 root가 아닌 복잡한 LR
	int data[] = { 500, 50, 1000, 100, 25, 750, 1250, 75, 125, 37, 12, 625, 875, 1125, 1375, 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
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 nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;

#else
	avl **tmp = root;

	while (*tmp)
	{
		//backup = *root;
		backup = *tmp;

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

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

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



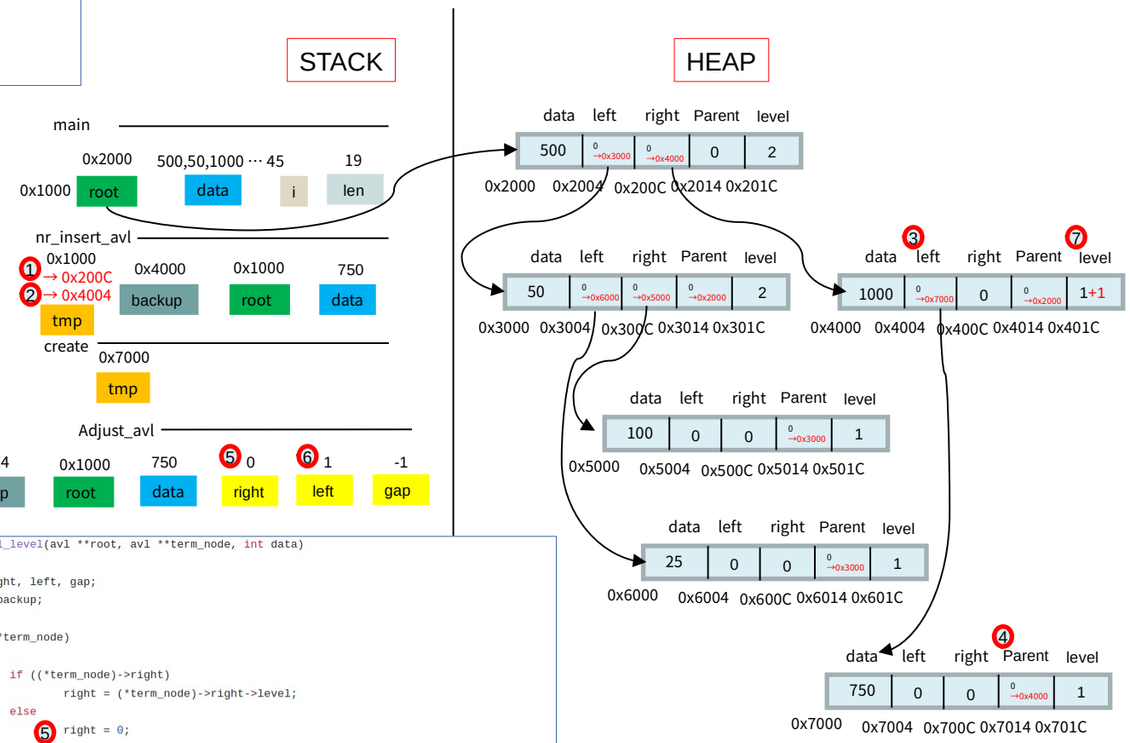

```c
void adjust_avl_level(avl **root, avl **term_node, int data)
{
	int right, left, gap;
	avl **backup;

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

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

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, term_node, decision_rotation(backup, right - left, data));
		else if (right > left)
			(*term_node)->level = right + 1;
		else
			(*term_node)->level = left + 1;

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

4-6) AVL 트리 LR 회전 이해하기

```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 };

	// 제어노드가 root인 복잡한 LR
	//int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	// 단순한 LR
	//int data[] = { 5, 3, 4 };
	// 제어노드가 root가 아닌 복잡한 LR
	int data[] = { 500, 50, 1000, 100, 25, 750, 1250, 75, 125, 37, 12, 625, 875, 1125, 1375, 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
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 nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;
#else
	avl **tmp = root;

	while (*tmp)
	{
		//backup = *root;
		backup = *tmp;

		if ((*tmp)->data > data)
			② tmp = &(*tmp)->left;
		else if ((*tmp)->data < data)
			① tmp = &(*tmp)->right;
	}

	③ *tmp = create_avl_node();
	(*tmp)->data = data;
	④ (*tmp)->parent = backup;

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



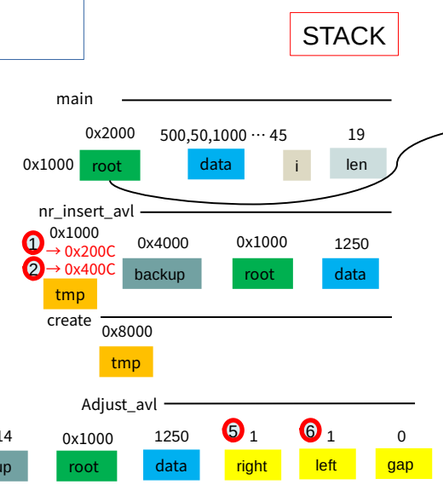

```c
void adjust_avl_level(avl **root, avl **term_node, int data)
{
	int right, left, gap;
	avl **backup;

	while(*term_node)
	{
		if ((*term_node)->right)
			right = (*term_node)->right->level;
		else
			⑤ right = 0;

		if ((*term_node)->left)
			⑥ left = (*term_node)->left->level;
		else
			left = 0;

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, term_node, decision_rotation(backup, right - left, data));
		else if (right > left)
			(*term_node)->level = right + 1;
		else
			⑦ (*term_node)->level = left + 1;

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

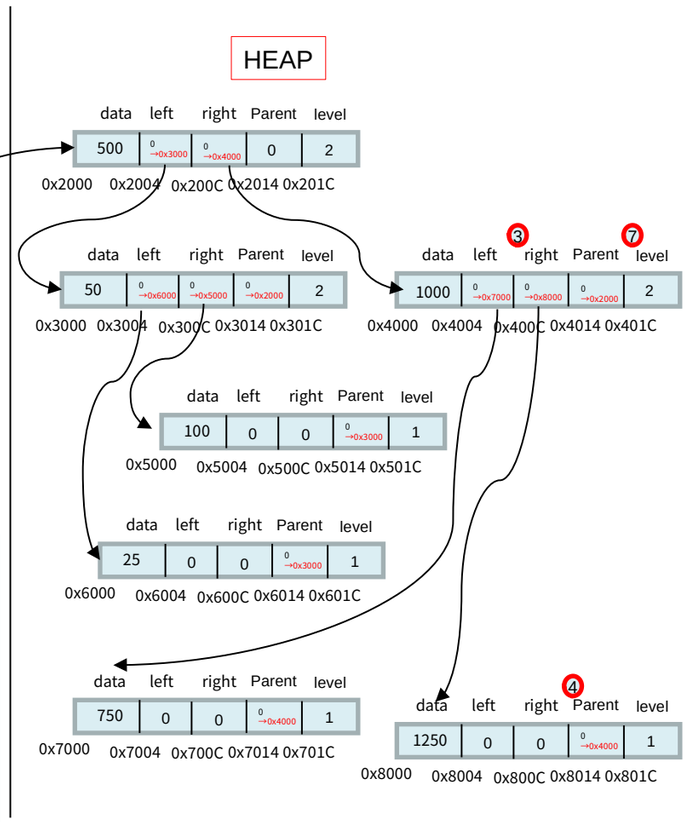
4-7) AVL 트리 LR 회전 이해하기

```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 };

	// 제어노드가 root인 복잡한 LR
	//int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	// 단순한 LR
	//int data[] = { 5, 3, 4 };
	// 제어노드가 root가 아닌 복잡한 LR
	int data[] = { 500, 50, 1000, 100, 25, 750, 1250, 75, 125, 37, 12, 625, 875, 1125, 1375, 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
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 nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;

#else
	avl **tmp = root;

	while (*tmp)
	{
		//backup = *root;
		backup = *tmp;

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

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

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





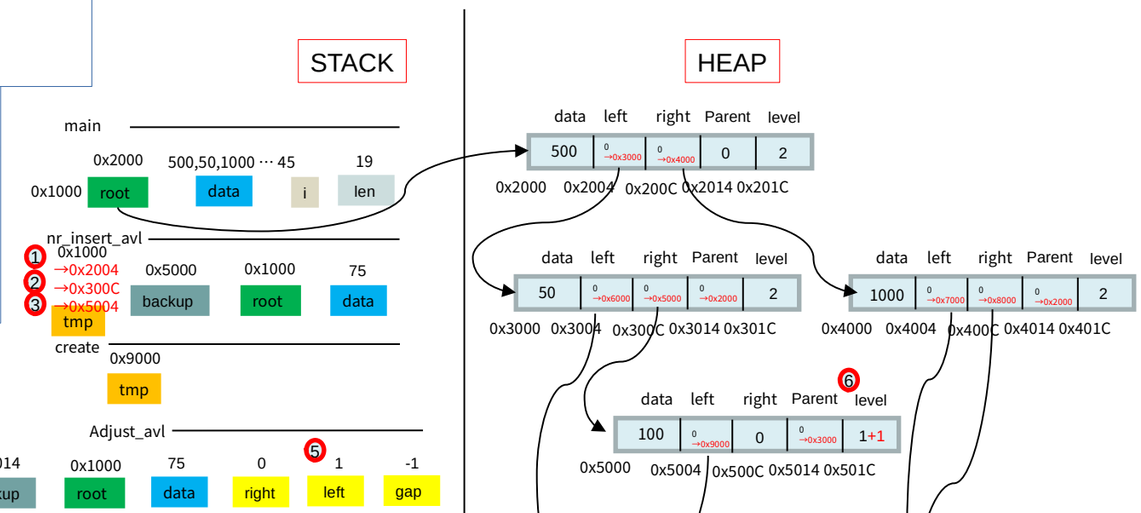

```c
void adjust_avl_level(avl **root, avl **term_node, int data)
{
	int right, left, gap;
	avl **backup;

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

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

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, term_node, decision_rotation(backup, right - left, data));
		else if (right > left)
			(*term_node)->level = right + 1;
		else
			(*term_node)->level = left + 1;

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

4-8) AVL 트리 LR 회전 이해하기

```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 };

	// 제어노드가 root인 복잡한 LR
	//int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	// 단순한 LR
	//int data[] = { 5, 3, 4 };
	// 제어노드가 root가 아닌 복잡한 LR
	int data[] = { 500, 50, 1000, 100, 25, 750, 1250, 75, 125, 37, 12, 625, 875, 1125, 1375, 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
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 nr_insert_avl_data(avl **root, int data)
{
	avl *backup = NULL;
#else
	avl **tmp = root;

	while (*tmp)
	{
		//backup = *root;
		backup = *tmp;

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

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

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




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

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

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

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, term_node, decision_rotation(backup, right - left, data));
		else if (right > left)
			(*term_node)->level = right + 1;
		else
			(*term_node)->level = left + 1;

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

4-8) AVL 트리 LR 회전 이해하기

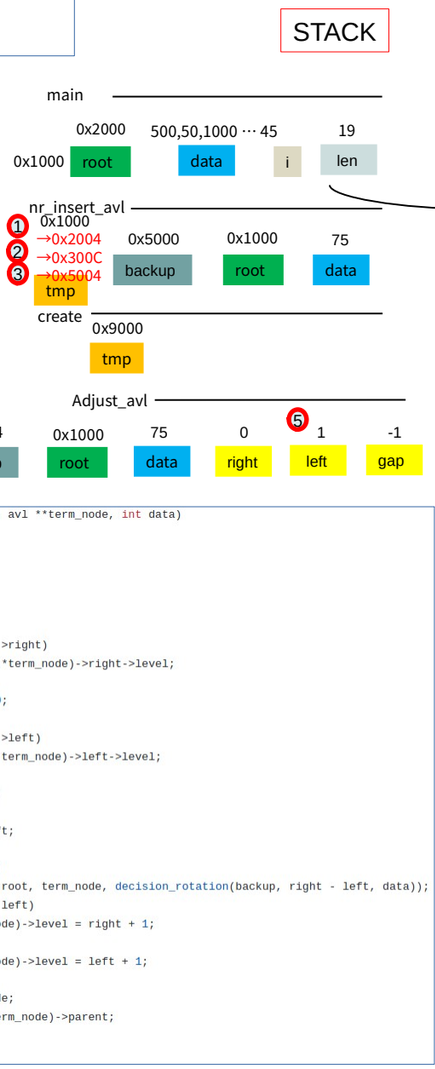
```c
int main(void)
{
	int i;
	avl *root = NULL;
	//int data[] = { 34, 17, 55, 10, 13, 12, 53, 57 };
	//int data[] = { 34, 17, 55, 10, 13 };

	// 제어노드가 root인 복잡한 LR
	//int data[] = { 50, 100, 25, 75, 125, 37, 12, 6, 30, 40, 45 };
	// 단순한 LR
	//int data[] = { 5, 3, 4 };
	// 제어노드가 root가 아닌 복잡한 LR
	int data[] = { 500, 50, 1000, 100, 25, 750, 1250, 75, 125, 37, 12, 625, 875, 1125, 1375, 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;
}
```

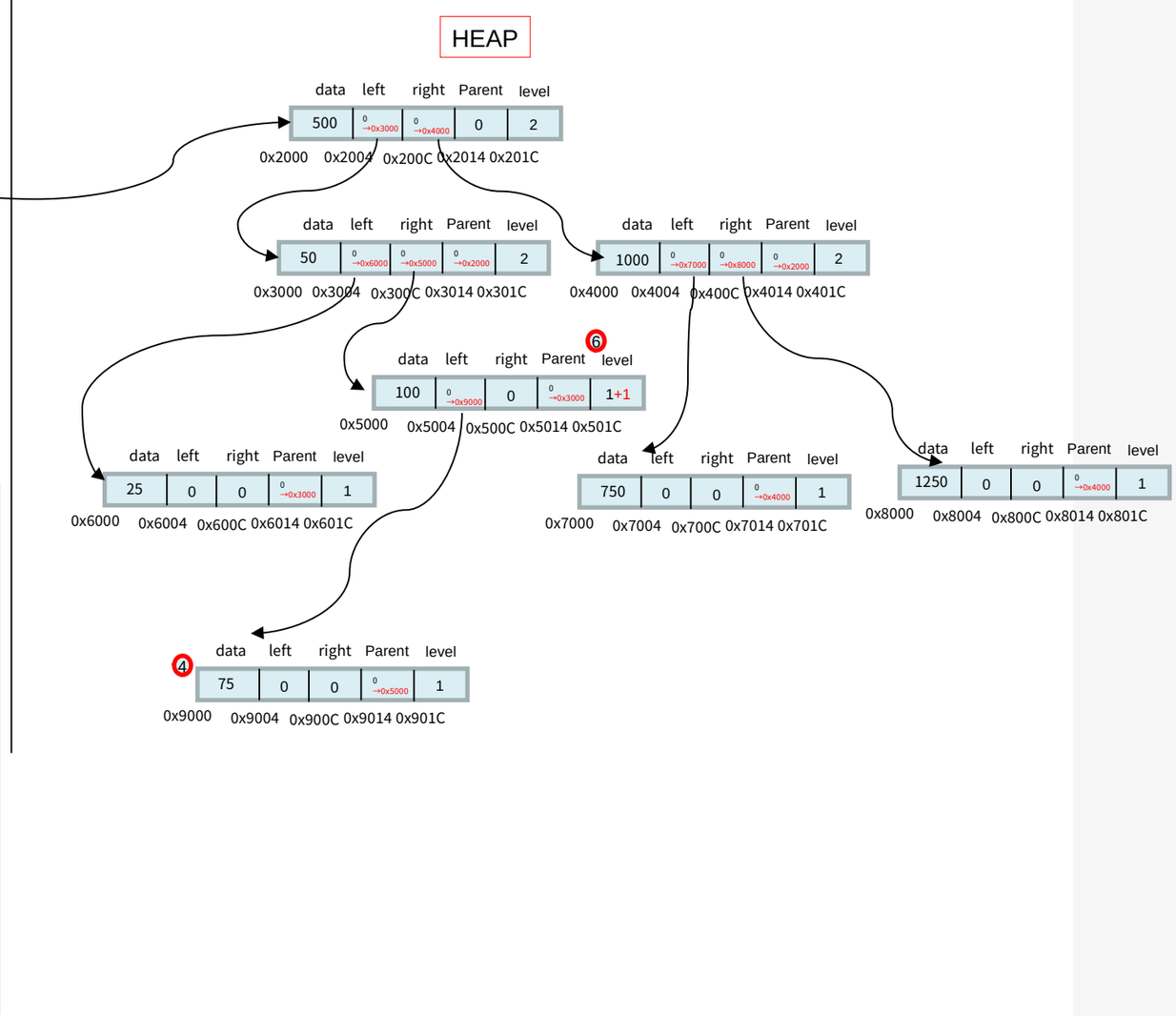
```c
avl *create_avl_node(void)
{
	avl *tmp;

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

	return tmp;
}
```

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

```c
#else
	avl **tmp = root;

	while (*tmp)
	{
		//backup = *root;
		backup = *tmp;

		if ((*tmp)->data > data)
		① ③	tmp = &(*tmp)->left;
		else if ((*tmp)->data < data)
		②	tmp = &(*tmp)->right;
	}

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

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

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

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

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

		gap = right - left;

		if (ABS(gap) > 1)
			rotation(root, term_node, decision_rotation(backup, right - left, data));
		else if (right > left)
			(*term_node)->level = right + 1;
		⑥ else
			(*term_node)->level = left + 1;

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





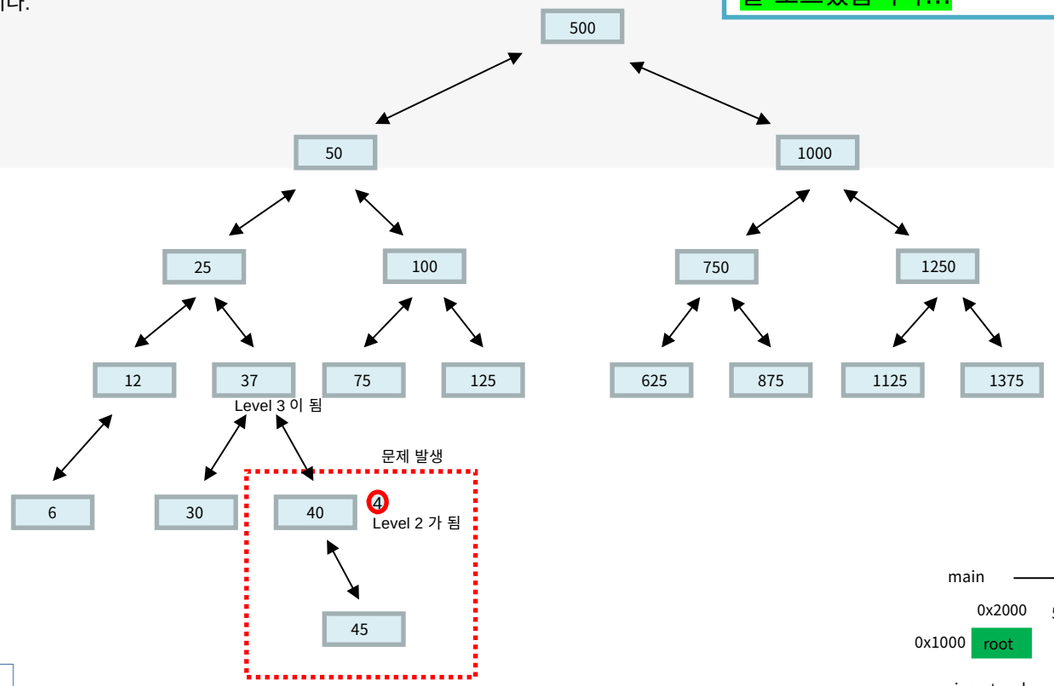
4-11) AVL 트리 LR 회전 이해하기

연결리스트 전체를 앞에서와 같이 하려면 내용이 너무 많아져서
우선은 전체적으로 아래와 같이 정리 하였습니다.

앞과 같이 계속해서 연결 리스트를 만들면 아래와 같이 구성 됩니다.

[질문]

LR을 판단하는 과정이 맞는지 잘 모르겠습니다...




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

```
#else
        avl **tmp = root;

        while (*tmp)
        {
                //backup = *root;
                backup = *tmp;

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

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

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

1) Insert가 완료 된 후 레벨 조정 함수 시작 부분을 살펴 보자.

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

        while(*term_node)   ⑥ *Term_node의 값이 존재? -> 37이 존재.
        {
            ① if ((*term_node)->right)
                        right = (*term_node)->right->level;
                else
                        right = 0;

                if ((*term_node)->left)
                        left = (*term_node)->left->level;
            ② else
                        left = 0;

            ③ gap = right - left;

                if (ABS(gap) > 1)
                        rotation(root, term_node, decision_rotation(backup, right - left, data));
                else if (right > left)
                        (*term_node)->level = right + 1;
            ④ else
                        (*term_node)->level = left + 1;

            ⑤ backup = term_node;
                term_node = &(*term_node)->parent;
        }
}
```

Term_node의 -> right는
40의 오른쪽이 존재 하므로, 있고
40의 레벨은 1 이었을 것이므로 right는 1 이 된다.

Term_node의 -> left는
존재하지 않으므로 0 이 된다.

Back up에 45의 partent 주소
Term_node에 45의 partent가 가르키는 40의 parent 주소

```
int decision_rotation(avl **root, int gap, int data)
{
        if (gap == LL)
        {
                if ((*root)->data < data)
                        return LR;

                return LL;
        }
        else    // RR
        {
                if ((*root)->data > data)
                        return RL;

                return RR;
        }
}
```

3) 자 이번에도 root를 받고, 이번엔 gap을 받아.
현재는 gap 이 '-2' 값이야. Data는 45 이겠지.

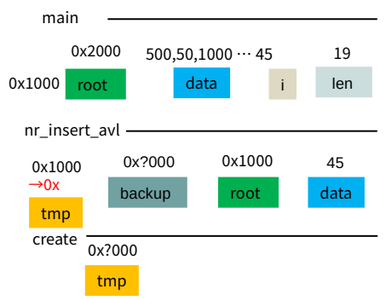
여기서 LL은 -2로 define 되어 있기 때문에 if문으로 들어가.

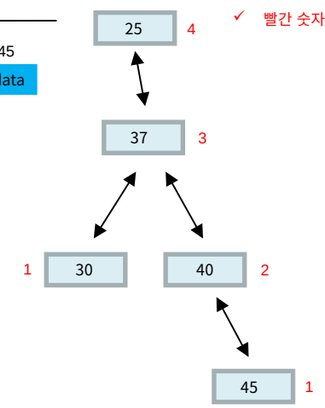
But, if문 안에 if문인 25 보다 45가 크기 때문에 return LR을 한다.

2) Root는 main의 root가 되고, term_node는 지금 현재 '50'지
그리고 method를 위해 일단 decision_rotation 함수로 넘어가.(위)

```
void rotation(avl **root, avl **term_node, int method)
{
        switch(method)
        {
                case LL:
                        printf("LL Rotation\n");
                        ll_rotation(root, term_node);
                        break;

                case LR:
                        printf("LR Rotation\n");
                        lr_rotation(root, term_node);
                        break;

                case RR:
                        printf("RR Rotation\n");
                        rr_rotation(root, term_node);
                        break;

                case RL:
                        printf("RL Rotation\n");
                        rl_rotation(root, term_node);
                        break;
        }
}
```

STACK





⑥ *Term_node의 값이 존재? -> 37이 존재.

오른쪽이 존재하나? Yes, right 변수에 '2'
왼쪽에 존재하나? Yes, left 변수에 '1'
gap = 2-1 = 1

Right > left 이므로, 37의 레벨은 '3'

그리고 backup
Term_node는 '37'의 parent인 '25'가 됨

오른쪽 존재? Yes, right 변수에 '3'
왼쪽 존재? Yes, left 변수에 '2'
Gap = 3-2 = 1

Right > left 이므로 25의 레벨은 '4'

그리고 backup
term_node는 '25의 parent인 '50'이 됨

오른쪽이 존재? Yes, right 변수가 '2'
왼쪽이 존재? Yes, left 변수가 '4'
Gap = -2

ABS(gap) > 1 가 드디어 된다.

자, 그러면 이제 rotation 함수를 살펴 보자.(우측 상단)

4-12) AVL 트리 LR 회전 이해하기

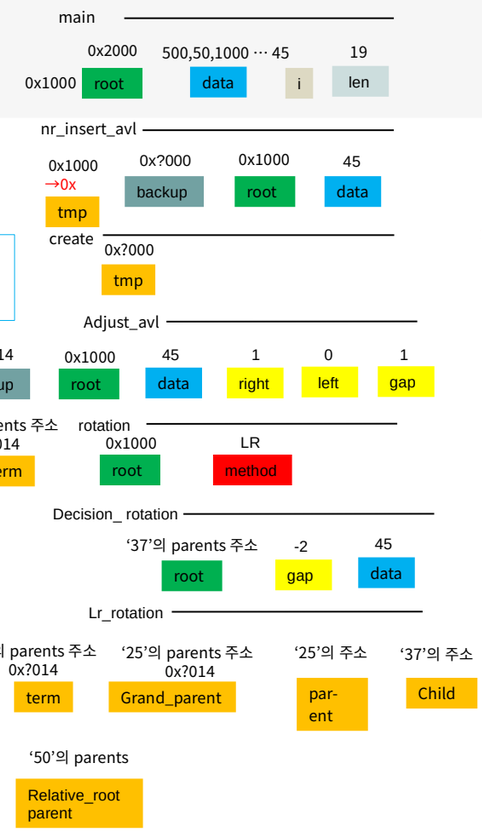
본격적으로 LR 회전 이해하기.

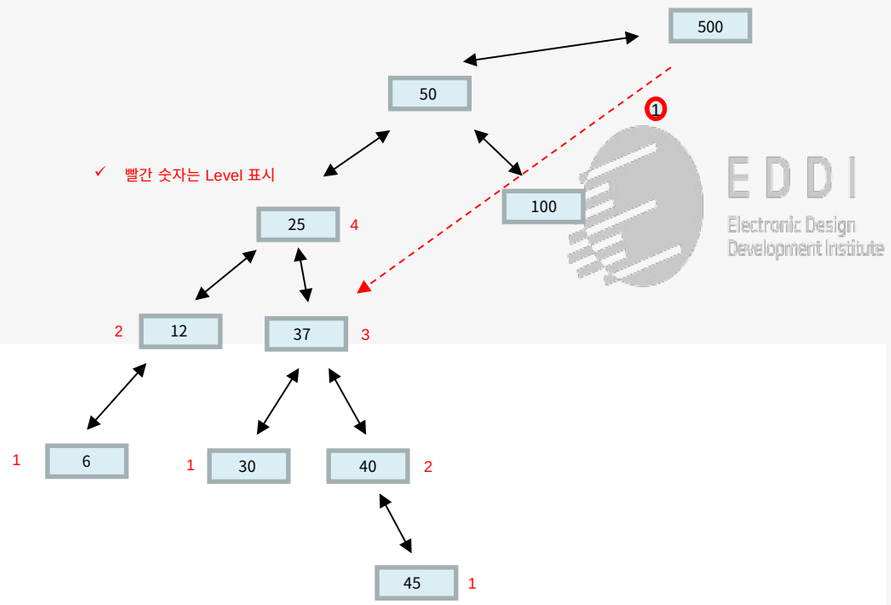
```
void lr_rotation(avl **root, avl **term_node)
{
#if 1
        avl *grand_parent = *term_node;
        avl *parent = (*term_node)->left;
        avl *child = parent->right;

        avl *relative_root_parent = (*term_node)->parent;

        // 1
        child->parent = grand_parent->parent;

        if (!relative_root_parent)
                *root = child;
    ① else if (relative_root_parent->left == grand_parent)
                relative_root_parent->left = child;
        else
                relative_root_parent->right = child;

        // 2
        grand_parent->parent = child;
        // 3
        parent->parent = child;

        // 4 & 8
        if (child->left)
                child->left->parent = parent;
        parent->right = child->left;

        // 5
        child->left = parent;

        // 6 & 9
        if (child->right)
                child->right->parent = grand_parent;
        grand_parent->left = child->right;

        // 7
        child->right = grand_parent;

        // level update
        level_update(&grand_parent);
        level_update(&parent);
        level_update(&child);
```

'37'의 parent = '50'의 parent
→ 50의 parent 37의 parent에 대입
→ 즉 37의 parent는 50의 parent를 가르킴

'500'의 left에
'37'을 가르 키도록 한다?

STACK





[질문]

앞서 backu과 trem 값이 LR을 판단하기 위한 값이
'25' 까지 감으로써 '50' '25' '37' 의 숫자가
LR 회전의 대상이 되었는데..

회전을 하는 과정이 잘 이해가 되지 않는 듯 합니다.

수업시간에 회전을 하는 과정에 대해
코드와 함께 설명 부탁 드리겠습니다.

```
LR Rotation
data = 500,     level = 5,      parent = NULL   left = 37,      right = 1000
data = 37,      level = 4,      parent = 500    left = 25,      right = 50
data = 25,      level = 3,      parent = 37     left = 12,      right = 30
data = 12,      level = 2,      parent = 25     left = 6,       right = NULL
data = 6,       level = 1,      parent = 12     left = NULL,    right = NULL
data = 30,      level = 1,      parent = 25     left = NULL,    right = NULL
data = 50,      level = 3,      parent = 37     left = 40,      right = 100
data = 40,      level = 2,      parent = 50     left = NULL,    right = 45
data = 45,      level = 1,      parent = 40     left = NULL,    right = NULL
data = 100,     level = 2,      parent = 50     left = 75,      right = 125
data = 75,      level = 1,      parent = 100    left = NULL,    right = NULL
data = 125,     level = 1,      parent = 100    left = NULL,    right = NULL
data = 1000,    level = 3,      parent = 500    left = 750,     right = 1250
data = 750,     level = 2,      parent = 1000   left = 625,     right = 875
data = 625,     level = 1,      parent = 750    left = NULL,    right = NULL
data = 875,     level = 1,      parent = 750    left = NULL,    right = NULL
data = 1250,    level = 2,      parent = 1000   left = 1125,    right = 1375
data = 1125,    level = 1,      parent = 1250   left = NULL,    right = NULL
data = 1375,    level = 1,      parent = 1250   left = NULL,    right = NULL
```