<자료구조>

1.

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


int main(void)
{
	int a[100]={0};
	int i;
	srand(time(NULL));


	for(i=1;i<=100;i++){
		a[i-1]=rand()%4096+1;
		printf("%d ", a[i-1]);
		if(i%10 == 0)
			printf("\n");
	}
	printf("\n");
	return 0;
}
```

2.

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


int find_NULL(int *arr)
{

	int i;
	int count = 0;
	printf("check 시작\n");
	for(i=0;i<100;i++){
		if(arr[i] == '\0')
			count ++;
	}
	return count;
}

int main(void)
{
	int a[100]={0};
	int i;
	srand(time(NULL));


	for(i=1;i<=100;i++){
		a[i-1]=rand()%4096+1;
		printf("%d ", a[i-1]);
		if(i%10 == 0)
			printf("\n");
	}
	printf("\n");
	printf("낭비된 공간의 수는 : %d \n",find_NULL(a));
```

```
	return 0;
}
```

3.

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


int find_NULL(int *arr)
{

	int i;
	int count = 0;
	printf("check 시작\n");
	for(i=0;i<100;i++){
		if(arr[i] == '\0')
			count ++;
	}
	return count;
}

int main(void)
{
	int a[100]={0};
	int i;
	int size = 4096;
	int num=1;
	srand(time(NULL));

	printf("사이즈를 선택하시오\n");
	printf("사이즈는 4096 의 배수입니다.\n");
	do{
		printf(" 131072 까지 가능합니다. 몇배를 할지 선택해 주세요.\n");
		scanf("%d",&num);

	}while((size*num)>=131072);


	for(i=1;i<=100;i++){
		a[i-1]=rand()%(size*num)+1;
		printf("%d ", a[i-1]);
		if(i%10 == 0)
			printf("\n");
	}
	printf("\n");
	printf("낭비된 공간의 수는 : %d \n",find_NULL(a));


	return 0;
}
```

4.

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


#define EMPTY 0
```

```
typedef struct __tree
{
	int data;
	struct __tree *link_right;
	struct __tree *link_left;

}tree;

tree *get_node()
{
	tree *tmp;
	tmp = (tree*)malloc(sizeof(tree));
	(tmp)->link_right = EMPTY;
	(tmp)->link_left = EMPTY;
	return tmp;
}

void binary(tree **root, int data)
{
	tree *tmp = *root;
	if(tmp == NULL){
			*root = get_node();
			(*root)->data = data;
			return ;
	}
	if((*root)->data > data)
	{
			 binary(&(*root)->link_left, data);
	}
	else if((*root)->data < data)
	{
			 binary(&(*root)->link_right, data);
	}

}
void print(tree *root)
{
	tree *tmp =root;
	if(root)
	{
		printf("data = %d ,", tmp->data);
		if(root->link_left){
			printf("left = %d, " ,root->link_left->data);
		}
		else
			printf("left = NUll, ");
		if(root->link_right){
				printf("right = %d\n", root->link_right->data);
		}
		else
				printf("right = NULL\n");


		print(root->link_left);
		print(root->link_right);
	}

}


int main(void)
{
```

```
	tree *root=EMPTY;
	int a[]={50, 45, 73, 32, 48, 46, 16, 37, 120, 47, 130 ,127, 124};
	int i, num=0;
	int len = sizeof(a)/sizeof(int);

	printf("len = %d \n",len);

	for(i =0;i<len;i++)
	{
		binary(&root,a[i]);
	}
	print(root);


	return 0;
}
```

5.재귀없이 이진트리

6.avl 재귀

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

#define EMPTY 0

typedef struct __avl
{
	int data;
	int level;
	struct __avl *link_left;
	struct __avl *link_right;

}avl;

typedef enum __rot
{
	RR,
	RL,
	LL,
	LR
}rot;


avl *get_node()
{
	avl *tmp;
	tmp = (avl*)malloc(sizeof(avl));
	tmp->link_left = EMPTY;
	tmp->link_right = EMPTY;
	return tmp;
}

int update_level(avl *root)
{
		int left = root->link_left ? root ->link_left->level :0;
		int right = root->link_right ? root ->link_right->level :0;

		if(left>right)
				return left+1;
		return right +1;
```

```
}

int rotation_check(avl *root)
{
	int left = root->link_left ? root -> link_left->level : 0;
	int right = root->link_right ? root -> link_right -> level : 0;

	return right - left ;
}

avl *chg_node(avl *root)
{
		avl *tmp = root;

		if(!(root->link_left)){
			free(tmp);
			return root ->link_right;
		}
		else if(!(root->link_right)){
			free(tmp);
			return root ->link_left;
		}
}
avl *find_max(avl *root, int *data)
{
	if(root->link_right){
			root->link_right = find_max(root->link_right, data);
	}
	else{
			*data = root->data;
			root= chg_node(root);
	}
	return root;
}
int kinds_of_rot(avl *root, int data)
{
	printf("data = %d\n", data);

	//for RR and RL

	if(rotation_check(root) >1)
	{
		if(root->link_right->data >data)
				return RL;
		return RR;
	}
	else if(rotation_check(root) <-1)
	{
		if(root->link_left->data < data)
				return LR;
		return LL;
	}
}


avl *rotation(avl *root , int num )
{
		avl *tmp = root;
		//RR:1
		//RL:2
		//LL:3
		//LR:4
	switch (num){
```

```
                        case RR:
                                        tmp = root ->link_right->link_left;
                                        root->link_right->link_left= root;
                                        root->link_right= tmp ;
                                        break;
                        case RL:
                                        tmp = root ->link_right ->link_left ->link_left;
                                        root ->link_right ->link_left ->link_right =root ->link_right;
                                        root ->link_right ->link_left ->link_left= root;
                                        root->link_right ->link_left = 0;
                                        root ->link_right =tmp;

                                        break;
                        case LL:
                                        tmp = root ->link_left->link_right;
                                        root->link_left->link_right= root;
                                        root->link_left= tmp ;
                                        break;
                        case LR:
                                        tmp = root ->link_left ->link_right ->link_right;
                                        root ->link_left ->link_right ->link_left =root ->link_left;
                                        root ->link_left ->link_right ->link_right= root;
                                        root->link_left ->link_right = 0;
                                        root ->link_left =tmp;
                                        break;
                        default:

                                        break;
        }

        return root;
}


void print(avl *root)
{
                avl *tmp =root;
                if(root){
                }
                printf("data = %d ,", tmp->data);
                if(root->link_left){
                                printf("left = %d, " ,root->link_left->data);
                }
                else
                                printf("left = NUll, ");
                if(root->link_right){
                                printf("right = %d\n", root->link_right->data);
                }
                else
                                printf("right = NULL\n");
                print(root->link_left);
                print(root->link_right);
}

void avl_ins(avl **root, int data)
{
        if(!(*root))
        {
                        *root = get_node();
                        (*root)->data = EMPTY;
                        return ;
        }
```

```
	if((*root)->data > data)
	{
			avl_ins(&(*root)->link_left , data);
	}
	else if((*root)->data <data)
	{
			avl_ins(&(*root)->link_right , data);
	}

	(*root)->level = update_level(*root);

	if(abs(rotation_check(*root)) >1)
	{
		printf("Rotation \n");
		*root = rotation(*root,kinds_of_rot(*root,data));
	}
}
avl *debinary(avl *root,int data)
{
	int num;
	avl *tmp;
	if(root == NULL)
	{
			printf("Not Found\n");
			return NULL;
	}
	else if(root->data >data)
			root->link_left = debinary(root->link_left,data);
	else if(root->data <data)
			root ->link_right = debinary(root->link_right, data);
	else if(root->link_left && root->link_right)
	{
			root->link_left =find_max(root->link_left, &num);
			root->data = num;
	}
	else
			root=chg_node(root);
	return root;
}



int main(void)
{
	avl *root=EMPTY;
	int a[]={50,45,73,32,48,46,16,37,120,127,124};
	int i, num=0;
	int len = sizeof(a)/sizeof(int);

	for(i=0 ; i <len;i++)
			avl_ins(&root, a[i]);
	print(root);
//	root=debinary(root,50);
	print(root);

	return 0;
}
```

7.Red Black 트리와 AVL 트리를 비교해보도록 한다.

AVL 트리는 2 진트리에서 level; 개념이 생겨 더 완벽한 2 진트리를 구성하고 있지만 삽입과 삭제시에 더 엄격한 균형을 요구한다. 그래서 더 많은 회전을 해야 할 때가 있다. 반면 RedBlack 트리는 제일 작은쪽 노드보다 긴쪽에 노드의 길이가 2 배라는 차이를 가지기에 속도면에서 더 효율적이다.

8.난수를 활용하여 Queue 를 구현한다.(20 번)

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

#define EMPTY 0

typedef struct __queue
{
	int data;
	struct __queue *link;

}queue;


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

void print(queue *head){

	queue *tmp=head;


	while(tmp){
		printf("값 : %d\n", tmp->data);
		tmp = tmp ->link;
	}

}

void enqueue(queue **head, int data)
{
	if(*head == NULL)
	{
		*head = get_node();
		(*head) -> data =data;

		//		printf("%d\n",data);
		return ;
	}

	enqueue(&((*head)->link),data);
}



int main(void)
{
	//int data=10;
```

```
	queue *head=EMPTY;
	int i;
	int arr[16]={0};
	srand(time(NULL));
	for(i=0;i<16;i++)
	{
		arr[i]=rand()%16;
		enqueue(&head, arr[i]);
	}

	print(head);
	return 0;
}
```

9.재귀호출을 사용하여 queue 를 구현하고 10, 20 을 집어넣는다.

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

#define EMPTY 0
typedef struct __queue
{
	int data;
	struct __queue *link;

}queue;

queue *get_node()
{
	queue *tmp;
	tmp = (queue *)malloc(sizeof(queue));
	tmp -> link = EMPTY;

	return tmp;
}

void enqueue(queue **head ,int data)
{
	if(*head == NULL){
		*head = get_node();
		(*head) -> data = data;
		return ;
	}
	enqueue((&(*head)->link),data);

}

void print(queue *head)
{
	queue *tmp = head;
	while(tmp)
	{
		printf("%d \n", tmp->data);
		tmp = tmp->link;
	}

}

queue *dequeue(queue *head, int data)
{
	queue *tmp = head;
	if(tmp == NULL)
```

```
                printf("값이 없습니다.\n");
        if(head->data != data)
                head ->link = dequeue(head ->link, data);
        else
        {
                printf("Now you delete %d \n", data);
                free(tmp);
                return head -> link;
        }
        return head;
}


int main(void)
{
        queue *head =EMPTY;
        enqueue(&head,10);
        enqueue(&head,20);

        print(head);

        return 0;

}
```

10.난수를 활용해서 Stack 을 구성한다.

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

#define EMPTY 0

struct node
{
        int data;
        struct node *link;

};

typedef struct node Stack;

Stack *get_nod()
{
        Stack *tmp;
        tmp = (Stack *)malloc(sizeof(Stack));
        tmp -> link = EMPTY;
        return tmp;
}

void push(Stack **top, int data)
{
        Stack *tmp;
        tmp = *top;
        *top = get_nod();
        (*top) -> data = data;
        (*top) -> link = tmp;

}

int pop(Stack **top)
```

```
{
	Stack *tmp;
	int num;
	tmp = *top;
	if(tmp == NULL){
				printf("값이 없다\n");
				return 0;
	}
	num = tmp -> data;
	*top = (*top)->link;
	free(tmp);
	return num;
}

int main(void)
{
	Stack *top;
	top = EMPTY;
	int arr[20]={0};
	int i = 0;
	srand(time(NULL));
	for(i=0;i<20;i++)
	{
		arr[i]=rand()%100+1;
		push(&top, arr[i]);
	}
	for(i=0;i<21;i++)
		printf("%d\n",pop(&top));
	return 0;
}
```

11.2.1 에서 만든 내용중 홀수만 빼내서 AVL 트리를 구성하도록 한다.

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

typedef enum __rot
{
	RR,
	RL,
	LL,
	LR
} rot;

typedef struct __avl_tree
{
	int lev;
	int data;
	struct __avl_tree *left;
	struct __avl_tree *right;
} avl;

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

bool is_dup(int *arr, int cur_idx)
{
	int i, tmp = arr[cur_idx];

	for(i = 0; i < cur_idx; i++)
		if(tmp == arr[i])
			return true;
```

```
	return false;
}

void init_rand_arr(int *arr, int size)
{
	int i;

	for(i = 0; i < size; i++)
	{
redo:
		//arr[i] = rand() % 15 + 1;
		arr[i] = rand() % 100 + 1;

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

void print_arr(int *arr, int size)
{
	int i;

	for(i = 0; i < size; i++)
		printf("arr[%d] = %d\n", i, arr[i]);
}

avl *get_avl_node(void)
{
	avl *tmp;
	tmp = (avl *)malloc(sizeof(avl));
	tmp->lev = 1;
	tmp->left = NULL;
	tmp->right = NULL;
	return tmp;
}

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

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

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

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

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

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

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

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

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

	push(&top, *tmp);

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

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

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

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

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

int update_level(avl *root)
{
	int left = root->left ? root->left->lev : 0;
	int right = root->right ? root->right->lev : 0;

	if(left > right)
		return left + 1;

	return right + 1;
}

int rotation_check(avl *root)
{
	int left = root->left ? root->left->lev : 0;
	int right = root->right ? root->right->lev : 0;

	return right - left;
}

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

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

```
{
	//parent->right = child->left ? child->left : child->right;
	parent->right = child->left;
	child->left = parent;
	parent->lev = update_level(parent);
	child->lev = update_level(child);
	return child;
}

avl *ll_rot(avl *parent, avl *child)
{
	//parent->left = child->right ? child->right : child->left;
	parent->left = child->right;
	child->right = parent;
	parent->lev = update_level(parent);
	child->lev = update_level(child);
	return child;
}

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

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

avl *rotation(avl *root, int ret)
{
	switch(ret)
	{
		case RL:
			printf("RL Rotation\n");
			return rl_rot(root, root->right);
		case RR:
			printf("RR Rotation\n");
			return rr_rot(root, root->right);
		case LR:
			printf("LR Rotation\n");
			return lr_rot(root, root->left);
		case LL:
			printf("LL Rotation\n");
			return ll_rot(root, root->left);
	}
}

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

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

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

	while(stack_is_not_empty(top))
	{
		printf("Extract Stack: %d, data = %d\n", --cnt, data);
		avl **t = (avl **)pop(&top);
		(*t)->lev = update_level(*t);
		if(abs(rotation_check(*t)) > 1)
```

```
                {
                        printf("Insert Rotation\n");
                        // Need to change here with pointer of pointer
                        //*tmp = rotation(*tmp, kinds_of_rot(*tmp, data));
                        //*root = rotation(*tmp, kinds_of_rot(*tmp, data));
                        /* It's just same as else. */
#if 0
                        if((*root) == (*t))
                                *root = rotation(*t, kinds_of_rot(*t, data));
                        else
                                *t = rotation(*t, kinds_of_rot(*t, data));
#endif
                        *t = rotation(*t, kinds_of_rot(*t, data));
                }
        }

#if 0
        //update_level(root);
        (*root)->lev = update_level(*root);

        if(abs(rotation_check(*root)) > 1)
        {
                printf("Insert Rotation!\n");
                *root = rotation(*root, kinds_of_rot(*root, data), data);
        }
#endif
}

avl *chg_node(avl *root)
{
        avl *tmp = root;

        if(!root->right)
                root = root->left;
        else if(!root->left)
                root = root->right;

        free(tmp);

        return root;
}

#if 0
avl *find_max(avl *root, int *data)
{
        if(root->right)
                root->right = find_max(root->right, data);
        else
        {
                *data = root->data;
                root = chg_node(root);
        }

        return root;
}
#endif

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

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

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

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

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

				(*tmp) = chg_node(*tmp);

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

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

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

			return;
		}

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

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

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

iint main(void)
{
		int i;
		avl *root = NULL;
		avl *test = NULL;
		int arr[20] = {0};
		int size = sizeof(arr) / sizeof(int) - 1;

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

		srand(time(NULL));

		init_rand_arr(arr, size);
```

```
	print_arr(arr, size);

	for(i = 0; i < 20; i++){
		arr[i]=rand()%100+1;
		if(arr[i]%2 != 0)
			avl_ins(&root, arr[i]);
	}

	print_tree(&root);

	return 0;
}
```

```
arr[6] = 94
arr[7] = 67
arr[8] = 10
arr[9] = 100
arr[10] = 86
arr[11] = 20
arr[12] = 36
arr[13] = 13
arr[14] = 83
arr[15] = 2
arr[16] = 57
arr[17] = 88
arr[18] = 81
Save Stack: 1, data = 51
Extract Stack: 0, data = 51
Save Stack: 1, data = 29
Save Stack: 2, data = 29
Extract Stack: 1, data = 29
Extract Stack: 0, data = 29
Insert Rotation
RL Rotation
Save Stack: 1, data = 57
Save Stack: 2, data = 57
Extract Stack: 1, data = 57
Extract Stack: 0, data = 57
Save Stack: 1, data = 47
Save Stack: 2, data = 47
Extract Stack: 1, data = 47
Extract Stack: 0, data = 47
Save Stack: 1, data = 55
Save Stack: 2, data = 55
Save Stack: 3, data = 55
Extract Stack: 2, data = 55
Extract Stack: 1, data = 55
Extract Stack: 0, data = 55
Insert Rotation
RR Rotation
Save Stack: 1, data = 77
Save Stack: 2, data = 77
Extract Stack: 1, data = 77
Extract Stack: 0, data = 77
Save Stack: 1, data = 59
Save Stack: 2, data = 59
Save Stack: 3, data = 59
Extract Stack: 2, data = 59
Extract Stack: 1, data = 59
Extract Stack: 0, data = 59
data = 51, lev = 4, left = 29, right = 57
data = 29, lev = 2, left = 27, right = 47
data = 27, lev = 1, left = NULL, right = NULL
data = 47, lev = 1, left = NULL, right = NULL
data = 57, lev = 3, left = 55, right = 77
data = 55, lev = 1, left = NULL, right = NULL
data = 77, lev = 2, left = 59, right = NULL
data = 59, lev = 1, left = NULL, right = NULL
jhb@onestar:~/My/Homework/hanbyuljung/today_test$
```

12.2.1 에서 짝수만 빼내서 RB 트리를 구성하도록 한다.

13.최적화 프로세스를 기술하도록 한다.

소스코드(.c) -- 전처리 --> 전처리후 소스(.i) -- C 컴파일 --> 어셈블리소스 (.s) -- 어셈블리컴파일 --> 오브젝트 파일(.o) -- 링크 --> 실행파일 (a.out)

14.이제 Queue 에서 데이터로서 숫자 값이 아닌 문자열을 받아보도록 하자

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

#define EMPTY 0

typedef struct __queue
{
	int data;
	struct __queue *link;

}queue;

queue *get_node(){

	queue *tmp;
	tmp = (queue *)malloc(sizeof(queue));
	tmp -> link = EMPTY;

	return tmp;
}

void enqueue(queue **head, int data){

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

		return ;
	}

	enqueue(&((*head)->link),data);
}
void print_queue(queue *head)
{
	queue *tmp;
	tmp = head;
	while(tmp)
	{
		printf("%c\n", tmp -> data);
		tmp = tmp ->link;
	}

}
void queue_delete(queue *head,int data)
{
	queue *tmp;
	tmp = head;
	while(tmp)
```

```
	{
		if((tmp -> data) == data){
		//	printf("같습니다.%d\n",data);
			tmp = tmp ->link;
		}
		else
		{
			printf("%d\n", tmp->data);
			tmp = tmp ->link;
		}
	}
}

void queue_delete2(queue *head, int data)
{
	queue *tmp;
	tmp = head;
	if((tmp->data) == data)
	{
		head -> link = tmp -> link;
		printf("같습니다.\n");
		free(tmp);
	}
	else if((tmp->data) != data)
	{
		head->link = tmp -> link;
		printf("res = %d\n", tmp -> data);
	}
	else
			return ;

	queue_delete2( (tmp->link) , data);
}

queue *queue_delete3(queue *head, int data)
{
	queue *tmp = head;

	if(tmp ==NULL)
			printf("There are no data that you delete\n");
	if(head ->data != data)
			head ->link = queue_delete3(head->link, data);
	else
	{
			// queue *res = head ->link;
			printf("Now you delete %d\n",data);
			free(tmp);
			return head->link;
	}
	return head;
}

int main(void){

	queue *heap = EMPTY;
//	srand(time(NULL));
	char arr[]="today";

	int i;
	for(i=0;i<strlen(arr);i++)
		enqueue(&heap, arr[i]);
```

```
	print_queue(heap);

	return 0;
}
```

```
jhb@onestar:~/My/Homework/hanbyuljung/class_9_me$ gcc -o quedel_14 quedel.c
jhb@onestar:~/My/Homework/hanbyuljung/class_9_me$ ./quedel_14
t
o
d
a
y
jhb@onestar:~/My/Homework/hanbyuljung/class_9_me$
```

15. AVL 트리에 데이터로서 숫자가 아닌 문자열을 입력하도록 프로그램하시오.

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

#define EMPTY 0;

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("%c 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[%c] = %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 = %c, lev = %d, ", root->data, root->lev);

                if(root->left)
                        printf("left = %c, ", root->left->data);
                else
                        printf("left = NULL, ");

                if(root->right)
                        printf("right = %c\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 = %c\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)
{
	avl *root=EMPTY;
//	int a[]={50,45,73,32,48,46,16,37,120,127,124};
	int i, num=0;
//	int len = sizeof(a)/sizeof(int);

	char arr[]="today";

	for(i=0;i<strlen(arr);i++)
		avl_ins(&root, arr[i]);

	print_tree(root);

	return 0;
}
```

```
jhb@onestar:~/My/Homework/hanbyuljung/today_test$ ./a.out
Insert Rotation!
data = d
LL Rotation
data = o, lev = 3, left = d, right = t
data = d, lev = 2, left = a, right = NULL
data = a, lev = 1, left = NULL, right = NULL
data = t, lev = 2, left = NULL, right = y
data = y, lev = 1, left = NULL, right = NULL
jhb@onestar:~/My/Homework/hanbyuljung/today_test$
```

16.Binary Tree 에 문자열을 입력한다.

17.성적 관리 프로그램을 만들어보자.

여태까지 배운 학습 내용들을 활용하여 성적 관리 프로그램을 설계하고 구현해보자.

1. 통계 기능(총 합산, 평균, 표준 편차 계산)

2. 성적순 정렬 기능

3. 성적 입력 기능

4. 학생 정보 삭제 기능