1.선행처리기
2.#include<stdio.h>

```
int n1=0,n2=0,n3=0,n4=0,n5=0;
int n6=0,n7=0,n8=0,n9=0,n10=0;
int n11=0,n12=0,n13=0,n14=0,n15=0,n16=0,n17=0,n18=0,n234=0;
int
n345=0,n26023=0,n346=0,n457=0,n1224=0,n34=0,n646=0,n732=0,n4467=0,n45=0,n623=0,n356=
0,n123=0,n3245=0,n6567=0,n567=0,n6789=0,n2334=0,n4576=0,n678=0,n789=0,n1000=0;
int
n2400=0,n978=0,n456=0,n234756=0,n5000=0,n500=0,n4564=0,n3243=0,n876=0,n645=0,n534=0,
n423=0,n312=0,n756=0,n235=0,n75678=0;
int maxn = 0;
void arrcmp(int* arr,int arrlen)
{
	int j = arrlen;
	for(j = 0; j < arrlen; j++){
		//printf("%d\n",arr[j]);
		if(arr[j] == 1)
			n1++;
		else if(arr[j] == 2)
			n2++;
		else if(arr[j] == 3)
			n3++;
		else if(arr[j] == 4)
			n4++;
		else if(arr[j] == 5)
			n5++;
		else if(arr[j] == 6)
			n6++;
		else if(arr[j] == 7)
			n7++;
		else if(arr[j] == 8)
			n8++;
		else if(arr[j] == 9)
			n9++;
		else if(arr[j] == 10)
			n10++;
		else if(arr[j] == 11)
			n11++;
		else if(arr[j] == 12)
			n12++;
		else if(arr[j] == 13)
			n13++;
		else if(arr[j] == 14)
			n14++;
		else if(arr[j] == 15)
			n15++;
		else if(arr[j] == 16)
			n16++;
		else if(arr[j] == 17)
			n17++;
		else if(arr[j] == 18)
```

```
        n18++;
else if(arr[j] == 234)
        n234++;
else if(arr[j] == 345)
        n345++;
else if(arr[j] == 26023)
        n26023++;
else if(arr[j] == 346)
        n346++;
else if(arr[j] == 457)
        n457++;
else if(arr[j] == 1224)
        n1224++;
else if(arr[j] == 34)
        n34++;
else if(arr[j] == 646)
        n646++;
else if(arr[j] == 732)
        n732++;
else if(arr[j] == 4467)
        n4467++;
else if(arr[j] == 45)
        n45++;
else if(arr[j] == 623)
        n623++;
else if(arr[j] == 356)
        n356++;
else if(arr[j] == 123)
        n123++;
else if(arr[j] == 3245)
        n3245++;
else if(arr[j] == 6567)
        n6567++;
else if(arr[j] == 567)
        n567++;
else if(arr[j] == 6789)
        n6789++;
else if(arr[j] == 2334)
        n2334++;
else if(arr[j] == 4576)
        n4576++;
else if(arr[j] == 678)
        n678++;
else if(arr[j] == 789)
        n789++;
else if(arr[j] == 645)
        n645++;
else if(arr[j] == 534)
        n534++;
else if(arr[j] == 423)
        n423++;
else if(arr[j] == 312)
```

```
                n312++;
            else if(arr[j] == 756)
                n756++;
            else if(arr[j] == 235)
                n235++;
            else if(arr[j] == 75678)
                n75678++;
            else if(arr[j] == 1000)
                n1000++;
            else if(arr[j] == 2400)
                n2400++;
            else if(arr[j] == 978)
                n978++;
            else if(arr[j] == 456)
                n456++;
            else if(arr[j] == 234756)
                n234756++;
            else if(arr[j] == 5000)
                n5000++;
            else if(arr[j] == 500)
                n500++;
            else if(arr[j] == 4564)
                n4564++;
            else if(arr[j] == 3243)
                n3243++;
            else if(arr[j] == 876)
                n876++;
        }
}
int arrmax(int* arrn, int arrlen){
        int j = arrlen;
        int i =0;
        int max= 0;
        int maxarr[3] = {0};
        int maxn = 0;
        for(i = 0; i < 3; i++)
        {
            for(j = 0; j < arrlen; j++){

                        if(maxarr[i] < arrn[j]){
                        maxarr[i] = arrn[j];
                        maxn = j;
                        }

            }
            printf("빈도수 %d 번째로 많은수 = %d \n",i,maxarr[i]);
        }

}
int main(void)
{
        int arr[] = {2400, 2400, 2400, 2400, 2400, 2400, 2400,
```

```
  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,
  2400, 2400,  2400,  2400, 2400,  2400, 1,  2,  3,  4,
  5,  1,  2,  3,  4,  5,  2400, 2400, 2400, 2400, 2400, 5000,
  1,  2,  3,  4,  5,  5000, 5000, 500,  500,  500,  500,  500,
  1,  2,  3,  4,  5,  500,  500,  500,  500,  500,  500,  500,
  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,
  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,
  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,
  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,
  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,
  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,  6,  7,  8,  9,  10,
  11,  12,  13,  14,  15,  16,  17,  18,  234,  345,  26023, 346,  345,  234,
  457,  3,  1224, 34,  646,  732,  5,  4467,  45,  623,  4,  356,  45,  6,  123,
  3245, 6567,  234,  567,  6789,  123,  2334, 345,  4576, 678,  789,  1000,
  2400, 2400,  2400, 2400, 2400, 2400, 1,  2,  3,  4,  5,  1,  2,  3,  4,  5,
  2400, 2400, 2400,  2400,  978,  456,  234756, 5000,  5000, 5000,  2400, 500, 5000, 2400, 500,
  500,  500,  500,  500,  500,  1,  2,  3,  4,  5,  500,  500,  500,  500,  500,
  500,  500, 500, 500, 500, 1,  2,  3,  4,  5,  500, 500, 500, 5000, 2400, 5000,
  5000, 5000, 5000, 5000, 5000, 5000,  5000,  1,  2,  3,  4,  5,  5000, 5000,
  5000, 5000, 2400, 5000, 500, 2400, 5000, 5000, 5000, 5000, 5000,  5000,  5000, 5000,
  1,  2,  3,  4,  5,  1,  2,  3,  4,  5,  5000, 5000, 5000, 5000, 5000,
  1,  2,  3,  4,  5,  5000, 5000,  5000,  5000, 5000,  234,  4564, 3243, 876,
  645,  534,  423,  312,  756,  235,  75678, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000,
  500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500, 2400,
  5000, 500, 2400, 5000, 500, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8,
  9, 6,  500, 2400, 5000,  500, 2400, 5000,  500, 2400, 5000,  500, 2400, 5000,  500, 2400, 5000,
  500, 2400, 5000};
        int arrn[] =
{n5000,n2400,n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,n13,n14,n15,n16,n17,n18,n234,n345,n2602
3,n346,n457,n1224,n34,n646,n732,n4467,n45,n623,n356,n123,n3245,n6567,n567,n6789,n2334,n4
576,n678,n789,n1000,n2400,n978,n456,n234756,n5000,n500};
        //printf("%d\n",arrn[]);
        arrcmp(arr,sizeof(arr)/sizeof(int));
        arrmax(arrn,sizeof(arrn)/sizeof(int));
        return 0;
}
```

3.

4.cpu 는 비교연산과 jump 연산이 치명적인데, while 문 사용중 에러발생시
멈추지않고 while 문 이 끝나야 그제서야 종료되는 경우도 있고, 그렇게 되면 불필요한 cpu 동작이 이루어져서 좋지 않다. 이를 막기 위해 에러발생시 break 을 사용하는데 한 개의 while 문이면 상관 없지만, 다중으로 이루어진 while 문은 5 개의 while 문으로 이루어져 있다면 break 을 5 번써서 점프를 하고 비교연산을 해야한다. 하지만 goto 를 사용하면 1 번의 점프로 cpu 의 운동낭비를 막을 수 있다.

5.포인터는 운영체제에 의존적인데, 운영체제가 32 비트의 처리단위를 가지고 있다면 이를 바이트로 환산한 4 바이트의 크기를 포인터의 크기로 갖는다. 마찬가지로 64 비트 처리단위를 가지고있는 운영체제 안에서의 포인터의 크기는 이를 환산함 8 바이트의 크기를 갖는 포인터를 갖는다.

6. ㅐ

7. ㅐ

8.void(*signal(int signnum, void(*handler)(int)))(int)는

void(*)(int)signal(int signnum, void(*handler)(int))로 바뀔 수 있는데
이름이 signal 이고 인자를 int 형 변수 signnum 과 반환형이 void 이고 전달인자로 int 를 갖는 함수포인터를 갖는 출력이 인자로 int 형을 갖고 출력을 void 로 갖는 함수 포인터를 갖는 함수이다.
9. ㅐ
10.파이프라인이 깨지는 경우는 점프연산을 할 때 인데, 이를 사용하면 불필요한 cpu 의 활동을 막을 수 있는 이점이 있다.
11. ㅐ
12. ㅐ
13. ㅐ
14. ㅐ
15.

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

typedef struct __tree
{
	int data;
	struct __tree *left;
	struct __tree *right;
} tree;

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

void tree_ins(tree **root, int data)
{
	if(*root == NULL)
	{
		*root = get_node();
		(*root)->data = data;
		return;
	}
	else if((*root)->data > data)
		tree_ins(&(*root)->left, data);
	else if((*root)->data < data)
		tree_ins(&(*root)->right, data);
}
```

```
void print_tree(tree *root)
{
	if(root)
	{
		printf("data = %d, ", root->data);

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

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

		print_tree(root->left);
		print_tree(root->right);
	}
}



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

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

	free(tmp);

	return root;
}

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

	return root;
}

tree *delete_tree(tree *root, int data)
{
```

```
	int num;
	tree *tmp;
	if(root == NULL)
	{
		printf("Not Found\n");
		return NULL;
	}
	else if(root->data > data)
		root->left = delete_tree(root->left, data);
	else if(root->data < data)
		root->right = delete_tree(root->right, data);
	else if(root->left && root->right)
	{
		root->left = find_max(root->left, &num);
		root->data = num;
	}
	else
		root = chg_node(root);
	return root;
}



int main(void)
{
	int i;
	int data[14] = {50, 45, 73, 32, 48, 46, 16,
		37, 120, 47, 130, 127, 124};

	tree *root = NULL;

	for(i = 0; data[i]; i++)
		tree_ins(&root, data[i]);

	print_tree(root);

	delete_tree(root, 50);
	printf("After Delete\n");

	print_tree(root);

	return 0;
}
16.#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct __tree
{
	int data;
	struct __tree *left;
	struct __tree *right;
```

```
} tree;

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

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

tree *get_tree_node(void)
{
	tree *tmp;
	tmp = (tree *)malloc(sizeof(tree));
	tmp->left = NULL;
	tmp->right = 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;
}

void non_recur_tree_ins(tree **root, int data)
{
	tree **tmp = root;

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

	*tmp = get_tree_node();
	(*tmp)->data = data;
}

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

void 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);

                //tmp = &(*tmp)->left;

                //*tmp = (tree *)pop(&top);
        }
}

#if 0
void print_tree(tree *root)
{
        if(root)
        {
                printf("data = %d, ", root->data);

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

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

                print_tree(root->left);
                print_tree(root->right);
        }
}
#endif

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

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

        free(tmp);

        return root;
}

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

        while(*tmp)
        {
```

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

void non_recur_delete_tree(tree **root, int data)
{
        tree **tmp = root;
        int num;

        while(*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;
                        return;
                }
                else
                {
                        (*tmp) = chg_node(*tmp);
                        return;
                }
        }

        printf("Not Found\n");
}

int main(void)
{
        int i;
        int data[14] = {50, 45, 73, 32, 48, 46, 16,
                37, 120, 47, 130, 127, 124};

        tree *root = NULL;

        for(i = 0; data[i]; i++)
                non_recur_tree_ins(&root, data[i]);

        print_tree(&root);

        non_recur_delete_tree(&root, 50);
```

```
	printf("After Delete\n");

	print_tree(&root);

	return 0;
}
17.#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

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

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

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

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

	return false;
}

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

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

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

```
                        goto redo;
                }
        }
}

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

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

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

void print_tree(avl *root)
{
        if(root)
        {
                printf("data = %d, lev = %d, ", root->data, root->lev);

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

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

                print_tree(root->left);
                print_tree(root->right);
        }
}

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

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

```
	return right + 1;
}

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

	return right - left;
}

int kinds_of_rot(avl *root, int data)
{
	printf("data = %d\n", data);

	// for RR and RL
	if(rotation_check(root) > 1)
	{
		if(root->right->data > data)
			return RL;

		return RR;
	}
	// for LL and LR
	else if(rotation_check(root) < -1)
	{
		if(root->left->data < data)
			return LR;

		return LL;
	}
}

avl *rr_rot(avl *parent, avl *child)
{
	//parent->right = child->left ? child->left : 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;
}
```

```
#if 0
avl *rl_rot(avl *parent, avl *child, int data)
{
}
#endif

#if 0
avl *lr_rot(avl *parent, avl *child, int data)
{
	avl *tmp;

	if(child->right->data > data)
	{
		tmp = child->right->left;
		child->right->right = parent;
		child->right->left = child;
		child->right = tmp;
		parent->left = NULL;
	}
	else
	{
		tmp = child->right->right;
		child->right->right = parent;
		child->right->left = child;
		parent->left = tmp;
		tmp = parent->left;
		child->right = NULL;
	}

	return tmp;
}
#endif

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

#if 1
avl *lr_rot(avl *parent, avl *child, int data)
{
	child = rr_rot(child, child->right);
	//child = rr_rot(child, child->left);
	return ll_rot(parent, child);
}
#endif

//void rotation(avl *root, int ret)
avl *rotation(avl *root, int ret, int data)
```

```
{
	switch(ret)
	{
		case RL:
			printf("RL Rotation\n");
			return rl_rot(root, root->right, data);
		case RR:
			printf("RR Rotation\n");
			return rr_rot(root, root->right);
		case LR:
			printf("LR Rotation\n");
			return lr_rot(root, root->left, data);
		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), 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), data);
                //rotation(*root, kinds_of_rot(*root, data));
        }
}

int main(void)
{
        int i;
```

```
	avl *root = NULL;
	avl *test = NULL;
	int arr[16] = {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);

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

	print_tree(root);

	printf("\nAfter Delete\n");
	avl_del(&root, arr[3]);
	avl_del(&root, arr[6]);
	avl_del(&root, arr[9]);

	print_tree(root);
#endif

	printf("\nDebug AVL\n");

	for(i = 0; i < 6; i++)
		avl_ins(&test, data[i]);

	print_tree(test);

	return 0;
}
18.#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 *mid_pop(stack **top)
{
	stack *tmp = (*top)->link;
	void *data = NULL;

	if(*top == NULL)
	{
		printf("Don't do like it!\n");
		return NULL;
	}

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

	return data;
}

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, stack **top)
{
        avl *tmp = root;

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

        free(tmp);
```

```
	/* for automatic stack unwinding */
	if(root)
	{
		printf("chg_node: mid pop\n");
		mid_pop(top);
	}
	else
	{
		printf("chg_node: pop\n");
		pop(top);
	}

	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, NULL);
			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;

			printf("Do one or nothing node\n");
			(*tmp) = chg_node(*tmp, &top);

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

int main(void)
{
	int i;
	avl *root = NULL;
	avl *test = NULL;
	int arr[16] = {0};
	int size = sizeof(arr) / sizeof(int) - 1;

	//int data[] = {100, 50, 200, 25, 75, 80};
	int data[] = {100, 50, 200, 25, 75, 70};
	int d2[] = {57, 32, 15, 7, 5, 18, 28, 34, 45, 79, 73, 93, 89, 97};
	int d3[] = {50, 100, 25, 10};

	srand(time(NULL));

	init_rand_arr(arr, size);
	print_arr(arr, size);

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

	print_tree(&root);
#endif

#if 0
	printf("\nAfter Delete\n");
	avl_del(&root, arr[3]);
	avl_del(&root, arr[6]);
	avl_del(&root, arr[9]);

	print_tree(&root);
#endif

#if 0
```

```
	printf("\nDebug AVL\n");

	for(i = 0; i < 6; i++)
	{
		avl_ins(&test, data[i]);
		print_tree(&test);
	}

	printf("\nFinal Result\n");
	print_tree(&test);
#endif

	for(i = 0; i < 4; i++)
		avl_ins(&test, d3[i]);

	print_tree(&test);

	avl_del(&test, 100);

	print_tree(&test);

	return 0;
}
```

19.redblack 트리는 avl 트리에 비해 회전수가 적어질 수 있도록 하였기 때문에 데이터 입력과 삭제에 대해서 avl 트리보다 성능이 우수하다. 하지만 검색속도는 avl 트리가 빠르다.

20. ㅐ

21. ㅐ

22. ㅐ

23. ㅐ

24.intel Architecture 와 ARM Architecture 의 성능차이는 비교하기 힘들다고 하지만 지원하는 운영체제가 다르다. intel 같은 경우에는 Microsoft 가 버티고 있는 반면에 ARM 은 Linux 가 지원한다. 또한 우리가 사용하는 스마트폰도 intel 보다는 ARM 을 사용한다. 데스크탑은 intel 을 사용하는경우가 많다

25.

26.

27.

28.int *p[3]은 int 형 포인터 배열 인데 3 개가 들어갈 수 있는 공간을 선언한 것이고,
int(*p)[3]은 int 형 배열의 3 개까지 가리킬 수 있는 포인터 이다.

29.

```
#include<stdio.h>
#include<time.h>
void rand131072(int* arr){
	int i;
	for(i = 0; i< 5; i++)
	{
		arr[i] = ((rand()%10)*134217727);
		arr[i] = arr[i]&(~(134217727));
		printf("%d\n", arr[i]);
	}
}
int main(void){

	srand(time(NULL));
```

```
	int arr[100] = {0};
	rand131072(arr);
	return 0;
}
```

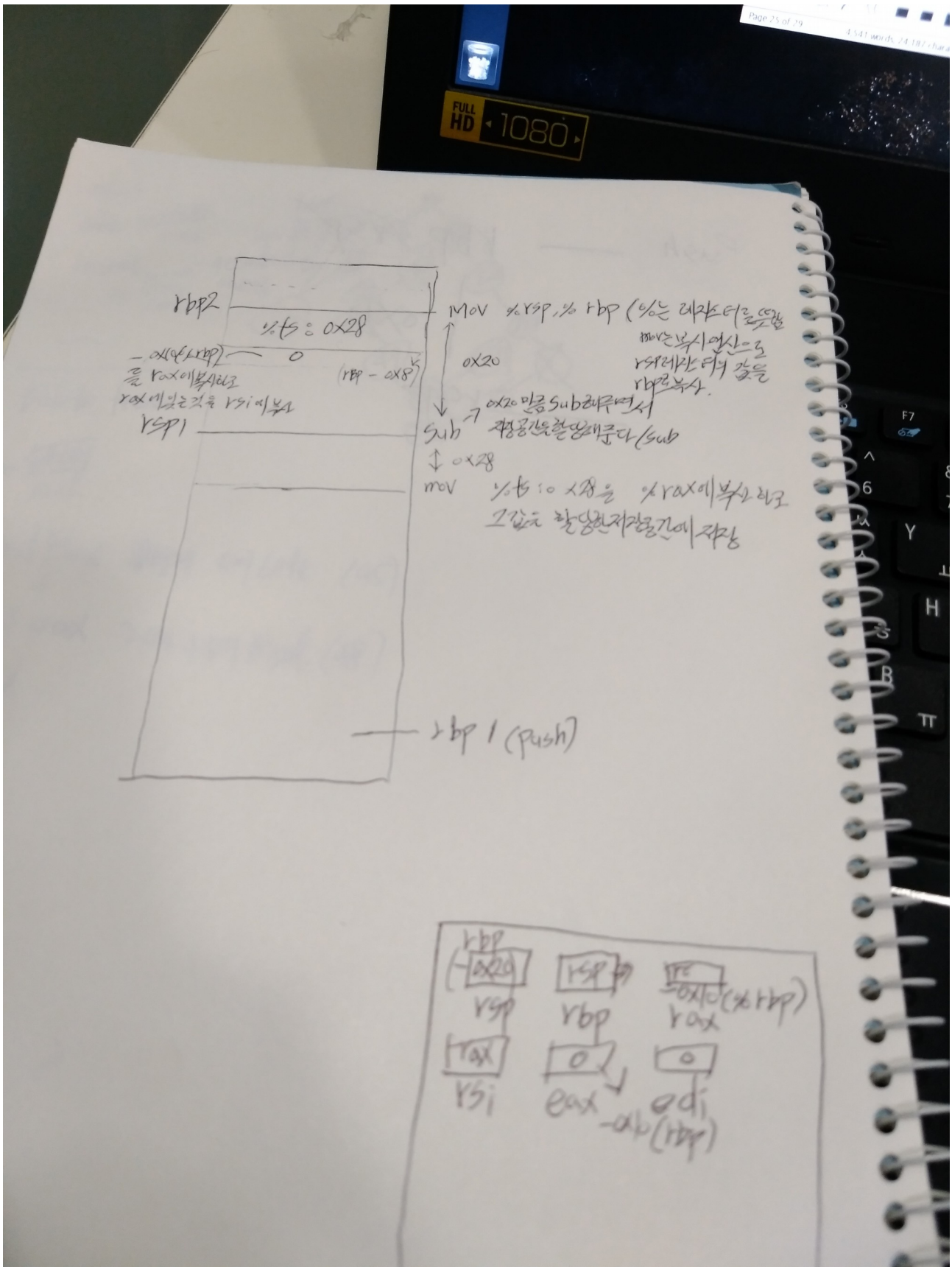
30.

31.변수는 정보를 저장할 수 있는 공간이다.

32.포인터는 변수의 주소를 저장할 수 있는 공간이다.

33.함수 포인터는 함수의 주소를 저장할 수 있는 공간이다.

34




```
=> 0x0000000000400898 <+0>:    push   %rbp
0x0000000000400899 <+1>:       mov    %rsp,%rbp
0x000000000040089c <+4>:       sub    $0x20,%rsp
0x00000000004008a0 <+8>:       mov    %fs:0x28,%rax
0x00000000004008a9 <+17>:      mov    %rax,-0x8(%rbp)
0x00000000004008ad <+21>:      xor    %eax,%eax
```

```
   0x00000000004008af <+23>:    movq   $0x0,-0x10(%rbp)
   0x00000000004008b7 <+31>:    mov    -0x10(%rbp),%rax
   0x00000000004008bb <+35>:    mov    %rax,%rsi
   0x00000000004008be <+38>:    mov    $0x400a53,%edi
   0x00000000004008c3 <+43>:    mov    $0x0,%eax
   0x00000000004008c8 <+48>:    callq  0x4005a0 <printf@plt>
   0x00000000004008cd <+53>:    mov    $0x0,%edi
   0x00000000004008d2 <+58>:    callq  0x4005d0 <time@plt>
   0x00000000004008d7 <+63>:    mov    %eax,%edi
   0x00000000004008d9 <+65>:    callq  0x4005c0 <srand@plt>
   0x00000000004008de <+70>:    movl   $0x0,-0x14(%rbp)
   0x00000000004008e5 <+77>:    jmp    0x40090a <main+114>
   0x00000000004008e7 <+79>:    mov    -0x14(%rbp),%eax
   0x00000000004008ea <+82>:    lea    0x1(%rax),%edx
   0x00000000004008ed <+85>:    mov    %edx,%eax
   0x00000000004008ef <+87>:    shl    $0x2,%eax
   0x00000000004008f2 <+90>:    add    %edx,%eax
   0x00000000004008f4 <+92>:    add    %eax,%eax
   0x00000000004008f6 <+94>:    mov    %eax,%edx
   0x00000000004008f8 <+96>:    lea    -0x10(%rbp),%rax
   0x00000000004008fc <+100>:   mov    %edx,%esi
   0x00000000004008fe <+102>:   mov    %rax,%rdi
   0x0000000000400901 <+105>:   callq  0x40071e <enqueue>
   0x0000000000400906 <+110>:   addl   $0x1,-0x14(%rbp)
   0x000000000040090a <+114>:   cmpl   $0x1,-0x14(%rbp)
   0x000000000040090e <+118>:   jle    0x4008e7 <main+79>
   0x0000000000400910 <+120>:   mov    -0x10(%rbp),%rax
   0x0000000000400914 <+124>:   mov    %rax,%rdi
   0x0000000000400917 <+127>:   callq  0x400855 <print_queue>
   0x000000000040091c <+132>:   mov    $0x0,%eax
   0x0000000000400921 <+137>:   mov    -0x8(%rbp),%rcx
   0x0000000000400925 <+141>:   xor    %fs:0x28,%rcx
   0x000000000040092e <+150>:   je     0x400935 <main+157>
   0x0000000000400930 <+152>:   callq  0x400590 <__stack_chk_fail@plt>
   0x0000000000400935 <+157>:   leaveq
   0x0000000000400936 <+158>:   retq
```

35.파이프라인은 점프연산 할 때 깨진다.

36. 메모리 계층구조에는 Stack Heap Data Text 가 존재하는데,
Stack 은 지역변수가 위치하는 영역이고,
Heap 은 동적할당된 것들이 위치하는 영역이고,
Data 는 전역변수 및 Static 으로 선언된 것들이 위치한 영역이고
초기화 되지 않은것들은 모두 0 으로 저장된다.
Text 는 Machine code 가 위치하는 영역이다.

37.c 언어의 기본 메모리 구조는 Stack 영역에 위치하고, 함수호출시 push 와 pop 으로 시작하고 종료한다.

38.메인메모리의 주소는 한정적인데 표현해야 할 주소는 많다 그래서 나온것이 가상메모리의 주소인데
이것을 사용하면 부족한 주소를 처리하게 하고 원활하게 한다.

39.
```
#include<stdio.h>
#include<time.h>
typedef struct man{
```

```
	int benefit;
	char name[20];
}Man;

int main(void){

	int max = 0, i;

	srand(time(NULL));
	Man man1 = {((rand()%10)*100),"minchul1"};
	Man man2 = {((rand()%10)*100),"minchul2"};
	Man man3 = {((rand()%10)*100),"minchul3"};


	printf("급여평균은 = %d ",(man1.benefit+man2.benefit+man3.benefit)/3);
	for(i = 0; i < 3 ; i++){
		if(man1.benefit > max)
		max = man1.benefit;
	}
	printf("최고급여는 = %d 이다\n",max);


	return 0;
}
```

40.리눅스에서 콘솔창에 gcc -g (디버깅할 파일 이름) 을 적고 엔터를 치면
현재 디렉토리에 a.out 이라는 파일이 생기는데 그것을 콘솔창에
gdb (a.out) 이라고 치면 된다 그러면 디버그 할 수 있는 화면이 나온다.
근데 a.out 으로 모든파일의 실행파일이 나오면 나중에 어떤 파일인지 구분하기 힘들다 그래서
customizing 하는데 이것을 하게되면 이름도 바꿀 수 있다.
Gcc -g -o (a.out 을 대신할 이름) (디버깅할 파일 이름)
이렇게하면 디버깅 할 수 있는 실행파일이 생긴다.
그러면 바뀐 파일을 가지고 gdb (바뀐파일이름) 을 치고 들어가면
콘솔창에 (gdb)가 기본적으로 앞에 나온다 그 앞에 r 을 하면 프로그램이 실행되며 디버깅 할 때
어셈블리 코드로 보고싶다면 disas 를 치고 코드 라인을 보고싶다면 l brakepoint 의 위치를
메인으로 위치하게 할 때에는 b main 원하는 위치로 바꾸고 싶을 때에는 b * (원하는 위치의 주소)를 치면 된다.

41.
```
#include<stdio.h>
#include<time.h>
#include<malloc.h>
#include<stdlib.h>
#define EMPTY 0

struct node{
	int data;
	struct node *link;
};

typedef struct node Stack;

Stack *get_node()
{
	Stack* tmp;
```

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

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

int pop(Stack **top)
{
	Stack *tmp;
	int num;
	tmp = *top;
	if(*top == EMPTY)
	{
		printf("Stack is empty!!!\n");
		return 0;
	}

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

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

42. ㅐ
43. ㅐ
44. 명령모드에서 v 를 누르고 전체를 드래그 한 후에 = 을 누르면 정리정돈된다.
45. gcc -g -o0 (디버깅 파일명) (원래파일명) 하면 최적화가 이루어진 컴파일이 이루어지고
    gcc -g (원래파일명) 하면 최적화가 이루어지지 않고 컴파일이 이루어진다.
46. ㅜ
47. ㅐ

48. ㅖ
49. ㅖ
50. ㅖ
51. gdb 는 디버깅을 사용하기위해 쓰는 명령어이다. ex) gdb (실행파일명)
52. 기계어 레벨에서 스택을 조정하는 명령어는 push pop sub mov 가있다.
53.p
54.c 언어 가위바위보 코드

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main()
{
 int a, s;
 printf("===가위바위보 게임===\n");
 printf("게임 방법\n");
 printf("가위=1\n");
 printf("바위=2\n");
 printf("보=3\n");
 srand(time(NULL));
 s = (rand() % 3 + 1);
 scanf("%d", &a);
 {
  if (a == 1 & s == 2)
   printf("졌습니다.\n");
  else if (a == 1 & s == 1)
   printf("비겼습니다.\n");
  else if (a == 1 & s == 3)
   printf("이겼습니다.\n");
  else if (a == 2 & s == 3)
   printf("졌습니다.\n");
  else if (a == 2 & s == 2)
   printf("비겼습니다.\n");
  else if (a == 2 & s == 1)
   printf("이겼습니다.\n");
  else if (a == 3 & s == 1)
   printf("졌습니다.\n");
  else if (a == 3 & s == 3)
   printf("비겼습니다.\n");
  else if (a == 3 & s == 2)
   printf("이겼습니다.\n");
 }
```

```
if (s == 1)
{
 if (a == 1)
 {
  printf("나:가위\n");
  printf("COM:가위\n");
 }
 if (a == 2)
 {
  printf("나:바위\n");
  printf("COM:가위\n");
 }
 if (a == 3)
 {
  printf("나:보\n");
  printf("COM:가위\n");
 }
}
else if (s == 2)
{
 if (a == 1)
 {
  printf("나:가위/n");
  printf("COM:바위\n");
 }
 if (a == 2)
 {
  printf("나:바위\n");
  printf("COM:바위\n");
 }
 if (a == 3)
 {
  printf("나:보\n");
  printf("COM:바위\n");
 }
}
else if (s == 3)
{
 if (a == 1)
 {
  printf("나:가위\n");
  printf("COM:보\n");
```

```
  }
  if (a == 2)
  {
   printf("나:바위\n");
   printf("COM:보\n");
  }
  if (a == 3)
  {
   printf("나:보\n");
   printf("COM:보\n");
  }
 }
 return 0;
}
```

55.p
56.p
57.p