

## <자료구조>

[복합문제 1.1] 값이 1 ~ 4096 까지 무작위로 할당되어 배열에 저장되도록 프로그래밍 하시오.  
배열의 크기는 100 개 정도로 잡는다

```
#include
```

```
#include
```

```
#include
```

```
int main(void){
```

```
int i;
```

```
int arr[100];
```

```
    srand(time(NULL));
```

```
    for(i=0;i<100;i++){
```

```
        arr[i] = 1+(rand()%4096);
```

```
        printf("arr[%d] = %d\n",i,arr[i]);
```

```
    }
```

```
    return 0;
```

```
}
```

```
mhn@mhn-Z20NH-AS51B5U:~/test$ gcc arr.c
mhn@mhn-Z20NH-AS51B5U:~/test$ ./a.out
arr[0] = 404
arr[1] = 3134
arr[2] = 738
arr[3] = 903
arr[4] = 2527
arr[5] = 3296
arr[6] = 3955
arr[7] = 2222
arr[8] = 1339
arr[9] = 758
arr[10] = 2611
arr[11] = 1943
arr[12] = 1341
arr[13] = 369
arr[14] = 1607
arr[15] = 1609
arr[16] = 304
arr[17] = 1695
arr[18] = 602
arr[19] = 1666
arr[20] = 771
arr[21] = 1652
arr[22] = 1092
```

```

arr[20] = 771
arr[21] = 1652
arr[22] = 1092
arr[23] = 728
arr[24] = 2417
arr[25] = 3114
arr[26] = 1398
arr[27] = 895
arr[28] = 1594
arr[29] = 1078
arr[30] = 2311
arr[31] = 1997
arr[32] = 115
arr[33] = 3048
arr[34] = 2900
arr[35] = 2642
arr[36] = 2247
arr[37] = 2758
arr[38] = 767
arr[39] = 3585
arr[40] = 3516
arr[41] = 3377
arr[42] = 1432
arr[43] = 760
arr[44] = 3745
arr[45] = 3038
arr[46] = 2368
arr[47] = 4049
arr[48] = 637
arr[49] = 2970
arr[50] = 1618
arr[51] = 1407
arr[52] = 525
arr[53] = 2710
arr[54] = 2135
arr[55] = 2942
arr[56] = 1727
arr[57] = 3532
arr[58] = 3836
arr[59] = 3320

```

```

arr[60] = 514
arr[61] = 2050
arr[62] = 1220
arr[63] = 628
arr[64] = 1002
arr[65] = 23
arr[66] = 3269
arr[67] = 3248
arr[68] = 2781
arr[69] = 4035
arr[70] = 2737
arr[71] = 2200
arr[72] = 3316
arr[73] = 72
arr[74] = 2959
arr[75] = 2964
arr[76] = 3109
arr[77] = 1230
arr[78] = 2916
arr[79] = 3745
arr[80] = 103
arr[81] = 438
arr[82] = 1056
arr[83] = 628
arr[84] = 3147
arr[85] = 3190
arr[86] = 3569
arr[87] = 777
arr[88] = 2625
arr[89] = 3308
arr[90] = 4096
arr[91] = 3138
arr[92] = 1261
arr[93] = 1219
arr[94] = 3766
arr[95] = 2262
arr[96] = 1242
arr[97] = 2938
arr[98] = 1414
arr[99] = 4022
mhn@mhn-Z20NH-ASS1BSU:~/test$ vi
mhn@mhn-Z20NH-ASS1BSU:~/test$

```

[복합문제 1.2] 앞서 만든 코드를 아래 설명을 보고 개량 하도록 한다.

각 배열은 물건을 담을 수 있는 공간에 해당한다.

앞서서 100 개의 공간에 물건들을 담았는데 공간의 낭비가 있을 수 있다.

이 공간의 낭비가 얼마나 발생했는지 파악하는 프로그램을 작성하시오.

[복합문제 1.3] 앞서 만든 코드를 보다 더 개량한다.

문제에서 확장하여 공간을 보다 효율적으로 관리하고 싶어서 4096, 8192, 16384 등의 4096 배수로 크기를 확장할 수 있는 시스템을 도입했다.

이제부터 공간의 크기는 4096 의 배수이고 최소 크기는 4096, 최대 크기는 131072 에 해당한다.

발생할 수 있는 난수는 1 ~ 131072 로 설정하고 이를 효율적으로 관리하는 프로그램을 작성하시오.

(사실 리눅스 커널의 Buddy 메모리 관리 알고리즘임)

문제 4. 이진 트리를 재귀 호출을 사용하여 구현하도록 한다. 재귀 호출로 구현한다.

```
#include
```

```
#include
```

```
struct node{
```

```
int data;
```

```
struct node *left;

struct node *right;

};

typedef struct node tree;

tree *get_node(){

tree *tmp;

tmp = (tree *)malloc(sizeof(tree));

tmp -> right = NULL;

tmp -> left = 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
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,73);

printf("after delete\n");

print_tree(root);

return 0;
}

```

```

mhn@mhn-Z20NH-AS51BSU:~/test$ gcc tree.c
mhn@mhn-Z20NH-AS51BSU:~/test$ ./a.out
data = 50, left = 45, right = 73
data = 45, left = 32, right = 48
data = 32, left = 16, right = 37
data = 16, left = NULL, right = NULL
data = 37, left = NULL, right = NULL
data = 48, left = 46, right = NULL
data = 46, left = NULL, right = 47
data = 47, left = NULL, right = NULL
data = 73, left = NULL, right = 120
data = 120, left = NULL, right = 130
data = 130, left = 127, right = NULL
data = 127, left = 124, right = NULL
data = 124, left = NULL, right = NULL
After Delete
data = 48, left = 45, right = 73
data = 45, left = 32, right = 46
data = 32, left = 16, right = 37
data = 16, left = NULL, right = NULL
data = 37, left = NULL, right = NULL
data = 46, left = NULL, right = 47
data = 47, left = NULL, right = NULL
data = 73, left = NULL, right = 120
data = 120, left = NULL, right = 130
data = 130, left = 127, right = NULL
data = 127, left = 124, right = NULL
data = 124, left = NULL, right = NULL
mhn@mhn-Z20NH-AS51BSU:~/test$

```

문제 5 이진 트리를 재귀 호출 없이 구현하도록 한다.

결과를 확인하는 print 함수(전위, 중위, 후위 순회) 또한 재귀 호출을 수행하면 안됨

```
#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;
}

```

```

mhn@mhn-Z20NH-AS51BSU:~/test$ gcc superfast_tree.c
mhn@mhn-Z20NH-AS51BSU:~/test$ ./a.out
data = 50, left = 45, right = 73
data = 45, left = 32, right = 48
data = 32, left = 16, right = 37
data = 16, left = NULL, right = NULL
data = 37, left = NULL, right = NULL
data = 48, left = 46, right = NULL
data = 46, left = NULL, right = 47
data = 47, left = NULL, right = NULL
data = 73, left = NULL, right = 120
data = 120, left = NULL, right = 130
data = 130, left = 127, right = NULL
data = 127, left = 124, right = NULL
data = 124, left = NULL, right = NULL
After Delete
data = 48, left = 45, right = 73
data = 45, left = 32, right = 46
data = 32, left = 16, right = 37
data = 16, left = NULL, right = NULL
data = 37, left = NULL, right = NULL
data = 46, left = NULL, right = 47
data = 47, left = NULL, right = NULL
data = 73, left = NULL, right = 120
data = 120, left = NULL, right = 130
data = 130, left = 127, right = NULL
data = 127, left = 124, right = NULL
data = 124, left = NULL, right = NULL
mhn@mhn-Z20NH-AS51BSU:~/test$

```

문제 6 AVL 트리를 재귀 호출을 사용하여 구현하도록 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;

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

    srand(time(NULL));

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

    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);

    return 0;
}

```

```

mhn@mhn-Z20NH-AS5185U:~/test$ ./a.out
84 dup! redo rand()
50 dup! redo rand()
64 dup! redo rand()
arr[0] = 91
arr[1] = 1
arr[2] = 50
arr[3] = 57
arr[4] = 53
arr[5] = 56
arr[6] = 34
arr[7] = 84
arr[8] = 64
arr[9] = 85
arr[10] = 2
arr[11] = 52
arr[12] = 49
arr[13] = 17
arr[14] = 61
Insert Rotation!
data = 50
LR Rotation
Insert Rotation!
data = 53
LL Rotation
Insert Rotation!
data = 56
RL Rotation
Insert Rotation!
data = 34
LR Rotation
Insert Rotation!
data = 64
LL Rotation
Insert Rotation!
data = 85
RR Rotation
Insert Rotation!
data = 17
RR Rotation

```

```

data = 53, lev = 5, left = 34, right = 84
data = 34, lev = 3, left = 2, right = 50
data = 2, lev = 2, left = 1, right = 17
data = 1, lev = 1, left = NULL, right = NULL
data = 17, lev = 1, left = NULL, right = NULL
data = 50, lev = 2, left = 49, right = 52
data = 49, lev = 1, left = NULL, right = NULL
data = 52, lev = 1, left = NULL, right = NULL
data = 84, lev = 4, left = 57, right = 91
data = 57, lev = 3, left = 56, right = 64
data = 56, lev = 1, left = NULL, right = NULL
data = 64, lev = 2, left = 61, right = NULL
data = 61, lev = 1, left = NULL, right = NULL
data = 91, lev = 2, left = 85, right = NULL
data = 85, lev = 1, left = NULL, right = NULL

After Delete
Delete Rotation!
data = 57
RL Rotation
data = 53, lev = 4, left = 17, right = 84
data = 17, lev = 3, left = 2, right = 50
data = 2, lev = 2, left = 1, right = NULL
data = 1, lev = 1, left = NULL, right = NULL
data = 50, lev = 2, left = 49, right = 52
data = 49, lev = 1, left = NULL, right = NULL
data = 52, lev = 1, left = NULL, right = NULL
data = 84, lev = 3, left = 61, right = 91
data = 61, lev = 2, left = 56, right = 64
data = 56, lev = 1, left = NULL, right = NULL
data = 64, lev = 1, left = NULL, right = NULL
data = 91, lev = 1, left = NULL, right = NULL
mhn@mhn-Z20NH-AS5185U:~/test$

```

문제 7 Red Black 트리와 AVL 트리를 비교해보도록 한다.

문제 8 난수를 활용하여 Queue 를 구현한다.

중복되는 숫자를 허용하지 않도록 프로그래밍 하시오.

제일 좋은 방법은 배열을 16 개 놓고 `rand() % 16` 을 해서 숫자가 겹치지 않는지 확인하면 된다.

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

struct node{

    int data;
    struct node *link;

};
typedef struct node Queue;

Queue *get_node(){

    Queue *tmp;
    tmp = (Queue *)malloc(sizeof(Queue));
    tmp -> link = NULL;
    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("%d\n",tmp->data);
        tmp = tmp->link;
    }

}
```

```

int main(void){

    Queue *head = NULL;
    int arr[16]={0};
    int i;

    srand(time(NULL));

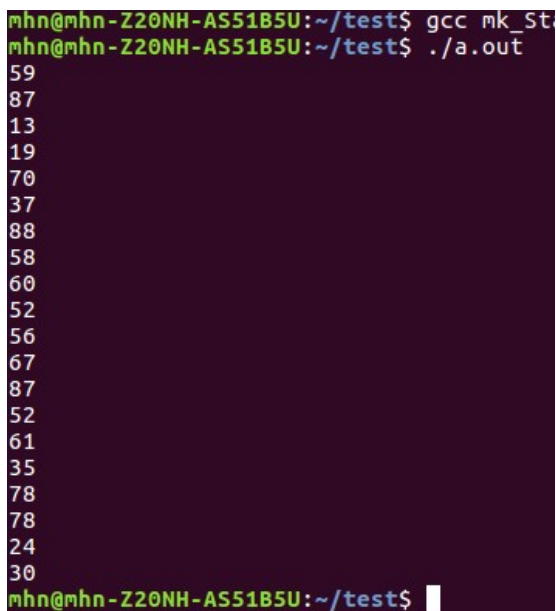
    for(i=0;i<16;i++){
        arr[i] = 1+(rand()%100);
        enqueue(&head,arr[i]);
    }

    print_queue(head);

    return 0;

}

```



```

mhn@mhn-Z20NH-AS51B5U:~/test$ gcc mk_St.c
mhn@mhn-Z20NH-AS51B5U:~/test$ ./a.out
59
87
13
19
70
37
88
58
60
52
56
67
87
52
61
35
78
78
24
30
mhn@mhn-Z20NH-AS51B5U:~/test$

```

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

```

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

struct node{

    int data;
    struct node *link;

};
typedef struct node Queue;

Queue *get_node(){

```

```
Queue *tmp;  
tmp = (Queue *)malloc(sizeof(Queue));  
tmp -> link = NULL;  
return tmp;
```

```
}
```

```
void enqueue(Queue **head,int data){
```

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

```
}
```

```
void dequeue(Queue **head,int data){
```

```
    Queue *tmp;  
    tmp = *head;  
    if((*head) == NULL){  
        printf("Queue is Empty!!!");  
    }  
    if(tmp->data == data){  
        free(tmp);  
        (*head) = tmp->link;  
    }else dequeue(&(tmp->link),data);
```

```
}
```

```
void print_queue(Queue *head){
```

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

```
}
```

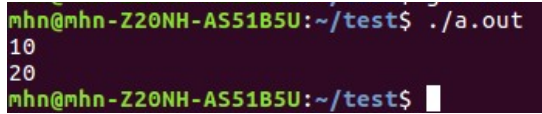
```
int main(void){
```

```
    Queue *head = NULL;  
  
    enqueue(&head,10);  
    enqueue(&head,20);  
    print_queue(head);
```

```

        /*dequeue(&head,20);
        print_queue(head);
*/
        return 0;
}

```



```

mhn@mhn-Z20NH-AS51B5U:~/test$ ./a.out
10
20
mhn@mhn-Z20NH-AS51B5U:~/test$

```

문제 10 [복합문제 2.1] 난수를 활용해서 Stack 을 구성한다.  
같은 숫자가 들어가지 않게 하고 20 개를 집어넣는다.

이때 들어가는 숫자는 1 ~ 100 사이의 숫자로 넣는다.

(마찬가지로 중복되지 않게 한다)

```

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

```

```

struct node{

    int data;

    struct node *link;

};

typedef struct node Stack;

```

```

Stack *get_node(){

    Stack *tmp;

    tmp = (Stack *)malloc(sizeof(Stack));

    tmp->link = NULL;

    return tmp;
}

```

```
}
```

```
void push(Stack **top,int data){
```

```
    Stack *tmp;
```

```
    tmp = *top;
```

```
    (*top) = get_node();
```

```
    (*top)->data=data;
```

```
    (*top)->link = tmp;
```

```
}
```

```
void print_stack(Stack *top){
```

```
    Stack *tmp;
```

```
    tmp = top;
```

```
    while(tmp){
```

```
        printf("%d\n",tmp->data);
```

```
        tmp = tmp->link;
```

```
    }
```

```
}
```

```
int main(void){
```

```
    Stack *top=NULL;
```

```
    int i;
```

```
    int arr[20]={0};
```

```
    srand(time(NULL));
```

```
    for(i=0;i<20;i++){
```

```

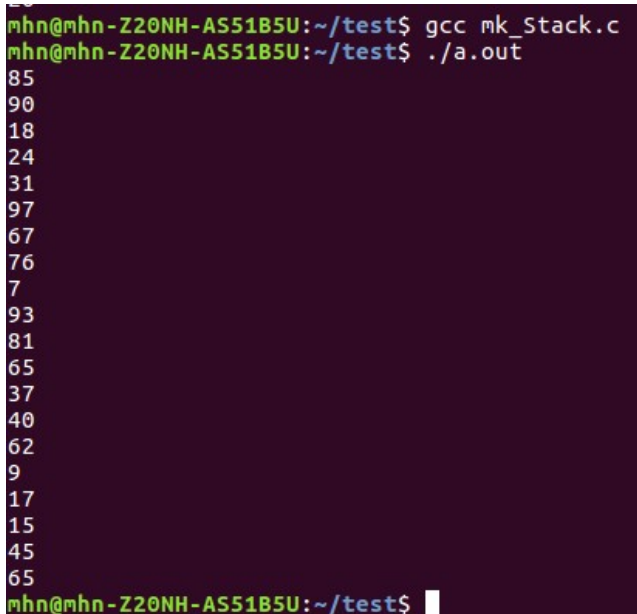
arr[i] = 1+(rand()%100);
push(&top,arr[i]);
}

print_stack(top);

return 0;

}

```



```

mhn@mhn-Z20NH-AS51B5U:~/test$ gcc mk_stack.c
mhn@mhn-Z20NH-AS51B5U:~/test$ ./a.out
85
90
18
24
31
97
67
76
7
93
81
65
37
40
62
9
17
15
45
65
mhn@mhn-Z20NH-AS51B5U:~/test$

```

문제 11 [복합문제 2.2] 2.1 에서 만든 내용중 홀수만 빼내서 AVL 트리를 구성하도록 한다.

문제 12 [복합문제 2.3] 2.1 에서 짝수만 빼내서 RB 트리를 구성하도록 한다.

문제 13 최적화 프로세스를 기술하도록 한다.

루프를 돌 때 goto 를 활용하도록 한다.  
 루프 안에서 조건 만족 시 break 등을 사용하여 반복을 줄이자.

문제 14 기존에는 숫자만 받아봤다.  
 이제 Queue 에서 데이터로서 숫자 값이 아닌 문자열을 받아보도록 하자.

문제 15 AVL 트리에 데이터로서 숫자가 아닌 문자열을 입력하도록 프로그램하시오.

문제 16 Binary Tree 에 문자열을 입력한다.

문제 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

## <임베디드 어플리케이션 분석>

문제 41 아래 Code 를 작성하고 이 Code 의 기계어에 대한 그림을 그리고 분석하시오.

```
void swap(int *a, int *b){
```

```
    int tmp;
```

```
    tmp = *a;
```

```
    *a = *b;
```

```
    *b = tmp;
```

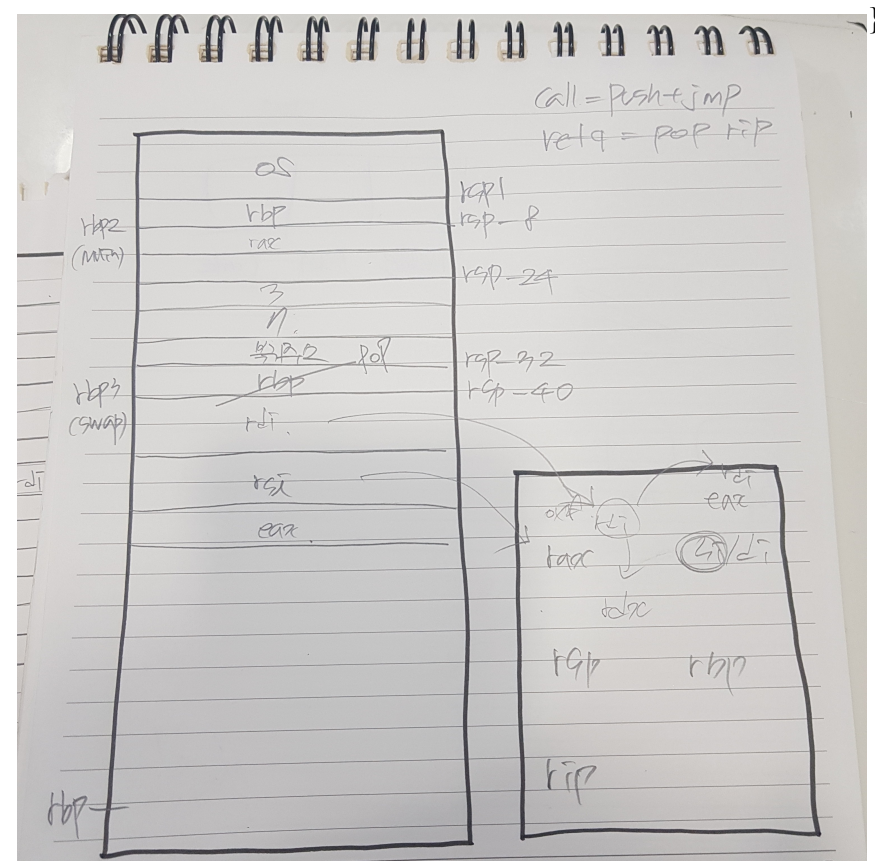
```
}
```

```
int main(void){
```

```
    int num1 = 3, num2 = 7;
```

```
    swap(&num1, &num2);
```

```
    return 0;
```





```

(gdb) disas
Dump of assembler code for function main:
   0x0000000000400573 <+0>:      push    %rbp
   0x0000000000400574 <+1>:      mov     %rsp,%rbp
   0x0000000000400577 <+4>:      sub     $0x10,%rsp
=>  0x000000000040057b <+8>:      mov     %fs:0x28,%rax
   0x0000000000400584 <+17>:     mov     %rax,-0x8(%rbp)
   0x0000000000400588 <+21>:     xor     %eax,%eax
   0x000000000040058a <+23>:     movl    $0x3,-0x10(%rbp)
   0x0000000000400591 <+30>:     movl    $0x7,-0xc(%rbp)
   0x0000000000400598 <+37>:     lea     -0xc(%rbp),%rdx
   0x000000000040059c <+41>:     lea     -0x10(%rbp),%rax
   0x00000000004005a0 <+45>:     mov     %rdx,%rsi
   0x00000000004005a3 <+48>:     mov     %rax,%rdi
   0x00000000004005a6 <+51>:     callq   0x400546 <swap>
   0x00000000004005ab <+56>:     mov     $0x0,%eax
   0x00000000004005b0 <+61>:     mov     -0x8(%rbp),%rcx
   0x00000000004005b4 <+65>:     xor     %fs:0x28,%rcx
   0x00000000004005bd <+74>:     je      0x4005c4 <main+81>
   0x00000000004005bf <+76>:     callq   0x400420 <__stack_chk_fail@plt>
   0x00000000004005c4 <+81>:     leaveq  %rsp
   0x00000000004005c5 <+82>:     retq

```

push %rbp → 현재 Stack Top 을 8byte 아래로 확장 후 그 공간에 Stack Frame 의 base 를 저장  
 mov %rsp, %rbp → 현재 Stack Top 을 rbp Register 에 저장  
 sub \$0x10,%rsp → rsp 를 16 바이트만큼 공간 확보  
 mov %rax, -0x8(%rbp) → rax 를 0x8 만큼 확보한 rbp 에 넣음  
 movl \$0x3, -0x10(%rbp) → 0x3(10 진수로 3)을 -0x10 만큼 확보한 rbp 에 넣음  
 movl \$0x7, -0xc(%rbp) → 0x7(10 진수로 7)을 -0xc 만큼 확보한 rbp 에 넣음  
 callq 0x400546 <swap> → call = push + jmp 이다. 다음 복귀주소를 메모리에 넣고 swap 함수로 점프한다.

```

(98b) <cs>
Dump of assembler code for function swap:
=> 0x0000000000400546 <+0>:      push    %rbp
    0x0000000000400547 <+1>:      mov     %rsp,%rbp
    0x000000000040054a <+4>:      mov     %rdi,-0x18(%rbp)
    0x000000000040054e <+8>:      mov     %rsi,-0x20(%rbp)
    0x0000000000400552 <+12>:     mov     -0x18(%rbp),%rax
    0x0000000000400556 <+16>:     mov     (%rax),%eax
    0x0000000000400558 <+18>:     mov     %eax,-0x4(%rbp)
    0x000000000040055b <+21>:     mov     -0x20(%rbp),%rax
    0x000000000040055f <+25>:     mov     (%rax),%edx
    0x0000000000400561 <+27>:     mov     -0x18(%rbp),%rax
    0x0000000000400565 <+31>:     mov     %edx,(%rax)
    0x0000000000400567 <+33>:     mov     -0x20(%rbp),%rax
    0x000000000040056b <+37>:     mov     -0x4(%rbp),%edx
    0x000000000040056e <+40>:     mov     %edx,(%rax)
    0x0000000000400570 <+42>:     nop
    0x0000000000400571 <+43>:     pop     %rbp
    0x0000000000400572 <+44>:     retq
End of assembler dump.

```

push %rbp → 현재 Stack Top 을 8byte 아래로 확장 후 그 공간에 Stack Frame 의 base 를 저장  
 mov %rsp, %rbp → 현재 Stack Top 을 rbp Register 에 저장  
 mov %rdi, -0x18(%rbp) → rdi 의 값을 0x18 만큼 rbp 에서 공간을 확보한 후 그 공간에 넣는다.  
 mov %rsi, -0x20(%rbp) → rsi 의 값을 0x20 만큼 rbp 에서 공간을 확보한 후 그 공간에 넣는다.  
 mov -0x8(%rbp), %rax → 0x8 만큼 확보한 rbp 에 있는 값을 rax 에 넣음.  
 mov %rax, %eax → rax 의 값을 eax 에도 넣는다.  
 mov %eax, -0x4(%rbp) → eax 의 값을 0x4 의 크기만큼 rbp 에서 공간을 확보한 후 넣는다.  
 mov -0x20(%rbp), %rax → 0x20 만큼 확보한 rbp 에 있는 값을 rax 에 넣음.  
 mov %rax, %edx → rax 의 값을 edx 에도 넣는다.  
 mov -0x18(%rbp), %rax → 0x18 만큼 확보한 rbp 에 있는 값을 rax 에 넣음.  
 mov %edx, %rax → edx 의 값을 rax 에도 넣는다.  
 mov -0x20(%rbp), %rax → 0x20 만큼 확보한 rbp 에 있는 값을 rax 에 넣음.  
 mov -0x4(%rbp), %edx → 0x8 만큼 확보한 rbp 에 있는 값을 edx 에 넣음.  
 mov %edx, (%rax) → edx 의 값을 rax 에 넣는다  
 pop %rbp → rbp 를 해제한다  
 retq → pop + rip 로 스택프레임 해제 후 복귀주소로 돌아간다.

문제 40 한달간 적은 시간이기도 하고 많은 시간이기도 했다.

한 달간 수업을 진행하면서 본인이 느낀점을 20 줄 이상 최대한 상세하게 기술하시오.

또한 앞으로에 대한 포부 또한 기술하길 바란다.

그리고 앞으로 어떤 일을 하고 싶은지 상세히 기술하도록 한다.

한달동안 배움의 기쁨보다는 실력적으로 너무 부족한 것 같아서 불안하고 자괴감을 많이 느꼈다.

선생님께서 지금 모른다고 좌절하지 말라고 하셨지만 사실 많이 좌절했다.

하지만 포기하지 않고 끝까지 할 생각이다. 7 개월 후에는 자신감으로 가득 차 있었으면 좋겠다.

문제 39  $\sin(x)$  값을 프로그램으로 구현해보자.

프로그램으로 구현한 결과와 실제 `sin()` 함수와 결과가 어떻게 차이가 나는지 비교해보도록 하자.

문제 38 함수포인터를 활용해보자.

등차 수열의 합을 구하는 프로그램을 for 문을 도는 방식으로 구현하고

등차 수열의 합 공식을 활용하여 구현해보자.

함수 포인터로 각각의 실행 결과를 출력하고 이 둘의 결과가 같은지 여부를 파악하는 프로그램을 작성하라.

문제 37 화면의 크기가 가로 800, 세로 600 이다.

여기서 표현되는 값은 x 값이 [-12 ~ +12] 에 해당하며

y 값은 [-8 ~ +8] 에 해당한다.

x 와 y 가 출력하게될 값들을 800, 600 화면에 가득차게 표기할 수 있는 스케일링 값을 산출하도록 프로그래밍 하시오.

문제 36 가위 바위 보 게임을 만들어보자.

프로그램을 만들고 컴퓨터랑 배틀 뜨자!

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

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

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

```
void game(int a,int you){
```

```
    char *arr[3]={"가위","바위","보"};
```

```
    int i;
```

```
    int com;
```

```
    srand(time(NULL));
```

```
    com=rand()%3+1;
```

```
    printf("컴퓨터는 %s, 당신은 %s\n", arr[com-1], arr[you-1]);
```

```
    printf("결과 : ");
```

```
    if (com==1)
```

```
        if (you==1) printf("비겼습니다.\n");
```

```
        else if (you==2) printf("당신이 이겼습니다.\n");
```

```
        else printf("컴퓨터가 이겼습니다.\n");
```

```
    else if (com==2)
```

```
        if (you==1) printf("컴퓨터가 이겼습니다.\n");
```

```
        else if (you==2) printf("비겼습니다.\n");
```

```
        else printf("당신이 이겼습니다.\n");
```

```
    else
```

```
        if (you==1) printf("당신이 이겼습니다.\n");
```

```
        else if (you==2) printf("컴퓨터가 이겼습니다.\n");
```

```
        else printf("비겼습니다.\n");
```

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

```

}

int main(void)
{
    int a,you;
    int i;

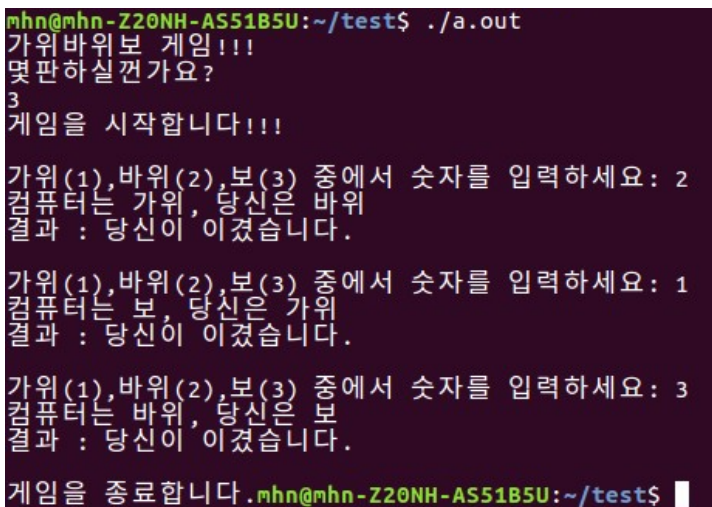
    srand(time(NULL));

    printf("가위바위보 게임!!!\n");
    printf("몇판하실건가요? \n");
    scanf("%d",&a);

    printf("게임을 시작합니다!!!\n\n");

    for(i=0;i<a;i++){
        printf("가위(1),바위(2),보(3) 중에서 숫자를 입력하세요: ");
        scanf("%d", &you);
        game(a,you);
    }
    printf("게임을 종료합니다.");
    return 0;
}

```



```

mhn@mhn-Z20NH-AS51B5U:~/test$ ./a.out
가위바위보 게임!!!
몇판하실건가요?
3
게임을 시작합니다!!!

가위(1),바위(2),보(3) 중에서 숫자를 입력하세요: 2
컴퓨터는 가위, 당신은 바위
결과 : 당신이 이겼습니다.

가위(1),바위(2),보(3) 중에서 숫자를 입력하세요: 1
컴퓨터는 보, 당신은 가위
결과 : 당신이 이겼습니다.

가위(1),바위(2),보(3) 중에서 숫자를 입력하세요: 3
컴퓨터는 바위, 당신은 보
결과 : 당신이 이겼습니다.

게임을 종료합니다. mhn@mhn-Z20NH-AS51B5U:~/test$

```

문제 35 a 좌표(3, 6), b 좌표(4, 4) 가 주어졌다.

원점으로부터 이 두 좌표가 만들어내는 삼각형의 크기를 구하는 프로그램을 작성하라.

문제 34 기계어 레벨에서 스택을 조정하는 명령어는 어떤것들이 있는가 ?

직접 연산을 하는 경우를 제외하고 기술하도록 한다. (4 가지)

Push

mov

call

pop

문제 33 gdb 를 사용하는 이유를 기술하시오.

디버깅은 문법적인 오류가 없어 컴파일은 성공적이거나 논리적인 오류가 존재할 때 그 오류를 찾기 위해 수행한다.

논리적인 오류 외에도 예측하지 못한 다양한 문제를 파악하고 처리하기 위해 사용하기도 한다.

문제 32 프로그램을 최적화하는 컴파일 옵션을 적고  
반대로 어떠한 최적화도 수행하지 않는 컴파일 옵션을 적도록 한다.

**gcc -g -o 실행파일명 소스파일명.c**

어떠한 최적화도 수행하지 않는 컴파일 옵션 : **gcc -g -o0 -o 실행파일명 소스파일명.c**

문제 31 vi 에서 코드가 정렬이 잘 안되어 있다.  
이 상황에서 어떻게 해야 예쁘고 깔끔하게 정렬되는가 ?

명령모드에서 **v** 를 누르고 정렬할 범위를 지정한다.

**ex)**전부 정리하려면 맨 위에서 **v** 를 누르고 끝까지 쪽 내려오면 된다.

**enter** 키를 누른다. 예쁘고 깔끔하게 정렬이 되어있다!

문제 30 리눅스에서 디버깅을 수행하기 위한 프로그램 컴파일 방법을 기술하시오.

컴파일 시 디버깅 옵션을 추가한다: **gcc -g -o 실행파일명 소스파일명.c**

디버거를 켜다 : **gdb 실행파일명**

문제 29 이름과 급여를 저장하도록 만든다.  
이름은 마음대로 고정시키도록 하고 급여는 rand() 를 활용

급여의 평균을 출력하고 가장 높은 한 사람의 이름과 급여를 출력하시오.

(값이 같을 수 있음에 유의해야 한다)

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

문제 28 디버깅하면서 보는 변수의 주소는 무엇일까 ?

문제 27 C 언어의 기본 메모리 구조에 대해 기술하시오.

메모리는 지역변수를 저장하고 있는 **stack** 과동적할당된 메모리는 가지는 **Heap**, 전역변수와 상수를 담고있는 **Data**, 마지막으로 **machine code** 가 위치하고 있는 **Text** 가 있다.

문제 26 메모리 계층 구조에 대해 기술하시오.  
레지스터,캐시,메모리 ,하드 디스크 순으로 점점 느려진다.

레지스터와 캐시는 **CPU** 내부에 존재하여 빠른 접근이 가능하지만 메모리는 **CPU** 외부에 존재하기 때문에 느리게 접근할 수 밖에 없다. 하드디스크는 **CPU** 가 직접 접근할 수 없다.

문제 25 파이프라인은 언제 깨지는가 ?  
분기 명령어를 사용할 때 깨진다. **ex)call,jmp**

문제 24 함수 포인터의 정의를 기술하시오.  
함수 포인터란 함수에 대한 메모리 주소를 담을 수 있는 것이다.

문제 23 포인터의 정의를 기술하시오.  
포인터란 메모리에 주소를 저장할 수 있는 공간이다

문제 22 변수의 정의를 기술하시오.  
변수란 메모리에 정보를 저장할 수 있는 공간이다

문제 21 단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.  
덧셈 혹은 뺄셈 같은 기능이 아님

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

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

```
int main(void) {
```

```
    int i;
```

```
    char str[] = "abcDEFgH";
```

```
    for(i=0;str[i];i++){
```

```
        printf("%c\n", str[i]^32);
```

```
    }
```

```
    return 0;
```

```
}  
mhn@mhn-Z20NH-AS51B5U:~/test$ ./a.out  
A  
B  
C  
d  
e  
f  
G  
h  
mhn@mhn-Z20NH-AS51B5U:~/test$
```

문제 20 임의의 값 x 가 있는데, 이를 134217728 단위로 정렬하고 싶다면 어떻게 해야할까 ?  
어떤 숫자를 넣던 134217728 의 배수로 정렬이 된다는 뜻임

(힌트 :  $134217728 = 2^{27}$ )

문제 19 `int *p[3]` 와 `int (*p)[3]` 는 같은 것일까 ? 다른 것일까 ? 이유를 함께 기술하도록 한다.

`int *p[3]` 와 `int (*p)[3]`는 다르다.

`int *p[3]` => p 는 int 형을 가리키는 포인터를 3 개 저장하는 배열이고

`int (*p)[3]` => p 는 int 형 3 개를 저장하고 있는 배열을 가리키는 포인터이다

문제 18 `char *str = "WTF, Where is my Pointer ? Where is it ?"` 라는 문자열이 있다.  
여기에 소문자가 총 몇 개 사용되었는지 세는 프로그램을 만들어보자!

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

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

```
int countt(char **str){

    int i,count=0;

    for(i=0;(*str)[i];i++){

        if((( *str)[i])>='a' && (( *str)[i])<='z'){

            count++;

        }

    }

    printf("%d\n",count);

}
```

```

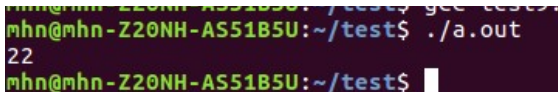
int main(){

    char *str = "WTF, Where is my Pointer ? Where is it ?";

    countt(&str);

return 0;
}

```



```

mhn@mhn-Z20NH-AS51B5U:~/test$ ./a.out
22
mhn@mhn-Z20NH-AS51B5U:~/test$

```

문제 17 아래와 같은 행렬을 생각해보자!

2 4 6

2 4 6

sapply(arr, func) 으로 위의 행렬을 아래와 같이 바꿔보자!

2 4 6

4 16 36

sapply 함수를 위와 같이 구현하라는 의미다.

(R 이나 python 같은 언어에서 지원되는 기법중 하나에 해당한다)

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

문제 16 네이버의 쓰레기같은 사다리 게임을 우리끼리 즐길 수 있는 것으로 대체하는 프로그램을 만들어보자.

우리반 학생들은 모두 25 명이다.

반 학생들과 함께 진행할 수 있는 사다리 게임을 만들도록 한다.

참여 인원수를 지정할 수 있어야하며 사다리 게임에서 걸리는 사람의 숫자도 조정할 수 있도록 만든다.

문제 15 Intel Architecture 와 ARM Architecture 의 차이점은 ?

Intel Architecture 는 함수의 입력을 스택으로 전달한다.

ARM 의 경우엔 함수 입력을 4 개까진 레지스터로 처리하고 4 개가 넘어갈 경우엔 스택에 집어넣는다.



또한 Intel Architecture 는 메모리에서 메모리로 연산이 가능하지만 ARM Architecture 은 반도체의 다 이 사이즈가 작아서 메모리에서 메모리로 연산이 불가능하다 그래서 ARM 은 먼저 메모리에서 레지스 터로 값을 옮기고 다시 이 레지스터 값을 메모리로 옮기는 작업을 한다(그러므로 성능을 높이고자 한다면 ARM 에서는 함수 입력을 4 개 이하로 만드는 것이 좋다)

문제 14 1, 4, 5, 9, 14, 23, 37, 60, 97, ... 형태로 숫자가 진행된다.  
23 번째 숫자는 무엇일까 ?

(프로그래밍 하시오)

문제 13 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... 형태로 숫자가 진행된다.  
1 ~ 27 번째까지의 수들로 홀수들의 합을 하고 짝수들의 합을 구한다.

홀수들의 합 - 짝수들의 합의 결과를 출력하시오.

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

문제 12 함수 포인터를 활용하여 float 형 3 by 3 행렬의 덧셈과 int 형 3 by 3 행렬의 덧셈을 구현하도록 하시오.

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

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

```
void intt(int (*a)[3],int (*b)[3]){  
  
    int i,j;  
  
    printf("int 형 3x3 행렬 곱셈\n");  
  
    for(i=0;i<3;i++){  
        for(j=0;j<3;j++){  
            printf("%d\t",a[i][j] * b[i][j]);  
        }  
        printf("\n");  
    }  
}
```

```
void floatt(float (*c)[3],float (*d)[3]){  
  
    int i,j;  
  
    printf("float 형 3x3 행렬 곱셈\n");
```

```

for(i=0;i<3;i++){
    for(j=0;j<3;j++){
        printf("%lf\t",c[i][j] * d[i][j]);
    }
    printf("\n");
}

}

int main(void){

    int a[][3] = {{2,3,1},{3,4,2},{2,2,1}};
    int b[][3] = {{3,1,2},{3,1,1},{1,3,2}};

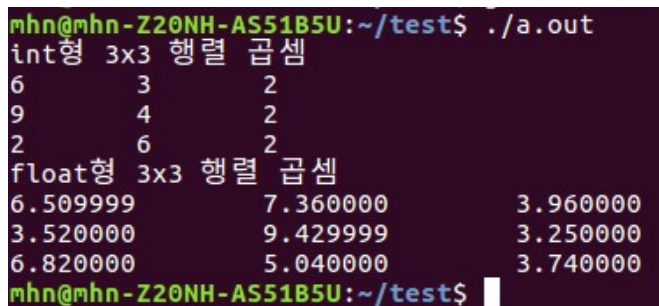
    float c[][3] = {{2.1,3.2,3.3},{3.2,4.1,1.3},{2.2,2.4,1.1}};
    float d[][3] = {{3.1,2.3,1.2},{1.1,2.3,2.5},{3.1,2.1,3.4}};

    intt(a,b);
    floatt(c,d);

    return 0;

}

```



```

mhn@mhn-Z20NH-AS51B5U:~/test$ ./a.out
int형 3x3 행렬 곱셈
6      3      2
9      4      2
2      6      2
float형 3x3 행렬 곱셈
6.509999 7.360000 3.960000
3.520000 9.429999 3.250000
6.820000 5.040000 3.740000
mhn@mhn-Z20NH-AS51B5U:~/test$

```

문제 11  $4x^2 + 5x + 1$  을 1 ~ 3 까지 정적분하는 프로그램을 구현하도록 한다.

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

문제 10 파이프라인이 깨지는 경우 어떠한 이점이 있는지 기술하시오.

문제 9 함수 포인터를 반환하고 함수 포인터를 인자로 취하는 함수의 주소를 반환하고 인자로 int 2 개를 취하는 함수를 작성하도록 한다.

(프로토타입이 각개 다를 수 있으므로 프로토타입을 주석으로 기술하도록 한다)

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

문제 8 다음 프로토타입을 기술하시오.

void (\* signal(int signum, void (\* handler)(int)))(int)

리턴 : void (\*)(int)

이름 : signal

인자 : int signum, void(\*handler)(int)

문제 7 이중 배열을 함수의 인자로 입력 받아 3 by 3 행렬의 곱셈을 구현하시오.

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

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

```
void mul(int a[][3],int b[][3]){
```

```
    int i,j;
```

```
    printf("3x3 행렬 곱셈\n");
```

```
    for(i=0;i<3;i++){
```

```
        for(j=0;j<3;j++){
```

```
            printf("%d\t",a[i][j] * b[i][j]);
```

```
        }
```

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

```
    }
```

```
}
```

```
int main(void){
```

```
    int a[][3] = {{2,3,5},{3,4,6},{2,2,1}};
```

```
    int b[][3] = {{3,4,2},{3,4,1},{1,3,4}};
```

```
    mul(a,b);
```

```
    return 0;
```

```
}
```

```
mhn@mhn-Z20NH-AS51B5U:~/test$ ./a.out
3x3 행렬 곱셈
6      12      10
9      16      6
2      6       4
mhn@mhn-Z20NH-AS51B5U:~/test$ vi mul.c
mhn@mhn-Z20NH-AS51B5U:~/test$
```

문제 6 위의 문장에서 Safety MCU 가 몇 번째 부터 시작하는지 찾아내도록 프로그래밍 해보자.

TI Cortex-R5F Safety MCU is very good to Real-Time System.

(이정도는 가볍게 해야 파싱 같은것도 쉽게 할 수 있다)

무능력자가 되지 않기 위해서라도 이정도는 가볍게 할 수 있어야 한다.

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

문제 5 포인터 크기에 대해 알고 있는대로 기술하시오.

포인터의 크기는 **H/W** 가 몇 비트를 지원하느냐에 따라 달라진다.

포인터는 메모리의 어떤 주소값이든 접근할 수 있어야 하기 때문이다.(64 비트 시스템에서는 주소 값을 64비트로 표현하기 때문에 포인터의 크기가 8바이트이다)

문제 4 전세계 각지의 천재들이 모여서 개발하는 리눅스 운영체제 코드에는 엄청 많은 양의 goto 가 사용되고 있다.

goto 를 도대체 왜 사용해야만 할까 ?

효율성 때문이다.

예를들어 다중루프를 탈출해야 할 때를 생각할 수 있는데, 만약 for 문이 여러 개 생겨 if,break 조합의 경우 for 문의 갯수 만큼 mov,cmp,jmp 를 해야 한다.

하지만 goto 는 jmp 명령어 하나만 사용하기 때문에 효율성이 뛰어나고, 그렇기 때문에 성능면에서도 훨씬 좋다

문제 3 12 비트 ADC 를 가지고 있는 장치가 있다.

보드는 12 V 로 동작하고 있고 ADC 는 -5 ~ 5 V 로 동작한다.

ADC 에서 읽은 값이 2077 일 때 이 신호를 디지털 관점에서 재해석하도록 프로그래밍 한다.

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

문제 2 배열에 아래와 같은 정보들이 들어있다.

문제 1 이것이 없으면 사실상 C 언어를 사용할 수 없다.

C 언어에서 함수를 호출하기 위해서도 이것은 반드시 필요하다.

이와 같은 이유로 운영체제의 부팅 코드에서도 이것을 설정하는 작업이 진행되는데 이것은 무엇일까 ?

스택