Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

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

2. 배열에 아래와 같은 정보들이 들어있다.
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, 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,

cnt 를 입력 함수에서 소팅하여 차례로 **3** 개를 출력하면 된다**. (**시간부족**)**

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

typedef struct _queue{
    int data;
    int cnt;
    struct _queue* link;
}queue;
queue* get_node(){
    queue *tmp = (queue*)malloc(sizeof(queue));
    tmp->cnt = 0;
    tmp->link = NULL;
}
queue* enqueue(queue *head, int data){

    if(head==NULL){
        head=get_node();
        head->data=data;
        return head;
    }
    if(head->data == data){
        head->cnt ++;
        return head;
    }
    else{
        head->link=enqueue(head->link,data);
```

```
    }
    return head;
}

void printf_queue(queue *head){
    if(head==NULL)
        return;
    printf("%d %d\n",head->data,head->cnt);
    printf_queue(head->link);
}

void printf_major(queue *head){
    int arr[3]={0};
    int tmp;

    if(head == NULL)
        return;



}

int main(void){

    int arr[]={2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400,
2400, 2400, 2400, 2400, 2400, 2400, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000,
5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 500, 500, 500, 500, 500, 500, 500,
500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 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, 978, 456, 234756 , 234 ,4564 ,3243, 876,645, 534, 423, 312, 756, 235 ,75678};
    queue *head = NULL;

    for(int i=0;i<sizeof(arr)/sizeof(int);i++){
        head=enqueue(head,arr[i]);
```

```
    }

    printf_queue(head);

    return 0;
}
```

여기서 가장 빈도수가 높은 3 개의 숫자를 찾아 출력하시오!
함수에서 이 작업을 한 번에 찾을 수 있도록 하시오.
(찾는 작업을 여러번 분할하지 말란 뜻임)

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.

```
#include <stdio.h>

int main(void){

    int data = 2077;
    int v;
    data = data-(0x1000/2);
    v=data&(~127);
    v=v/1024;

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


    return 0;
}
```

4. 전세계 각지의 천재들이 모여서 개발하는 리눅스 운영체제 코드에는
엄청나게 많은 양의 goto 가 사용되고 있다.
goto 를 도대체 왜 사용해야만 할까 ?

복잡한 반복문에서 조건이 만족되었을 때, 한번에 탈출하기 위한 것이다.

5. 포인터 크기에 대해 알고 있는대로 기술하시오.

포인터의 크기는 컴퓨터의 메모리 **bit** 수에 따른다. 컴퓨터의 산술 연산이 **ALU** 에 의존적이기 때문이다.
32 비트의 경우 **4 byte** 크기 이며 그 만큼의 주소를 표현하기 위해서 포인터의 크기도 **4** 바이트가 되어야 한다.

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

위의 문장에서 Safety MCU 가 몇 번째 부터
시작하는지 찾아내도록 프로그래밍 해보자.
(이정도는 가볍게 해야 파싱 같은것도 쉽게 할 수 있다)

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

void make_str(char **str, int size){

        *str = (char*)malloc(size+1);
}


int main(void){
   char str[]="TI Cortex-R5F Safety MCU is very good to Real-Time System";
   char *tmp=(char*)malloc(sizeof(str));
   int i;
   int cnt=0;
```

```
    for(i=0;i<sizeof(str);i++){
        if(str[i]==' '|| str[i]=='\0'){
            tmp[cnt+1]='\0';
            //printf("%s\n",tmp);
            if(strcmp(tmp,"Safety")==0){
                printf("string array[%d] is Starting point!\n",i-6);
            }
            make_str(&tmp,sizeof(str));
            cnt=0;

            continue;
        }
        tmp[cnt]=str[i];
        cnt++;
    }

    return 0;
}
```

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

8. void (* signal(int signum, void (* handler)(int)))(int) 의 프로토타입을 기술하시오.
void (*)(int) signal(int signum, void (* handler)(int))

리턴형은 함수 포인터 이며 리턴이 보이드 **int** 형 인자 하나를 받고
이름은 **signal**
매개변수는 **int** 형인자 하나와 함수 포인터 하나인데 보이드 리턴에 인자를 **int** 로 받는다**.**

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)

// void(*)(void) (*)(void (* handler)(int)) signal(int a, int b)

void(* (* signal(int a, int b))(void (* handler)(int)))(void)

10. 파이프라인이 깨지는 경우 어떠한 이점이 있는지 기술하시오.

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

12. 값이 1 ~ 4096 까지 무작위로 할당되어 배열에 저장되도록 프로그래밍 하시오.
(배열의 크기는 100 개정도로 잡는다)

13. 12 번 문제에서 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.

14. 13 번 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)

15. 이진 트리를 재귀 호출을 사용하여 구현하도록 한다.
(일반적인 SW 회사들 면접 당골 문제 - 이게 되면 큐 따위야 문제 없음)

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

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

```
}tree;

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


int judge_null(tree *root){
   if(root->left == NULL && root->right != NULL){//having right sub
        return 2;
   }
   else if(root->left != NULL && root->right == NULL){//having left sub
        return 1;
   }
   else if(root->left !=NULL && root->right !=NULL){// both no null
        return 3;
   }
   else{// both null
        return 4;
   }
}

tree *findmax(tree *root){//input left address
   if(root->right !=NULL)
        findmax(root->right);
   else
        return root;
}

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);
} //트리 삽입

void print_tree(tree *root){
    if(root){
        printf("%d,\n ",root->data);
        /*
        if(root->left)
            printf("left=%d, ",root->left->data);
        else
            printf("left = NULL, ");

        if(root->right)
            printf("left=%d, ",root->right->data);
        else
            printf("left = NULL, ");
*/
        print_tree(root->left);
        print_tree(root->right);
    }
} // 트리 표현


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

  //  printf("%d\n",judge_null(root));
```

```
    if((*root)->data < data){ //bigger source data
            delete_tree(&(*root)->right,data);//move right
    }
    else if((*root)->data > data){//smaller source data
       delete_tree(&(*root)->left,data); //move left
    }
    else{// same value

          if(judge_null(*root)==1){//left sub
             printf("left sub\n");
             *root=(*root)->left;
          }
          else if(judge_null(*root)==2){//right sub
             printf("right sub\n");
             *root=(*root)->right;
          }
          else if(judge_null(*root)==3){//both sub
             printf("both sub\n");
             tree *tmp=findmax((*root)->left);
             (*root)->data=tmp->data;
             (*root)->left->right=tmp->left;
          }
          else{//no sub
             printf("no sub\n");
             *root=NULL;
             free(*root);
             return;
          }
          return;
    }
}


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);
    printf("\n");

    delete_tree(&root,50);
    print_tree(root);
    return 0;
}
```

16. 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;
}
```

17. AVL 트리를 재귀 호출을 사용하여 구현하도록 한다.

18 번에 **17** 번도 구현완료

18. AVL 트리를 재귀 호출 없이 구현하도록 한다.

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

typedef enum _rot{
   RR,
   RL,
   LR,
   LL
}enum_rot;

typedef struct _avl{
   int data;
   int level;
   struct _avl *left;
   struct _avl *right;
}avl;

typedef struct _stack{
   void **d_p;
   struct _stack *link;
}stack;

void push(stack **top, void **d_p){
```

```
    if(d_p ==NULL)
        return;
    if(*top == NULL){
        *top = (stack*)malloc(sizeof(stack));
        (*top)->d_p = d_p;
        (*top)->link = NULL;
    }
    stack *tmp = (stack*)malloc(sizeof(stack));
    tmp->d_p = d_p;
    tmp->link = *top;
    (*top)=tmp;
}

void** pop(stack **top){
    void **tmp;
    stack * stmp = *top;
    if(*top==NULL){
        return NULL;
    }
    tmp = stmp->d_p;
    (*top)=stmp->link;
    free(stmp);

    return tmp;
}

avl* get_node(){
    avl *tmp = (avl*)malloc(sizeof(avl));
    tmp->left = NULL;
    tmp->right = NULL;
    tmp->level = 1;
    return tmp;
}
```

```
int update_level(avl *root){
    int left;
    int right;
    left = root->left ? root->left->level : 0;
    right = root->right ? root->right->level : 0;
    if(left>right)
        return left+1;
    return right+1;
}

int check_rot(avl *root){
    int left;
    int right;
    left = root->left ? root->left->level : 0;
    right = root->right ? root->right->level : 0;
    return left-right;
}

int check_rot_type(avl* root){
    if(check_rot(root)>0){//L
        if(check_rot(root->left)<0)
            return LR;
        return LL;
    }
    else{//R
        if(check_rot(root->right)>0)
            return RL;
        return RR;
    }
}

avl* RRchange(avl* root){
    avl *tmp = root->right;
    root->right = tmp->left;
```

```
    tmp->left = root;
    //
    root->level = update_level(root);
    tmp->level = update_level(tmp);
    //tmp->left->level = update_level(tmp->left);
    return tmp;
}

avl* LLchange(avl* root){
    avl *tmp = root->left;
    root->left = tmp->right;
    tmp->right = root;
    //
    root->level = update_level(root);
    tmp->level = update_level(tmp);
    //tmp->right->level = update_level(tmp->right);
    return tmp;
}

avl* RLchange(avl* root){
    root->right=LLchange(root->right);
    return RRchange(root);
}

avl* LRchange(avl* root){
    root->left=RRchange(root->left);
    return LLchange(root);
}

avl *rotation(avl* root, int rot_stat){
    if(rot_stat==RR){
        return RRchange(root);
    }
    else if(rot_stat==RL){
```

```
        return RLchange(root);
    }
    else if(rot_stat==LR){
        return LRchange(root);
    }
    else if(rot_stat==LL){
        return LLchange(root);
    }
    else
        printf("what the fuxxx?");
}

void insert_non_cur(avl** root,int val){

    stack * top = NULL;
    avl ** tmp = NULL;

    while(*root !=NULL){
        push(&top,(void**)root);
        if((*root)->data > val)
            root= &(*root)->left;
        else if((*root)->data <= val)
            root = &(*root)->right;
    }

    (*root) = get_node();
    (*root)->data = val;

    while(top !=NULL){
        tmp =(avl**)pop(&top);
        (*tmp)->level = update_level(*tmp);
        if(abs(check_rot(*tmp))>1){
            *tmp = rotation(*tmp,check_rot_type(*tmp));
        }
```

```
    }
}

void insert(avl** root,int val){
    if(*root == NULL){
        *root=get_node();
        (*root)->data=val;
        return;
    }

    if((*root)->data > val){
        insert(&(*root)->left,val);
    }
    else if((*root)->data <= val){
        insert(&(*root)->right,val);
    }

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

    if(abs(check_rot(*root))>1){
        *root = rotation(*root,check_rot_type(*root));
    }

}

void printf_avl(avl *root){
    if(root==NULL)
        return;
    printf("%d ",root->data);
    printf("%d\n",root->level);
    printf_avl(root->left);
    printf_avl(root->right);
}
```

```
void delete(avl **root){

}

int main(void){

    avl *root = NULL;
    int arr[20];
    srand(time(NULL));

   int arr2[] = {57,32,15,7,5,18,28,38,34,45,79,73,93,89,97};
   // int arr2[] = {1,2,3,9,8,7,10,10};

    for(int i=0;i<sizeof(arr)/sizeof(int);i++){
        arr[i]=rand()%200+1;
    }

    for(int i=0;i<sizeof(arr2)/sizeof(int);i++){
        insert_non_cur(&root,arr2[i]);
    }

    printf_avl(root);

    return 0;
}
```

19. Red Black 트리와 AVL 트리를 비교해보도록 한다.

레드블랙 트리와 **AVL** 트리는 탐색 속도는 **AVL** 이 빠르지만
입력 및 삭제의 경우 레드블랙 트리가 더 빠르다**.**
많은 입력 삭제가 필요한 경우 레드블랙 트리가 효율성이 좋다**.**

20. 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 <stdlib.h>
#include <time.h>
#include <stdbool.h>

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

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() % 100 + 1;

         if(is_dup(arr, i))
```

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

queue *get_node(void)
{
        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 = head;

        while(head)
        {
                printf("head->data = %d\n", head->data);
```

```
                head = head->link;
        }
}


int main(void)
{
        int i;

        queue *head = NULL;
        int data[22] = {0};
        int size;
      size =sizeof(data)/sizeof(int) -1;
        srand(time(NULL));
        init_rand_arr(data,size);


        for(i = 0; i < size; i++)
                enqueue(&head,data[i]);

        print_queue(head);

        return 0;
}
```

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

```
#include <stdio.h>

void mat_int_add(int (*mat1)[3], int (*mat2)[3]){
   int tmp[3][3];
   int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r;
```

```
a=mat1[0][0];
b=mat1[0][1];
c=mat1[0][2];
d=mat1[1][0];
e=mat1[1][1];
f=mat1[1][2];
g=mat1[2][0];
h=mat1[2][1];
i=mat1[2][2];


j=mat2[0][0];
k=mat2[0][1];
l=mat2[0][2];
m=mat2[1][0];
n=mat2[1][1];
o=mat2[1][2];
p=mat2[2][0];
q=mat2[2][1];
r=mat2[2][2];


tmp[0][0]=a+j;
tmp[0][1]=b+k;
tmp[0][2]=c+l;
tmp[1][0]=d+m;
tmp[1][1]=e+n;
tmp[1][2]=f+o;
tmp[2][0]=g+p;
tmp[2][1]=h+q;
tmp[2][2]=i+r;

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

```
            printf("%2d ",tmp[i][j]);
        }
        printf("\n");
    }
}

void mat_float_add(float (*mat1)[3], float (*mat2)[3]){
    float tmp[3][3];
    int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r;

    a=mat1[0][0];
    b=mat1[0][1];
    c=mat1[0][2];
    d=mat1[1][0];
    e=mat1[1][1];
    f=mat1[1][2];
    g=mat1[2][0];
    h=mat1[2][1];
    i=mat1[2][2];


    j=mat2[0][0];
    k=mat2[0][1];
    l=mat2[0][2];
    m=mat2[1][0];
    n=mat2[1][1];
    o=mat2[1][2];
    p=mat2[2][0];
    q=mat2[2][1];
    r=mat2[2][2];


    tmp[0][0]=a+j;
    tmp[0][1]=b+k;
```

```
    tmp[0][2]=c+l;
    tmp[1][0]=d+m;
    tmp[1][1]=e+n;
    tmp[1][2]=f+o;
    tmp[2][0]=g+p;
    tmp[2][1]=h+q;
    tmp[2][2]=i+r;

    for(int i=0; i<3;i++){
        for(int j=0;j<3;j++){
            printf("%5f ",tmp[i][j]);
        }
        printf("\n");
    }
}

int main(void){
    void (*fp1) (int (*mat1)[3],int (*mat2)[3]);
    void (*fp2) (float (*mat1)[3],float (*mat2)[3]);

    fp1 = mat_int_add;
    fp2 = mat_float_add;

    int iarr[3][3]={0};
    float farr[3][3]={0};

    fp1(iarr,iarr);


    return 0;
}
```

22. 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.
(프로그래밍 하시오)

```
#include <stdio.h>

int make_fibo(int size){
    int first = 1;
    int second = 1;

    if(size  == 1)
        return first;
    else if(size ==2){
        return second;
    }
    else{
        return make_fibo(size-2) + make_fibo(size -1);
    }
}

int main(void){
    int i;
    int tmp;
    int oddSum=0;
    int evenSum=0;

    for(i=1;i<28;i++){

        tmp =make_fibo(i);
        printf("%d\n",tmp);
        if(tmp%2 ==1){
            oddSum += tmp;
        }
        else{
            evenSum += tmp;
```

```
        }
    }

    printf("odd sum is %d and even sum is %d\n",oddSum,evenSum);

    return 0;
}
```

23. 1, 4, 5, 9, 14, 23, 37, 60, 97, ... 형태로 숫자가 진행된다.
23 번째 숫자는 무엇일까 ?
(프로그래밍 하시오)

```
#include <stdio.h>

int make_fibo(int size){
    int first = 1;
    int second = 4;

    if(size  == 1)
        return first;
    else if(size ==2){
        return second;
    }
    else{
        return make_fibo(size-2) + make_fibo(size -1);
    }
}

int main(void){
    int i;
    int tmp;
    int oddSum=0;
    int evenSum=0;
```

```
    for(i=1;i<24;i++){

        tmp =make_fibo(i);
        printf("%d\n",tmp);
    }


    return 0;
}
```

24. Intel Architecture 와 ARM Architecture 의 차이점은 ?

하지만 **ARM** 은 로드/스토어 아키텍처라고 해서
메모리 에서 메모리로 연산이 불가능하다**.**
(반도체 다이 사이즈가 작기 때문임**)**
다이 사이즈가 작아서 **Functional Unit** 갯수가 적음
(이런 연산을 지원해줄 장치가 적어서 안됨**)**

그래서 **ARM** 은 먼저 메모리에서 레지스터로 값을 옮기고
다시 이 레지스터 값을 메모리로 옮기는 작업을함
로드하고 스토어하는 방식이라고 해서
로드/스토어 아키텍처라고함

25. 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.

26. 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

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

int func(int data){
    int res;
    res = data*data;
    return res;
}

void sapply(int (*p)[3], int (*f)(int data)){
    for(int i=0;i<3;i++){
        p[1][i]=f(p[0][i]);
    }
}

int main(void){
    int arr[][3]={{2,4,6},{2,4,6}};
    sapply(arr,func);

    return 0;
}
```

sapply 함수를 위와 같이 구현하라는 의미다.
(R 이나 python 같은 언어에서 지원되는 기법중 하나에 해당한다)

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

28. int *p[3] 와 int (*p)[3] 는 같은 것일까 ? 다른 것일까 ?
이유를 함께 기술하도록 한다.
전혀 다르다.
Int *p[3] 은 인트형 포인터가 **3** 개가 나란히 있는 형태이다.
Int (*p)[3] 은 포인터 한개이며 해당 포인터가 **12** 바이트씩 건너 뛸 수 있다. **(**즉 배열이 얼만큼의 크기를 가지고 있는지 알고있는 포인터**)**

29. 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)

입력된 값 **& (~134217727)** 으로 연산을 한다.

30. 단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.
(프로그래밍 하시오), 덧셈 혹은 뺄셈 같은 기능이 아님

31. 변수의 정의를 기술하시오.
메모리 공간에 주소값을 가지고 올라가는 **data.**
즉 값의 변화가 가능하다.

32. 포인터의 정의를 기술하시오.
변수를 가리키는 주소를 저장하는 변수.

33. 함수 포인터의 정의를 기술하시오.
함수를 가리키는 주소를 저장하는 변수.

34. 재귀호출을 사용하여 queue 를 구현하고 10, 20 을 집어넣는다.
enqueue 과정에 대한 기계어 분석을 수행하여 동작을 파악하도록 한다.
그림과 함께 자세하게 설명하시오.

35. 파이프라인은 언제 깨지는가 ?
분기 명령이 떨어지면 깨지게 된다.

36. 메모리 계층 구조에 대해 기술하시오.
레지스터 캐시 메모리 디스크 가 있다.
속도는 레지스터 캐시 메모리 디스크 순이며 용량은 디스크 메모리 캐시 레지스터 순이다.
디스크에서 값을 불러와 메모리에 임시 저장하며 메모리의 값을 불러와서 캐시에 담아 버퍼처럼 사용하고 레지스터에서 값을 불러와서 연산을 하게 된다.

37. C 언어의 기본 메모리 구조에 대해 기술하시오.
스택/힙/데이타/텍스트 메모리 구조이다.
스택은 지역 변수가 위치한다.
힙은 동적 할당되어 사용자가 원하는데로 사용이 가능하다.
데이타 영역은 전역 변수 및 스태틱으로 선언된 것들이 위치한다.
텍스트 영역은 머신 코드가 위치하는 영역이다.

38. 우리가 사용하는 윈도우나 리눅스, 맥(유닉스)에서 보는
변수 주소는 진짜 주소일까 아닐까 ?
알고 있는대로 기술하시오.

39. 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)

40. 리눅스에서 디버깅을 수행하기 위한 프로그램 컴파일 방법을 기술하시오.
Gcc -g +파일명
으로 실행파일 생성 후 **gdb** + 실행파일 과 같은 구조의 명령어로 디버깅이 가능하다.
만일 최적화 모드를 풀고 싶다면 **-O0** 과 같이 최적화 레벨을 선택하여 **gcc** 명령어에 넣어주면 된다.

41. 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 <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct __stack{
    int data;
    struct __stack *link;
}stack;

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

int pop(stack **top)
{
    stack *tmp = *top;
    int data = 0;

    if(*top == NULL)
    {
        printf("stack is empty!\n");
        return 0;
    }

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

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

```
    return data;
}

void push(stack **top, int data)
{
    if(data == 0)
        return;

    stack *tmp = *top;
    *top = get_stack_node();
    (*top)->data = data;
    (*top)->link = tmp;
}


int main(void)
{
        int i, size;
        int data[22] = {0};
        stack *top =NULL;
        avl * root = NULL;
        rb_tree *rbt = NULL;
        rb_node *find = NULL;

        srand(time(NULL));

        size = sizeof(data) / sizeof(int) - 1;

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

        for(i=0;i<size;i++)
           push(&top,data[i]);
```

```
        return 0;
}
```

42. 41 번에서 홀수만 빼내서 AVL 트리를 구성하도록 한다.

43 번에 한꺼번에 풀이**.**

43. 41 번에서 짝수만 빼내서 RB 트리를 구성하도록 한다.

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

#define         BLACK       0
#define         RED         1

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

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

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;



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));
   }
}

typedef struct __rb_node
{
        int data;
        int color;
```

```
	struct __rb_node *left;
	struct __rb_node *right;
	struct __rb_node *parent;
} rb_node;

typedef struct __rb_tree
{
	struct __rb_node *root;
	struct __rb_node *nil;
} rb_tree;

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() % 100 + 1;

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

```
                        goto redo;
                }
        }
}

void rb_left_rotate(rb_tree **tree, rb_node *x)
{
        rb_node *y;
        rb_node *nil = (*tree)->nil;

        y = x->right;
        x->right = y->left;

        if(y->left != nil)
                y->left->parent = x;

        y->parent = x->parent;

        if(x == x->parent->left)
                x->parent->left = y;
        else
                x->parent->right = y;

        y->left = x;
        x->parent = y;
}

void rb_right_rotate(rb_tree **tree, rb_node *y)
{
        rb_node *x;
        rb_node *nil = (*tree)->nil;

        x = y->left;
        y->left = x->right;
```

```
	if(nil != x->right)
		x->right->parent = y;

	x->parent = y->parent;

	if(y->parent->left == y)
		y->parent->left = x;
	else
		y->parent->right = x;

	x->right = y;
	y->parent = x;
}

void rb_tree_ins_helper(rb_tree **tree, rb_node *z)
{
	rb_node *x;
	rb_node *y;
	rb_node *nil = (*tree)->nil;

	z->left = z->right = nil;
	y = (*tree)->root;
	x = (*tree)->root->left;

	while(x != nil)
	{
		y = x;

		if(x->data > z->data)
			x = x->left;
		else
			x = x->right;
	}
```

```
        z->parent = y;

        if(((*tree)->root == y) || (y->data > z->data))
                y->left = z;
        else
                y->right = z;
}

rb_node *rb_tree_ins(rb_tree **tree, int data)
{
        rb_node *x;
        rb_node *y;
        rb_node *tmp;

        x = (rb_node *)malloc(sizeof(rb_node));
        x->data = data;

        rb_tree_ins_helper(tree, x);

        tmp = x;
        x->color = RED;

        while(x->parent->color)
        {
                if(x->parent == x->parent->parent->left)
                {
                        y = x->parent->parent->right;

                        if(y->color)
                        {
                                x->parent->color = BLACK;
                                y->color = BLACK;
                                x->parent->parent->color = RED;
```

```
                x = x->parent->parent;
        }
        else
        {
                if(x->parent->right == x)
                {
                        x = x->parent;
                        rb_left_rotate(tree, x);
                }

                x->parent->color = BLACK;
                x->parent->parent->color = RED;

                rb_right_rotate(tree, x->parent->parent);
        }
}
else
{
        y = x->parent->parent->left;

        if(y->color)
        {
                x->parent->color = BLACK;
                y->color = BLACK;
                x->parent->parent->color = RED;
                x = x->parent->parent;
        }
        else
        {
                if(x->parent->left == x)
                {
                        x = x->parent;
                        rb_right_rotate(tree, x);
                }
```

```
                        x->parent->color = BLACK;
                        x->parent->parent->color = RED;

                        rb_left_rotate(tree, x->parent->parent);
                    }
                }
        }

        (*tree)->root->left->color = BLACK;

        return tmp;
}

rb_tree *rb_tree_create(void)
{
        rb_tree *rbt;
        rb_node *tmp;

        rbt = (rb_tree *)malloc(sizeof(rb_tree));

        tmp = rbt->nil = (rb_node *)malloc(sizeof(rb_node));
        tmp->parent = tmp->left = tmp->right = tmp;
        tmp->color = BLACK;
        tmp->data = 0;

        tmp = rbt->root = (rb_node *)malloc(sizeof(rb_node));
        tmp->parent = tmp->left = tmp->right = rbt->nil;
        tmp->color = BLACK;
        tmp->data = 0;

        return rbt;
}
```

```
void rb_tree_preorder_print(rb_tree *tree, rb_node *x)
{
	rb_node *nil = tree->nil;
	rb_node *root = tree->root;

	if(x != tree->nil)
	{
		printf("data = %4i, ", x->data);

		if(x->left == nil)
			printf("left = NULL, ");
		else
			printf("left = %4i, ", x->left->data);

		if(x->right == nil)
			printf("right = NULL, ");
		else
			printf("right = %4i, ", x->right->data);

		printf("color = %4i\n", x->color);

		rb_tree_preorder_print(tree, x->left);
		rb_tree_preorder_print(tree, x->right);
	}
}

void rb_tree_print(rb_tree *tree)
{
	rb_tree_preorder_print(tree, tree->root->left);
}

int data_test(int n1, int n2)
{
	if(n1 > n2)
```

```
                return 1;
        else if(n1 < n2)
                return -1;
        else
                return 0;
}


rb_node *rb_tree_successor(rb_tree *tree, rb_node *x)
{
        rb_node *y;
        rb_node *nil = tree->nil;
        rb_node *root = tree->root;

        if(nil != (y = x->right))
        {
                while(y->left != nil)
                        y = y->left;

                return y;
        }
        else
        {
                y = x->parent;

                while(y->right == x)
                {
                        x = y;
                        y = y->parent;
                }

                if(y == root)
                        return nil;
```

```
                return y;
        }
}

int pop(stack **top)
{
   stack *tmp = *top;
   int data = 0;

   if(*top == NULL)
   {
        printf("stack is empty!\n");
        return 0;
   }

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

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

void push(stack **top, int data)
{
   if(data == 0)
        return;

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

```
int main(void)
{
	int i, size;
	int data[22] = {0};
	stack *top =NULL;
	avl * root = NULL;
	rb_tree *rbt = NULL;
	rb_node *find = NULL;
	int tmp;

	srand(time(NULL));

	size = sizeof(data) / sizeof(int) - 1;

	init_rand_arr(data, size);
	rbt = rb_tree_create();

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

	for(i=0;i<size;i++)
	   push(&top,data[i]);

	for(i=0;i<size;i++){
	   tmp = pop(&top);
	   if(tmp %2 == 1){
		avl_ins(&root,tmp);
	   }
	   else{
		rb_tree_ins(&rbt, tmp);
	   }
	}
```

```
        rbt = rb_tree_create();
/*
        for(i = 0; i < size; i++)
                rb_tree_ins(&rbt, data[i]);

        rb_tree_print(rbt);


*/
        return 0;
}
```

44. vi 에서 코드가 정렬이 잘 안되어 있다.
이 상황에서 어떻게 해야 예쁘고 깔끔하게 정렬되는가 ?
Visual 모드로 들어가서 =을 하면 깔끔하게 정리된다**.**

45. 프로그램을 최적화하는 컴파일 옵션을 적고
반대로 어떠한 최적화도 수행하지 않는 컴파일 옵션을 적도록 한다.
gcc -g -O0 으로 최적화 없이 수행**/ gcc** 를 그냥 쓰면 프로그램 최적화 컴파일

46. 최적화 프로세스를 기술하도록 한다.
만일 **(~1028)** 이라고 할 때**,** 최적화가 없으면 비트를 반전 시키는 연산을 하고 대입을 하는데
최적화 기술로 반전 된 값을 바로 어셈블리어에 넣는다**.**

47. 기존에는 자료구조에서 숫자값만을 받았다.
이제 Queue 에서 데이터로서 숫자가 아닌 문자열을 받아보자.

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

```
typedef char* DATA;

typedef struct __queue
{
   DATA data;
   struct __queue *link;
} queue;


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

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

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

void print_queue(queue *head)
{
        queue *tmp = head;
```

```
        while(head)
        {
                printf("head->data = %s\n", head->data);
                head = head->link;
        }
}


int main(void)
{
        int i;

        queue *head = NULL;

        enqueue(&head,"hello");
        enqueue(&head,"dudu");
        enqueue(&head,"what's");
        enqueue(&head,"up!?");

        print_queue(head);

        return 0;
}
```

48. **Binary Tree 에서 위 작업을 수행해보라.**

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

typedef char* DATA;

typedef struct __tree
{
        DATA 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, DATA data)
{
        if(*root == NULL)
        {
                *root = get_node();
                (*root)->data = data;
                return;
        }
        else if(strlen((*root)->data) > strlen(data))
                tree_ins(&(*root)->left, data);
        else if(strlen((*root)->data) < strlen(data))
                tree_ins(&(*root)->right, data);
}



void print_tree(tree *root)
{
        if(root)
        {
```

```
                printf("data = %s, ", root->data);
                print_tree(root->left);
                print_tree(root->right);
        }
}



int main(void)
{
        int i;
        tree *root = NULL;

        tree_ins(&root, "hello");
        tree_ins(&root, "this is my world!");
        tree_ins(&root, "do you...");
        tree_ins(&root, "know???");

        print_tree(root);

        return 0;

}
```

49. AVL Tree 에서 위 작업을 수행해보라.

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

typedef char* string;

typedef enum __rot
```

```
{
        RR,
        RL,
        LL,
        LR
} rot;

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


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 = %s, lev = %d, ", root->data, root->lev);
                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, string data)
{

	// for RR and RL
	if(rotation_check(root) > 1)
	{
		if(rotation_check(root->right) > 0)
			return RR;

		return RL;
	}
	// for LL and LR
	else if(rotation_check(root) < -1)
```

```
        {
                if(rotation_check(root->left) < 0)
                        return LL;

                return LR;
        }
}

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, int data)
{
        child = ll_rot(child, child->left);

        return rr_rot(parent, child);
```

```
}


avl *lr_rot(avl *parent, avl *child, int data)
{
        child = rr_rot(child, child->right);

        return ll_rot(parent, child);
}


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, string data)
{
        if(!(*root))
```

```
        {
                (*root) = get_avl_node();
                (*root)->data = data;
                return;
        }

        if(strlen((*root)->data) > strlen(data))
                avl_ins(&(*root)->left, data);
        else if(strlen((*root)->data) < strlen(data))
                avl_ins(&(*root)->right, data);


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

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

        }
}


int main(void)
{
        int i;
        avl *test = NULL;

        avl_ins(&test, "hello");
        avl_ins(&test, "fofo");
        avl_ins(&test, "lalaoih");
        avl_ins(&test, "kokokhdfg");

        print_tree(test);
```

```
        return 0;
}
```

50. 성적 관리 프로그램을 만들어보자.

1. 통계 기능(총 합산, 평균, 표준 편차 계산)
2. 성적순 정렬 기능
3. 성적 입력 기능
4. 학생 정보 삭제 기능

51. gdb 를 사용하는 이유를 기술하라.

디버깅을 하기 위해서 사용한다.

52. 기계어 레벨에서 스택을 조정하는 명령어는 어떤것들이 있는가 ?

PUSH, POP RETQ CALL 등등이 있다.

53. a 좌표(3, 6), b 좌표(4, 4) 가 주어졌다.

원점으로부터 이 두 좌표가 만들어내는 삼각형의 크기를 구하는 프로그램을 작성하라.

54. 가위 바위 보 게임을 만들어보자.

프로그램을 만들고 컴퓨터랑 배틀 한다.

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

int main(void){
   int random;
   int user;

   srand(time(NULL));

   while(1){
        random = rand()%3+1;
        printf("가위 1 바위 2 보 3\n");
```

```
scanf("%d",&user);

if(user == 1){
   if(random == 1){
        printf("draw!\n");
   }
   else if(random ==2){
        printf("Lose!\n");
   }
   else if(random ==3){
        printf("Win!\n");
   }
}
else if(user ==2){

   if(random == 1){
        printf("Win!\n");
   }
   else if(random ==2){
        printf("draw!\n");
   }
   else if(random ==3){
        printf("Lose!\n");
   }

}
else if(user ==3){


   if(random == 1){
        printf("Lose!\n");
   }
   else if(random ==2){
        printf("Win!\n");
```

```
            }
            else if(random ==3){
                printf("draw!\n");
            }
        }
    }


    return 0;
}
```

55. 화면 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 출력하게 될 값들을 800, 600 화면에 가득차게 표현할 수 있는
    스케일링 값을 산출하는 프로그램을 작성하도록 한다.

56. 등차 수열의 합을 구하는 프로그램을 for 문을 도는 방식으로 구현하고
    등차 수열의 합 공식을 활용하여 구현해본다.
    함수 포인터로 각각의 실행 결과를 출력하고
    이 둘의 결과가 같은지 여부를 파악하는 프로그램을 작성하라.

57. sin(x) 값을 프로그램으로 구현해보도록 한다.
    어떤 radian 값을 넣던지 그에 적절한 결과를 산출할 수 있도록 프로그래밍 한다.