TI DSP, MCU 및 Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – 문한나

mhn97@naver.com

사전평가

1. 단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.
   C 언어로 프로그램을 구현하시오.

```c
#include <stdio.h>

int main(void){

        char a;

        printf("문자를 입력하세요");
        scanf("%c",&a);

        if(a >= 'A' && a <= 'Z'){
                printf("%c",a+32);
        }else if (a >= 'a' && a <= 'z'){
                printf("%c",a-32);
        }else printf("다시 입력하세요");

        return 0;

}
~
~
```

2. Stack 및 Queue 외에 Tree 라는 자료구조가 있다.
   이 중에서 Tree 는 Stack 이나 Queue 와는 다르게 어떠한 이점이 있는가?

   데이터 탐색 및 삽입 삭제를 좀 더 효율적으로 할 수 있다.

3. 임의의 값 x 가 있는데, 이를 4096 단위로 정렬하고 싶다면 어떻게 해야할까?
   C 언어로 프로그램을 구현하시오.

4. int p[7] 와 int (*p)[7] 가 있다.
   이 둘의 차이점에 대해 기술하시오.

   int p[7] = int 형 값을 7 개 저장할 수 있는 배열 p
   int (*p)[7] = int 형 데이터타입을 저장할 수 있는 공간의 주소 7 개를 가진 포인터 배열

5. 다음을 C 언어로 구현하시오.
   이번 문제의 힌트를 제공하자면 함수 포인터와 관련된 문제다.
   아래와 같은 행렬을 생각해보자!
   1 2 3
   1 2 3
   sapply(arr, func) 이라는 함수를 만들어서 위의 행렬을 아래와 같이 바꿔보자!
   sapply 에 func 함수가 연산을 수행하는 함수로 만들어야 한다.
   1 2 3
   1 4 9

6. 다음 사항에 대해 아는대로 기술하시오.
   Intel Architecture 와 ARM Architecture 의 차이점은 무엇인가?

7. 이것이 없으면 C 언어의 함수를 호출할 수 없다.
   여기서 이야기하는 이것은 무엇일까 ?

8. 아래 내용을 C 로 구현해보도록 하자.
   3x^2 + 7x 를 1 ~ 2 까지 정적분하는 프로그램을 구현해보라.
   3x^2 에서 ^2 는 제곱을 의미한다.
   예로 x 에 1 이 들어가면 3x^2 = 9 가 된다.

9. Memory Hierarchy(메모리 계층 구조)에 대해 기술하시오.

<Memory Hierarchy>

Register
Cache
Memory
Disk

위에 있을수록 빠르고, 아래로 내려갈수록 느리다
용량은 순서가 반대이다

10. C 언어에서 중요시하는 메모리 구조에 대해 기술하시오.
(힌트: Stack, Heap, Data, Text 에 대해 기술하시오.)

Stack : 지역변수
Heap : 동적할당된 메모리
Data : 정적변수와 상수
Text : 함수

11. 파이프라인이 깨지는 경우에 대해 기술하시오.

분기 명령어를 사용할 때 ex)call,jmp

12. void (* signal(int signum, void (* handler)(int)))(int)라는 signal 함수의 프로토타입을 기술하시오.
프로토타입을 기술하라는 의미는 반환형(리턴 타입)과 함수의 이름, 그리고 인자(파라메터)가 무엇인지 기술하라는 뜻임.

리턴:void (*)(int)
이름:signal
인자:(int signum, void (*handler)(int)))

13. goto 를 사용하는 이유에 대해 기술하시오.

goto 의 명령어는 jmp 하나이기 때문이다.
만약 for 문이 여러 개 생겨 if, break 조합의 경우 for 문의 갯수 만큼 mov, cmp, jmp 를 해야 한다. 때문에 goto 가 cpu 파이프라인을 덜 손상시키며, 성능면에서도 월등히 좋다.

14. 아래 질문에 적절한 C 코드를 구현하시오.
TI Cortex-R5F Safety MCU is very good to Real-Time System.
위의 문장에서 Safety MCU 가 몇 번째 부터 시작하는지 찾아내보자!
(배열의 인덱스로 표기해도 상관없고, 실제 위치로 표기해도 상관없으며
둘 중 무엇인지 표기하시오)

15. 아래 질문에 대한 C 코드를 작성하시오.
배열에 아래와 같은 정보들이 들어있다.
2400, 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
여기서 가장 빈도수가 높은 3 개의 숫자를 찾아 출력하시오!

18. 운영체제의 5 대 요소 5 가지를 적으시오.

19. Stack 자료구조를 아래와 같은 포맷에 맞춰 구현해보시오. (힌트: 이중 포인터)

ex)

```
int main(void)
{
    stack *top = NULL;
    push(&top, 1);
    push(&top, 2);
    printf("data = %d₩n", pop(&top));
}
```

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

    Stack *top = EMPTY;
    push(&top,1);
    push(&top,2);
    printf("%d₩n",pop(&top));
    return 0;

}
```

20. Binary Tree 나 AVL Tree, Red-Black Tree 와 같이 Tree 계열의 자료구조를 재귀 호출 없이 구현하고자 한다. 이 경우 반드시 필요한 것은 무엇인가 ?

21. Binary Tree 를 구현하시오.
초기 데이터를 입력 받은 이후 다음 값이 들어갈 때 작으면 왼쪽 크면 오른쪽으로 보내는 방식으로 구현하시오. 삭제 구현이 가능하다면 삭제도 구현하시오.

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

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<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,32);
    printf("after delete₩n");

    print_tree(root);


    return 0;
}
```

22. AVL 트리는 검색 속도가 빠르기로 유명하다.
Red-Black 트리도 검색 속도가 빠르지만 AVL 트리보다 느리다.
그런데 어째서 SNS 솔루션 등 에서는 AVL 트리가 아닌 Red-Black 트리를 사용할까?

AVL 트리는 검색 속도는 빠르지만 대규모로 입,출력을 하게 될 시 느려진다.
그래서 검색속도는 AVL 트리보다 조금 떨어지지만 대규모 데이터 처리에도 입,출력이 빠르게 처리되는 Red-Black 트리를 사용하게 되었다.

23. AVL 트리를 C 언어로 구현하시오.
AVL 트리는 밸런스 트리로 데이터가 1, 2, 3, 4, 5, 6, 7, 8, 9 와 같이
순서대로 쌓이는 것을 방지하기 위해 만들어진 자료구조다.

```
#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;
}
```

85. 1 ~ 100 까지의 숫자중 홀수만 더해서 출력해보시오.

```
#include <stdio.h>

int main(){

    int i;
    int res=0;

    for(i=1;i<=100;i++){
        if(i % 2 != 0){
            res = res+i;
        }
    }

    printf("1~100까지 홀수의 합 : %d",res);
    return 0;
}
```

86. 1 ~ 100 까지 숫자를 모두 더해서 첫 번째 배열에 저장하고
1 ~ 100 까지 숫자중 홀수만 더해서 두 번째 배열에 저장하고
1 ~ 100 까지 숫자중 짝수만 더해서 세 번째 배열에 저장한다.
다음으로 1 ~ 100 까지 숫자중 3 의 배수만 더해서 네 번째 배열에 저장한다.
각 배열의 원소를 모두 더해서 결과값을 출력하시오.

```
#include <stdio.h>

int main(void){

	int arr[4]={0};
	int i;

	for(i=1;i<=100;i++){

		arr[0] = arr[0]+i;

	}

	for(i=1;i<=100;i++){
		if(i%2 == 0){
			arr[2] = arr[2]+i;
		}else
			arr[1] = arr[1]+i;
	}

	for(i=1;i<=100;i++){
		if(i%3 == 0){
			arr[3] = arr[3]+i;
		}
	}

	printf("%d₩n",arr[0]);
	printf("%d₩n",arr[1]);
	printf("%d₩n",arr[2]);
	printf("%d",arr[3]);
	return 0;

}
```