

# TI DSP, MCU 및 Xilinx Zynq FPGA 프로그램 전문가 과정

강사 - Innova Lee(이상훈)

[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)

학생 - 은태영

[zero\\_bird@naver.com](mailto:zero_bird@naver.com)

# 사전평가

단 한번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.

아스키 코드 상 대문자 'A(65)' 와 소문자 'a(97)' 는 32의 차이를 갖고 있다. 그렇기 때문에 대소문자를 변환하고자 할 경우, 소문자일 경우 -32 를 대문자일 경우 +32 를 더해주면 된다. 이때, 주의할 점은 64 이하의 숫자나, 123 이상의 숫자는 특수문자이기 때문에 if 문을 통하여 예외처리를 해 주는 것이 좋다.

# 사전평가

stack 및 queue 외에 tree 라는 자료구조가 있다.  
이 중에서 tree 는 stack 이나 queue 와는 다르게  
어떠한 이점이 있는가?

stack 이나 queue의 경우 시작점을 잡은 후, 각  
node가 갖고있는 link를 통하여 이동을 한다.  
이 때, node의 수가 많아질수록 원하고자 하는  
데이터를 찾는데 많은 시간이 걸리게 된다.  
그에 비해 tree의 경우 데이터의 내용에 따라  
구분하여 링크를 연결하기 때문에, 데이터 탐색 시  
보다 빠르게 원하는 자료를 찾을 수 있다.

# 사전평가

임의의 값  $x$ 가 있는데 이를 4096 단위로 정렬하고 싶다면 어떻게 해야 할까?

$\%$  와  $/$  를 이용하여 정렬할 수 있다.

$x/4096$  을 통해 몇 개의 값을 갖고 있는지 알 수 있으며,  $x\%4096$  을 통해 나머지 값을 알 수 있다.  
이를 이용하여 정렬이 가능하다.

# 사전평가

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

`int p[7]` 의 경우 `int` 형으로 이루어진 7개의 공간, 즉 `int` 배열을 나타내는 말이다.

`int (*p)[7]` 의 경우, 이를 보기 쉽게 풀어서 쓰면 ‘`int *[7] p`’ 라는 뜻으로, `int` 의 7개 공간에 대한 포인터라는 뜻이다. 즉, `int` 1개가 4byte 이기 때문에 7개 는 28byte를 나타내며, 해당 크기 만큼에 대한 포인터라는 뜻이다.

# 사전평가

다음은 c 언어로 구현하시오.

이번 문제의 힌트를 제공하자면 함수 포인터와 관련된 문제이다. 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

```
tewill@tewill-B85M-D3H: ~/my_proj/lesson013
#include <stdio.h>

void sapply(int (*arr)[3], int func)
{
    int i, j;
    for(i = 0; i < 3; i++)
        for(j = 0; j < func-1; j++)
            arr[1][i] = (arr[1][i]) * (arr[0][i]);
}

void print_num(int (*num)[3]){
    int i, j;

    for(i = 0; i < 2; i++){
        for(j = 0; j < 3; j++){
            printf("%d\t", num[i][j]);
        }
        printf("\n");
    }
}

int main(void)
{
    int num[2][3] = {{1, 2, 3}, {1, 2, 3}};

    print_num(num);

    sapply(num, 2);

    print_num(num);

    return 0;
}
"a1.c" 35L, 454C 1,2 All
```

# 사전평가

다음 사항에 대해 아는 대로 기술하시오.  
intel architecture 와 arm architecture 의  
차이점은 무엇인가?

stack 이 쌓이는 방향이 다르다.  
arm 의 경우 항상 아래로 쌓이지만, intel의 경우  
종류에 따라 반대가 되기도 한다.

# 사전평가

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

stdio.h 에 대한 이야기이다. c 언어에 대한 기본적인 라이브러리를 포함한 헤더 파일이기 때문에, 선언하지 않을 경우 c 언어의 함수를 호출할 수 없다.

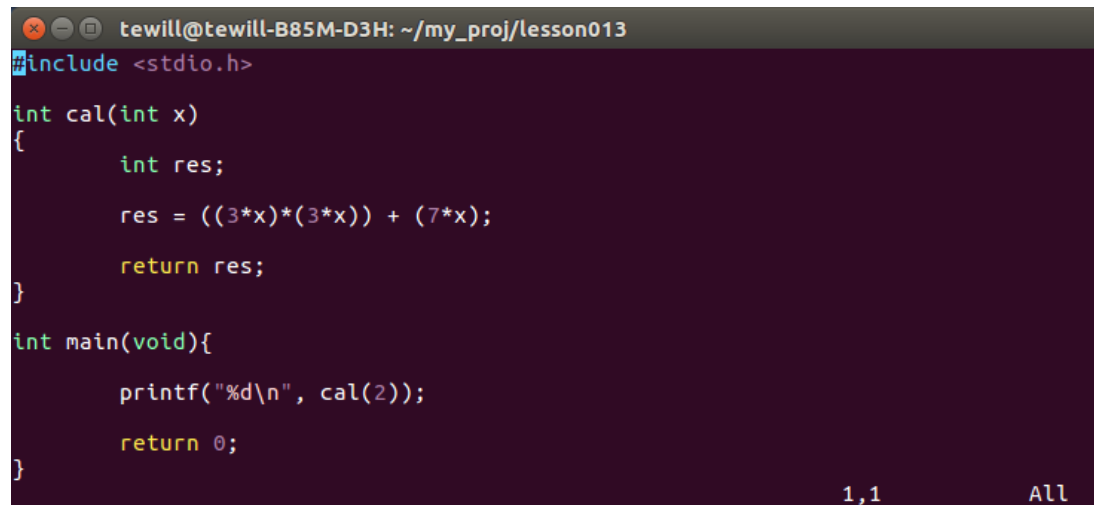


# 사전평가

아래 내용을 c 로 구현해보도록 하자.

$3x^2 + 7x$  를 1 ~ 2 까지 정적분 하는 프로그램을 구현해보라.  $3x^2$ 에서  $^2$ 는 제곱을 의미한다.

예로  $x$  에 1이 들어가면  $3x^2 = 9$  가 된다.



```
tewill@tewill-B85M-D3H: ~/my_proj/lesson013
#include <stdio.h>

int cal(int x)
{
    int res;

    res = ((3*x)*(3*x)) + (7*x);

    return res;
}

int main(void){
    printf("%d\n", cal(2));

    return 0;
}
```

1,1 All

# 사전평가

다음 사항에 대해 기술하십시오.  
memory hierarchy(메모리 계층 구조)에 대해  
기술하십시오.

메모리 계층 구조는 레지스터, 캐시, 메모리,  
하드 디스크로 구성된 것을 말한다.  
레지스터와 캐시는 cpu 내부에 위치하여 빠르고,  
메모리와 하드 디스크는 외부에 위치해 더 느리게  
접근한다. 레지스터나 캐시의 경우 비싸기 때문에,  
필요한 만큼만 사용하기에 계층 구조가 피라미드  
모양으로 구성된다.

# 사전평가

다음 질문에 대해 기술하십시오.

c 언어에서 중요시하는 메모리 구조에 대해  
기술하십시오. (힌트 : stack, heap, data, text 에  
대해 기술하십시오.)

# 사전평가

다음 사항에 대해 답하십시오.

파이프라인이 깨지는 경우에 대해 기술하십시오.

파이프 라인이 깨지는 경우는 CALL 등 JMP 를 할 경우 발생한다. 발생하는 원인으로서는 모듈이 동시에 작업을 하기 때문에, JMP 를 실행하기 전 까지 다른 작업을 진행하고 있지만, JMP 를 실행하게 되면 미리 준비했던 작업들이 쓸모 없어지고, PUSH 부터 다시 시작되기 때문이다.

이렇게 손실되는 것을 파이프 라인의 손실이라 한다.

# 사전평가

아래 질문에 대해 답하십시오.

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

`(int)` 라는 `signal` 함수의 프로토타입을 기술하십시오.

프로토 타입을 기술하라는 의미는 반환형(리턴 타입)

과 함수의 이름, 그리고 인자(파라미터)가 무엇인지

기술하라는 뜻임.

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

반환형 : `void (*) (int)`

함수 이름 : `signal`

인자 : `int signum, void(* handler(int))`

# 사전평가

다음 질문에 대해 답하십시오.

goto 를 사용하는 이유에 대해 기술하십시오.

반복문을 많이 사용한 상태에서 모든 반복문을  
나가고자 할 경우, 반복문의 숫자만큼 break 를 해야  
한다. 이는 break 하는 만큼 JMP 로 인해 파이프  
라인이 손실된다는 소리인데, goto 를 사용할 경우  
한번의 호출로 이를 해결할 수 있다.

# 사전평가

아래 질문에 대한 c코드를 작성하시오.  
배열에 아래와 같은 정보들이 들어있다.  
여기서 가장 빈도수가 높은 3개의 숫자를 찾아  
출력하시오!

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

# 사전평가

아래 질문에 대한 c코드를 작성하시오.

배열에 아래와 같은 정보들이 들어있다.

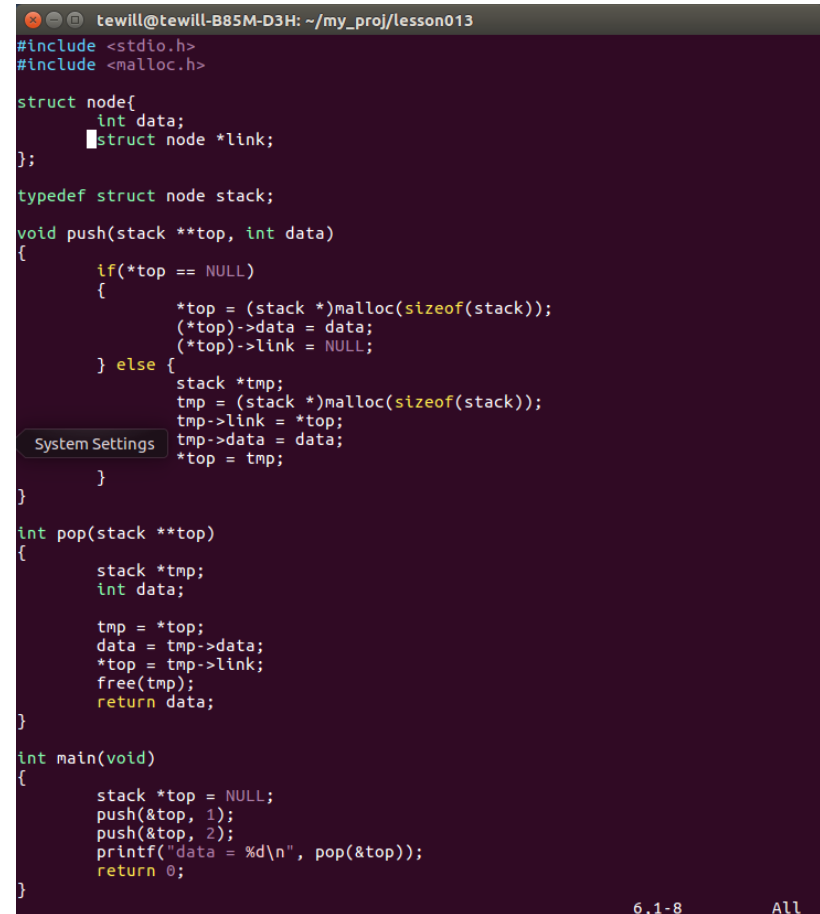
여기서 가장 빈도수가 높은 3개의 숫자를 찾아  
출력하시오!



# 사전평가

아래 자료 구조를 c 언어로 구현해 보시오.  
Stack 자료구조를 아래와 같은 포맷에 맞춰  
구현해 보시오. (힌트: 이중 포인터)  
ex)

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



```
tewill@tewill-B85M-D3H: ~/my_proj/lesson013
#include <stdio.h>
#include <malloc.h>

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

typedef struct node stack;

void push(stack **top, int data)
{
    if(*top == NULL)
    {
        *top = (stack *)malloc(sizeof(stack));
        (*top)->data = data;
        (*top)->link = NULL;
    } else {
        stack *tmp;
        tmp = (stack *)malloc(sizeof(stack));
        tmp->link = *top;
        tmp->data = data;
        *top = tmp;
    }
}

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

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

int main(void)
{
    stack *top = NULL;
    push(&top, 1);
    push(&top, 2);
    printf("data = %d\n", pop(&top));
    return 0;
}
```

6,1-8 All

# 사전평가

다음 질문에 대해 답하십시오.

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

반복문을 통하여, 재귀함수를 통해 진행되던 것을 함수 내부에서 진행한다.

# 사전평가

다음 자료 구조를 c 로 구현하시오.

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.

# 사전평가

다음 질문에 대해 기술하십시오.

AVL 트리는 검색 속도가 빠르기로 유명하다.

Red-Black 트리도 검색 속도가 빠르지만

AVL 트리보다 느리다.

그런데 어째서 SNS 솔루션 등에서는 AVL 트리가

아닌 Red-Black 트리를 사용할까 ?

AVL 트리의 경우 검색 속도는 가장 빠르지만,

데이터의 입력 및 삭제에 있어서 Red-Black 트리에

비하여 정렬을 자주하게 된다. 즉, 입력 및 삭제 시

속력이 Red-Black 에 비해 느리다.

SNS 솔루션 등의 경우 데이터의 추가 및 삭제가

많기 때문에 해당 기능에서 느린 AVL 이 아닌

Red-Black 트리를 사용한다.

# 사전평가

다음 자료구조를 c 로 구현하시오.

AVL 트리를 C 언어로 구현하시오.

AVL 트리는 밸런스 트리 데이터가

1, 2, 3, 4, 5, 6, 7, 8, 9 와 같이 순서대로 쌓이는  
것을 방지하기 위해 만들어진 자료구조다.

# 사전평가

제외된 문제는 풀지 못했습니다.

