Xilinx Zynq FPGA,TI DSP, MCU 기반의 프로그래밍 전문가 과정

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

학생 – 정한별
hanbulkr@gmail.com

함수 포인터 (/hestit/1858)

float (* (* test(void (*p)(void)))(float (*)(int, int)))(int, int)
위와 같은 프로토타입의 함수가 구동되도록 프로그래밍 하시오.

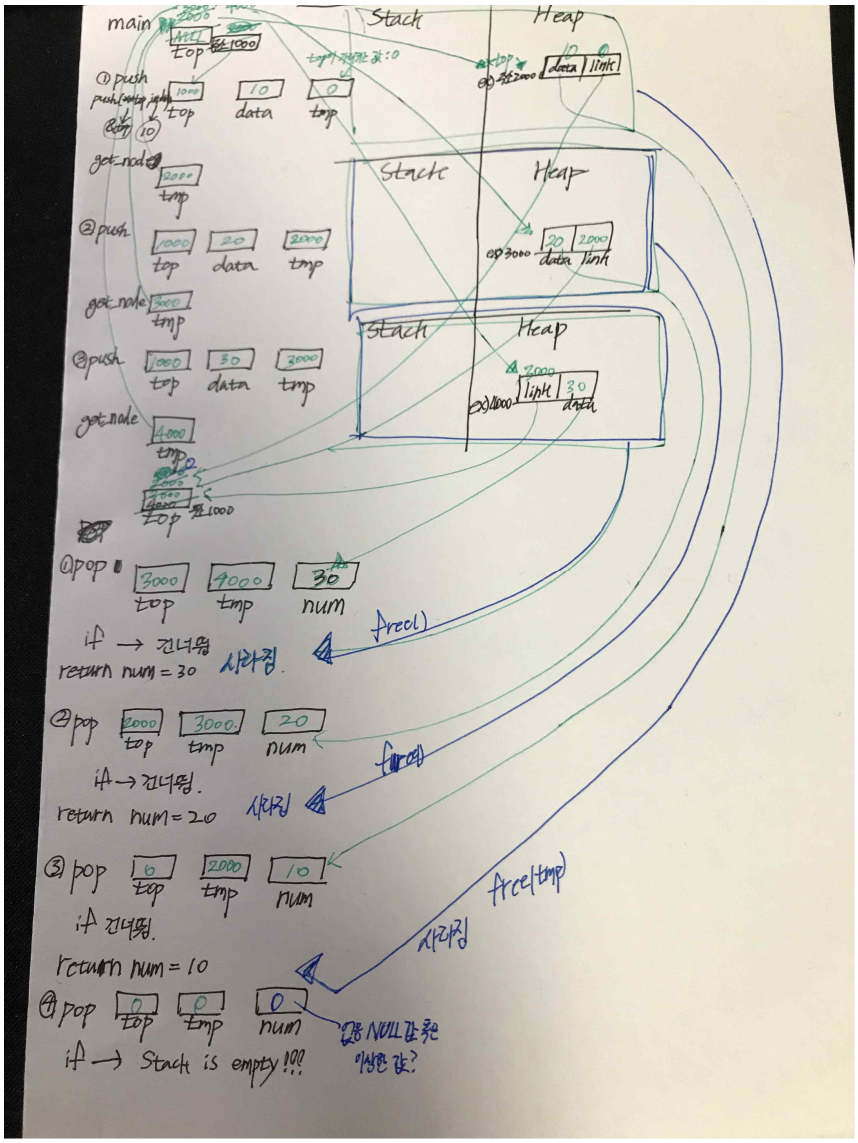
```
#include <stdio.h>

//float(*(*test(void(*)(void)))(float (*)(int, int)))(int,int);

//float test(void(*)(void)) (float (*)(int, int))
//1.  float (*)(int,int) (*)( ) test(void(*)(void)) (float (*)(int, int)

//float ( *test(void(*)(void)) )( float (*)(int,(int)) )
//2.  float (*)(int, int) (*)(float (*)(int, int)) test(void(*)(void))

float pof_test1(int n1, int n2)
{
        return n1+n2*2.7;
}
float pof_test2(int n1, int n2)
{
        return n1 + n2 +3.3 ;
}

void *message(void)
{
        printf("이벤트가 발생하여 실행합니다.");
}


//float(*(*test(void(*)(void)))(float (*)(int, int)))(int,int)
//float (*)(int, int) (*)(float (*)(int, int)) test(void(*)(void))

// float (*(*)(float (*)(int, int)))(int,int)
// float (*)(int, int) (*)(float (*)(int,int))
float (*(*test(void (*p3)(void)))(float (*p2)(int, int)))(int,int)
{
        float res;
        p3();
        res =  p2(5,5);
        printf("res = %f",res);
        return 0;
}

//float (*)(int, int) (*)(float (*)(int, int)) test(void(*)(void));
int main (void)
{
        float res=0;

        test()(5,5)(4,4);
        printf("test =%f", res);

        return 0;
}
```

Stack(push , pop)

```
#include<stdio.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, 10);
    push(&top, 20);
    push( &top, 30);
    printf("%d₩n", pop(&top));
    printf("%d₩n", pop(&top));
    printf("%d₩n", pop(&top));
    printf("%d₩n", pop(&top));
    return 0;
}
```

위의 코드를 분석한 그림



<6일차>

*삼항 연산자

- **(조건 ? 참 : 거짓) 의 형태로 쓸수 있다. ex) num>5 ? return 0 : return 1**

*헤더 파일의 추가 –수업중 컴파일시 자동으로 찾아주는 명령어...

*rand 함수

- 헤더파일 <time.h>, <stdlib.h>
- 1. srand(time(NULL))
- 2. rand()

*typedef

- typedef int INT[5]; ------> int (*p)[2] 랑 비슷한 문법.
- int i;
- INT arr;

이런 식으로 바꾸어서 쓸 수 있다.

*malloc() 함수

- memory 구조상 heap에 데이터를 할당함.
- data가 계속 들어올 경우.
- 얼마 만큼의 data가 들어오는지 알 수 없음.
- 들어올 때만다 동적으로 할당할 필요 있음.
- *마지막에 free()함수를 이용해 동적할당을 해제해야 메모리가 원활하게 돌아간다.]

*free() 함수

- free()는 memory 구조상 heap에 data를 할당 해제함.
- malloc()의 반대 역할을 수행함.

*calloc() 함수

- int *num_ptr = (int*)calloc(2, sizeof(int)); 동적 할당을 int형 크기로(4바이트) 2개 받는다.
- int *num_ptr = (int*)malloc(sizeof(int)*2); 동적 할당을 int형 크기로 (4바이트) 2개 받는다.

*Understanding struct
구조체를 왜 사용하는가?

- 자료를 표현할 때 하나로 묶어서 표현.
- 문자열과 숫자열 한번에 묶어서 관리하고 싶을 때 등. (ex. 전화번호부)
- 구조체 안에 포인터에 접근하고 싶을 때, '->'를 쓴다.
- 구조체 안에 접근시에는 '.'를 사용한다
-
- typedef struct __id_card{ }id_card; -->여기서 '__'건드리지 말라는 뜻 (중요하다는 의미를 가지고 있다. 개발자 사이에서는 암묵적인 룰)

*구조체 (커스텀 데이터 타입)

```
struct pos{
        double x_pos;
        double y_pos;

};

int main(void)
{
        double num;
        struct pos position;

        num = 1.2;

        position.x_pos = 3.3;
        position.y_pos = 7.7;

        printf("sizeof(position)= %lu₩n",sizeof(position));
        printf("%lf₩n",position.x_pos);
        printf("%lf₩n",position.y_pos);

        return 0;
}
```

*구조체 안에 구조체 넣기

```
#include<stdio.h>

typedef struct __id_card{
        char name[30];
        char id [15];
        unsigned int age;


}id_card;

typedef struct __city{
        id_card card;
        char city[30];

}city;


int main(void){
        int i;
        city info={{"Marth Kim ", "800903-1012589",34},"seoul"};

        printf("city = %s, name = %s, id = %s, age = %d₩n",info.city,
info.card.name,
                                        info.card.id, info.card.age);

        return 0;
}
```

*구조체 배열과 포인터

```
#include<stdio.h>
#include<stdlib.h>
typedef struct __id_card{
        char *name;
        char *id;
        unsigned int age;


}id_card;

typedef struct __city{
        id_card *card;
        char city[30];

}city;


int main(void){
        int i;
        city info={NULL,"seoul"};
        info.card = (id_card*)malloc(sizeof(id_card));

        info.card -> name = "MArth Kim";
        info.card -> id = "800903-1012589";
        info.card -> age = 33;

        printf("city = %s, name = %s, id = %s, age = %d₩n",
                                        info.city,    info.card->name,    info.card->id,
info.card->age);
        free(info.card);

        return 0;
}
```

*구조체 참조 연산자

```
#include<stdio.h>

typedef struct __data{

                int val;
                struct __data *data_ref;
}data;


int main(void)
{
                int i;
                data *data_p;

                data d1 = {3,NULL};
                data d2 = {7,NULL};
```

```
                d1.data_ref = &d2;
                d2.data_ref = &d1;

                data_p = &d1;
                for(i = 1;i<=10;i++)
                {
                        printf("%3d", data_p->val);
                        (data_p->val)++;

                        data_p = data_p->data_ref;
                        if(!(i%2))
                                        printf("₩t");
                }
                printf("₩n");
                return 0;
}
```

*enum(열거형)의 유용성(네트워크 프로그래밍에 사용 함)

```
#include<stdio.h>


typedef enum __packet{

        ATTACK,
        DEFENCE,
        HOLD,
        STOP,
        SKILL,
        REBIRTH,
        DEATH =44,
        KILL,
        ASSIST

}packet;



int main(void)
{
        packet packet;


        for(packet = ATTACK; packet <= REBIRTH; packet++)
                        printf("enum num = %d₩n", packet);
        for(packet = DEATH; packet <= ASSIST; packet ++)
                        printf("enum num = %d ₩n",packet);

        return 0;

}
```

*함수 포인터 기본기
void (* signal(int signum, void (* handler)(int)))(int);
리눅스 System Call 인 signal을 제대로 이해하기 위한 기본죽에 기본

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

* 함수 프로토타입이란 ?

리턴, 함수명, 인자에 대한 기술서
그렇다면 위 함수에 대한 프로토타입은 뭘까 ?

이전에 배웠던 int (*p)[2]; -> int (*)[2] p

리턴: void (*)(int)
함수명: signal
인자: int signum 과 void (* handler)(int)

void (*p)(void):
void 를 리턴하고 void 를 인자로 취하는
함수의 주소값을 저장할 수 있는 변수 p

int aaa(int, int);

int (*p)(int, int);

*상용 SW에서 어떤식으로 함수 포인터를 활용하는가(Matlab 등등)

*시스템(커널) 개발 차원에서 함수 포인터의 중요성
시스템 커널에의 프로그램들을 까보면 함수 포인터들이 많이 들어있다. '별들의 전쟁'이라고 표현할 정도로 '*'가 함수 표현에 많이 들어가 있고 사용되어 있다.
1. 비동기식 처리
2. HW 개발 관점에서 interupt
3. system call (only one of the SW)

<7일차>

*함수 포인터 응용 문제 풀이
1.번 / cafe/hestit/788

```
#include <stdio.h>


// int (*)(flaoat, double, int) pof_test_main(float (*)(int, int))

float pof_test1(int n1, int n2)
{
        return (n1 + n2) *0.23573;
}

int pof_test2(float n1, double n2, int n3)
{
        return (n1 + n2+ n3)/ 3.0;
}


int (*pof_test_main(float (*p)(int, int)))(float , double , int ){
        float res;
        res =p(4,3);
        printf("res = %f₩n",res);

        return pof_test2;
}


int main (void)
{
        int res;
        pof_test_main(pof_test1)(3.7 , 2.4, 7);
        printf("pof_test_main res =%d", res);

        return 0;
}
```

2.번 / cafe/hestit/788

```
// int (*)(int, int) *(int) pof_test_main(flaoat (*)(int, double))
// int (*(* pof_test_main(float(*)(int,double)))(int)) (int, int)




int pof1(int n1, int n2)
{
        return n1 + n2;
}

// int (*)(int, int) subpof1(int)
int  (* subpof1(int n))(int,int)
```

```
{
        printf("n= %d₩n",n);
        return pof1;
}

// float (*)(int, double)
float pof2(int n1, double n2)
{
        return n1*n2;
}

int (*(* pof_test_main(float (*p)(int, double)))(int))(int,int)
{
        float res;
        res = p(3, 7.7);
        printf("res = %f₩n", res);
        return subpof1;

}

int main (void){
                float res;

                res=pof_test_main(pof2)(3)(7,3);
                printf("res = %f₩n",res);
                return 0;
}
```

*c언어 기타 잔당 처리(잡다한 라이브러리 함수들)

- env: 터미널에서 그냥 치면 환경변수를 볼 수 있다.
- put():
- get():
- strlen(): malloc 쓸 때, 들어온 문자열 길이를 모를 때 사용.
- strncpy(복사할 것, 복사할 곳, 복사할 개수): 글자를 카피한다.
- strncmp(비교할 것, 비교할 곳, 비교 개수): 글자를 비교한다.
- memmove(복사할 것, 복사할 곳, 복사할 크기):복사함. 메모리를 보호하지만 속도가 느림.
- memcopy(복사할 것, 복사할 곳, 복사할 크기): 복사, 속도가 빠르지만 정확성이 떨어짐.

*복잡한 함수 포인터 활용법

int (*(* bbb(void))(void))[2]; // 실제 문법
int (*)[2](*)(void) bbb(void) // 인간이 보기 편하게 만듬

void (* bbb(void))(void)
리턴: void (*)(void)
이름: bbb
인자: void

void (*)(void) bbb(void)

void ccc(void (*p)(void))
리턴: void
이름: ccc
인자: void (*p)(void)

int (* ddd(void))(void)
리턴: int (*)(void)
이름: ddd
인자: void

int (*)(void) ddd(void)

void (* bbb(void (*p)(void)))(void)
리턴: void (*)(void)
이름: bbb
인자: void (*)(void)

void (*)(void) bbb(void (*p)(void))

//
int (* aaa(void))[2]
int (*)[2] aaa(void)

배열 2 개짜리 묶음의 주소를 반환하고
인자로 void 를 취하는 함수 aaa
//

//
int (*(* bbb(void))(void))[2]
int (*)[2](*)(void) bbb(void)

배열 2 개짜리 묶음의 주소를 반환하고
인자로 void 를 취하는 함수 포인터를 반환하며
인자로 void 를 취하는 함수 bbb
//

//
int (*(*(*p[][2])(void))(void))[2]
int (*)[2] (*)(void) bbb(void)
int (*)[2] (*)(void) (*)(void) p[][2]

배열 2 개짜리 묶음의 주소를 반환하고
인자로 void 를 취하는 함수 포인터를 반환하며
다시 인자로 void 를 취하는 함수 포인터를 배열형태로 가짐
//

*함수 포인터와 인터럽트, 비동기 처리

1. 비동기 처리
2. HW 개발 관점에서 인터럽트
3. 시스템 콜(유일한 SW 인터럽트임)

여기서 인터럽트들(SW, HW)은
사실상 모두 비동기 동작에 해당한다.
결국 1 번(비동기 처리)가 핵심이라는 의미다.

그렇다면 비동기 처리라는 것은 무엇일까 ?

기본적으로 동기 처리라는 것은 송신하는 쪽과 수신하는 쪽이 쌍방 합의하에만 달성된다.
(휴대폰 전화 통화 등등)
반면 비동기 처리는 이메일, 카톡등의 메신저에 해당한다.
그래서 그냥 일단 던져 놓으면 상대방이 바쁠 때는 못 보겠지만 그다지 바쁘지 않은 상황이라면 메시지를 보고 답변을 줄 것이다.
이와 같이 언제 어떤 이벤트가 발생할지 알 수 없는 것들을 다루는 녀석이 바로 함수 포인터다.

사람이 이런데서는 임기응변을 잘 해야 하듯이
컴퓨터 관점에서 임기응변을 잘 하도록 만들어주는 것이 바로 함수 포인터다.
(결론: 비동기 처리 - 함수 포인터)

*자료구조(스택) 구현하기
*스택 자료구조 그림 그리기 (위에 과제로 함)

*이중 포인터의 이점(트리 계열의 자료 구조를 재귀 호출 없이 구현 가능)

int main(int argc, char ** argv)

***머릿속으로 그린 그림을 코드로 구현할 수 있는 능력을 키우는 것이 자료구조를 학습하는 이유다. 이것을 잘해야 sw를 씹어먹을 수 있다.**
(알고리즘 작업 능력이 월등히 상승하게 됨)