TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 최대성
c3d4s19@naver.com

2018.03.02. – 6,7일차 수업

---

* 3항 연산자

[조건] ? [참일 경우] : [거짓인 경우] ;

Ex) (val[i] & comp) ? 1 : 0 ;

---

* typedef 사용

자료형에 새로운 이름을 부여하고자 할 때 사용

주로 구조체나 함수 포인터에 사용한다.

```
typedef struct _test{
        int* data1;
        double data2;
} Test
```

---

* 메모리 동적 할당

malloc()은 메모리 영역의 heap에 데이터를 할당한다.

(memory allocation의 약자)

Data가 얼마나 들어올지 모르는 상황에서 주로 쓰인다.

Ex) int* newData = (int*)malloc(sizeof(int) * 2);

free()는 heap에서 동적할당한 메모리 영역을 할당 해제하는 역할을 한다.

Ex) free(newData);

calloc()은 malloc()과 완전히 동일한 역할을 하지만 사용 방법에 차이가 있다.

(1번째 인자는 할당할 개수, 2번째 인자는 할당할 크기)

Ex) int* newData = (int*)calloc(2, sizeof(int));

realloc()은 "calloc()", "malloc()"으로 할당한 크기를 조절할 경우 쓰인다. 재할당이라고 보면 된다.

Ex) newData = (int*)realloc(newData, sizeof(int) * 3);

---

* enum(열거형) 함수

네트워크 프로그래밍에서 유용하게 쓰인다

```
typedef enum __packet {
                ATTACK, HOLD, STOP,
                DEATH, KILL = 44, ASSIST
        } packet;
```

---

* memmove(,,) memcpy(,,) 둘다 같은 결과

차이점은 memcpy가 성능이 더 좋지만 메모리 보호가 없고 memmove는 메모리 보호가 확실하지만 성능이 상대적으로 낮다.

Ex) int dst[5], src[5] = {1, 2, 3, 4, 5};

memcpy(dst, src, 3);

memcpy([복사목적지], [복사원본], [복사길이]);

---

* 함수 프로토타입이란?

리턴(return), 함수명(name), 인자(parameter)에

대한 기술서

그렇다면 다음 함수들에 대한 프로토타입은?

(이전에 배웠던 int (*p)[2]; -> int (*)[2] p 이용하면)

Ex1) void (*p)(void);

리턴: void

함수명: p

인자: (*)(void)

-> void 를 리턴하고 void 를 인자로 취하는

함수의 주소값을 저장할 수 있는 변수 p

Ex2) int (* aaa(void))[2];

리턴: int(*)[2]

이름: aaa

인자: void (*)void

Ex3) int (* ddd(void))(void);

리턴: int (*)(void)

이름: bbb

인자: void

Ex4) void ccc(void (*p)(void))

리턴: void

이름: ccc

인자: void (*p)(void)

Ex5) void(* bbb(void (*p)(void)))(void)

리턴: void (*)(void)

이름: bbb

인자: void (*)(void)

---

* 복잡한 함수 포인터 활용법

Ex6) int (* aaa(void))[2] // 실제 문법

int (*)[2] aaa(void) // 보기 편하게 변형

리턴: int(*)[2]

함수명: aaa

인자: void

Ex7) int (*(* bbb(void))(void))[2]; // 실제 문법

int (*)[2](*)(void) bbb(void) // 보기 편하게 변형

리턴(변형 식): int(*)[2](*)(void)

리턴(실제 문법): int (*(*)(void))[2]

함수명: bbb

인자: void

Ex8) void (*signal(int signum, void (*handler)(int)))(int);

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

리턴: void (*)(int)

함수명: signal

인자: int signum 과 void (* handler)(int)

관련 과제 링크

1. http://cafe.naver.com/hestit/788
2. http://cafe.naver.com/hestit/1858

1번 과제 풀이

```
/*
문제 1. (int를 반환하고 float, double, int를 인자로
취하는 함수 포인터)를 반환하고
(float을 반환하고 int 2개를 인자로 취하는
함수포인터)를 인자로 취하는 함수를 작성하여
프로그램이 정상적으로 동작하도록 프로그래밍하시오.
(힌트 : 함수는 총 main, pof_test_main, pof_test1,
pof_test2으로 4개를 만드세요)

return: int (*)(float,double,int)
name: pof1
parameter: float (*)(int, int)
int (*)(float,double,int) pof(float (*)(int, int))
int (* pof(float (*)(int, int)))(float,double,int)
int (* pof(float (*p)(int, int)))(float,double,int){}
```

문제 2. (int 2개를 인자로 취하고 int를 반환하는 함수포인터를 반환하며 인자로 int를 취하는 함수포인터)를
반환하는 함수는
(float을 반환하고 인자로 int, double을 취하는 함수 포인터)를 인자로 취한다.
이를 프로그래밍하고 역시 정상적으로 동작하도록 프로그래밍하시오.
(힌트 : 함수는 총 main, pof_test_main, pof1, subpof1, pof2로 5개를 만드세요)

return: int (*)(int,int) (*)(int)
return: int (*(*)(int))(int,int)
name: pof2
parameter: float (*)(int,double)
int (*(*)(int))(int,int) pof(float (*)(int,double))
int (*(* pof(float (*)(int,double)))(int))(int,int)

int (*)(int,int) (*)(int) pof(float (*)(int,double))
int (*)(int,int) (* pof(float (*)(int,double)))(int)
int (*(* pof(float (*)(int,double)))(int))(int,int)
int (*(* pof(float (*p)(int,double)))(int))(int,int)
*/

```c
#include <stdio.h>

int (* pof_test_main(float (*p)(int, int)))(float,double,int);
float pof_test1(int n1, int n2);
int pof_test2(float n1, double n2, int n3);

int (*(* pof2_test_main2(float
(*p)(int,double)))(int))(int,int);
int (* subpof1(int n))(int,int);
float pof2(int n1, double n2);
int pof1(int n1,int n2);

int (* pof_test_main(float (*p)(int, int)))(float,double,int){
    float res;
    res = p(4,3);
    printf("pof1 res = %f₩n",res);
    return pof_test2;
}

float pof_test1(int n1, int n2){
    return (n1+n2)*0.2345;
}
int pof_test2(float n1, double n2, int n3){
    return (n1+n2+n3)/3.0;
}


int (*(* pof_test_main2(float
(*p)(int,double)))(int))(int,int){
    float res;
    res = p(3, 7.7);
    printf("res = %f₩n", res);

    return subpof1;
}

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

int (* subpof1(int n))(int,int){
    printf("n = %d₩n",n);
    return pof1;
}

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

int main(){
    int res = pof_test_main(pof_test1)(3.7,2.4,7);
    printf("pof_test res = %d₩n",res);
    //-----------------------------------------
    res = pof_test_main2(pof2)(3)(7,3);
    printf("res2 = %d₩n", res);
    return 0;
  }
```

2번 과제 풀이

```
/*
float (* (* test(void (*p)(void)))(float (*)(int, int)))(int, int)
위와 같은 프로토타입의 함수가 구동되도록
프로그래밍 하시오.
실제함수식 -> 변형식
float (*)(int, int) (*)(float (*)(int, int))   test(void
(*p)(void))
리턴: float (*)(int, int) (*p)(float(*)(int, int))
-> float (*(*p)(float(*)(int, int)))(int, int)
이름: test
인자: void (*p)(void)
*/

#include <stdio.h>

//float(*(*)(float(*)(int, int)))(int, int)   test(void(*p)(void))
float(*(*test1(void(*p)(void)))(float(*)(int, int)))(int, int);
float(*test2(float(*)(int, int)))(int, int);
float test3(int, int);
void test4(void);

//float(*(*)(float(*)(int, int)))(int, int)   test(void(*p)(void))
float(*(*test1(void(*p)(void)))(float(*)(int, int)))(int, int){
        printf("test1!₩n");
        p();
        return test2;
}

//float(*)(int, int)   test2(float(*)(int, int));
float(*test2(float(*p)(int, int )))(int, int){
        printf("test2!₩n");
        p(1, 2);
        return test3;
}
float test3(int a, int b) {
        printf("test3!₩n");
        return   (a + b) + 0.33;
}
void test4(void) {
        printf("test4!₩n");
        printf("void ₩n");
}

int main() {
        float result = test1(test4)(test3)(50,6);
        printf("result! = %f₩n", result);
        printf("₩n₩n₩n₩n₩n", result);

        result = test2(test3)(1, 2);
        printf("result = %f₩n", result);
        return 0;
}
```

---

*함수포인터 사용 목적

1. 비동기 처리

2. HW 개발 관점에서 인터럽트

3. 시스템 콜(유일한 SW인터럽트)

여기서 인터럽트들(SW, HW)은

사실상 모두 비동기 동작에 해당한다.

결국 1번(비동기 처리)가 핵심이라는 의미이다.

그렇다면 비동기 처리라는 것은 무엇일까

기본적으로 동기 처리라는 것은

송신하는 쪽과 수신하는 쪽이 쌍방 합의하에만 달성된다. (휴대폰 전화 통화 등)

반면 비동기 처리는 일방적으로 보내서 처리된다.

(이메일. 카톡 등 메신저와 같이 일단 보내 놓으면 상대방이 바쁠때는 못보지만 그다지 바쁘지 않은 상황이면 답을 준다)

이와 같이 언제 어떤 이벤트가 발생할지

알 수 없는 것들을 다루는 것이 함수 포인터이다.

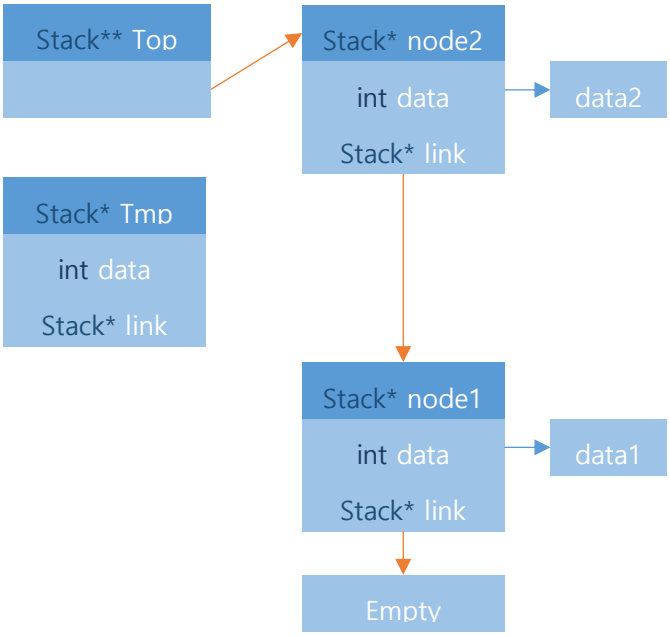
(비동기 처리 –> 함수 포인터 이용)

자료구조 push() 함수 실행 과정 그림 그리기

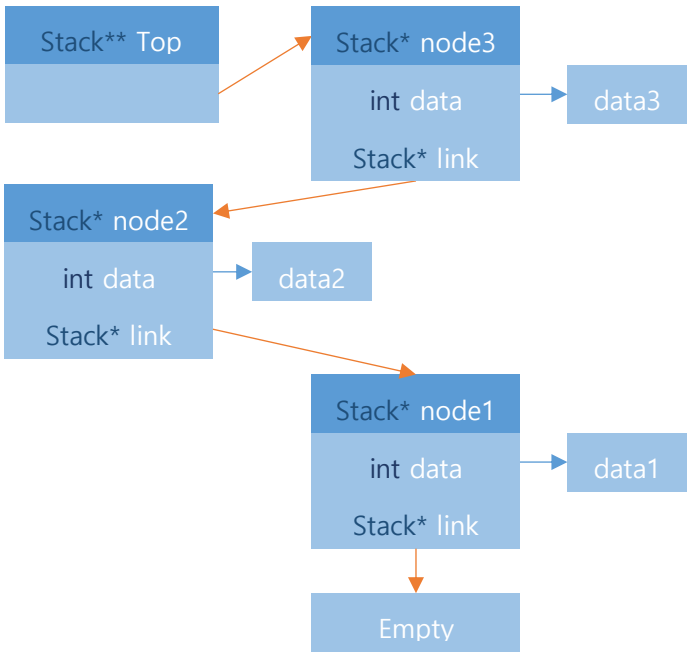
```
struct node{
    int data;
    struct node *link;
};
typedef struct node Stack;
Stack* get_node()
{
    Stack* tmp;
    tmp = (Stack *)malloc(sizeof(Stack));
    tmp->link = EMPTY;
    return tmp;
}
void push(Stack** top, int data){
    Stack* tmp;
    tmp = *top;
    *top = get_node();
    (*top)->data = data;
    (*top)->link = tmp;
  }

int main(){
    Stack* top = EMPTY;

}
```



Stack* Tmp 생성 후 Stack** Top이 가르키는 node2 포인터 값을 새로 생성한 Tmp에 복사하여 넣고 Top이 가르키는 주소에는 새로운 node3의 데이터값을 넣고 Stack* link포인터로 Tmp주소와 연결시킨다.



그러면 다음과 같이 Tmp는 node2의 역할을 하게 되고 *Top이 가르키는 주소값에는 새로운 node3의 데이터들이 보관된다.