EDDI
Electronic Design
Development Institute

에디로봇아카데미
임베디드 마스터 Lv1 과정

제 3기

2022. 02. 11

김원석

CONTENTS

EDDI
Electronic Design
Development Institute

구조체

EDDI
Electronic Design Development Institute

- typedef : 실제 C 타입을 우리가 원하는 간소화된 타입으로 변경하는 작업을 지원한다.

구조체를 사용하는 이유?

- 자신만의 새로운 데이터 타입을 만들 수 있다.
- 결국 구조체 = 데이터 타입
- 즉, 변수나 배열을 만들 때 적용했던 규칙들이 그대로 적용될 수 있다.

ex)

```
typedef struct array_list list;
struct array_list
{
        int data;
        struct array_list *link;
};
```

- array_list라는 구조체를 list라는 데이터 타입으로 간소화
- array_list 구조체 정의

주의사항

```c
// 이 부분은 잘못 만들어진 부분입니다!
// 요런 실수를 하면 안되니 예로 남겨두겠습니다.
void set_both_link (list target1, list target2)
{
        target1.link = &target2;
        target2.link = &target1;
}

int main(void)
{
        // 1.
        struct array_list test1 = { 3, NULL };
        // 2.
        list test2 = { 7, NULL };

        set_both_link(test1, test2);

        print_list_struct(test1);
        print_list_struct(test2);

        return 0;
}
```

```
(gdb)
36              set_both_link(test1, test2);
(gdb)
38              print_list_struct(test1);
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/oem/proj/eddi/academy/EmbeddedMasterLv1/37| /LSH/c/6/a.out

Breakpoint 1, main () at struct_ex2.c:30
30      {
(gdb) n
32              struct array_list test1 = { 3, NULL };
(gdb)
34              list test2 = { 7, NULL };
(gdb)
36              set_both_link(test1, test2);
(gdb) p test1
$1 = {data = 3, link = 0x0}
(gdb) p test2
$2 = {data = 7, link = 0x0}
(gdb) p &test1
$3 = (struct array_list *) 0x7fffffffda90
(gdb) p &test2
$4 = (list *) 0x7fffffffdaa0
(gdb) s
set_both_link (target1=..., target2=...) at struct_ex2.c:24
24      {
(gdb) n
25              target1.link = &target2;
(gdb) p &target2
$5 = (list *) 0x7fffffffda60
(gdb) p &target1
$6 = (list *) 0x7fffffffda70
(gdb)
```

target1과 target2는 test1과 test2의 값에 의한 호출이므로 set_both_link 함수의 스택 프레임에서 값을 바꿔도 main함수 프레임의 test1과 test2에는 영향을 주지 못한다.

주의사항


```c
void set_both_link (list *target1, list *target2)
{
        // 포인터 변수를 통해 구조체 내부에 접근하는 경우엔 '->' 연산자를 사용합니다.
        target1->link = target2;
        target2->link = target1;
}

int main(void)
{
        // 1.
        struct array_list test1 = { 3, NULL };
        // 2.
        list test2 = { 7, NULL };

        set_both_link(&test1, &test2);

        print_list_struct(test1);
        print_list_struct(test2);

        return 0;
}
```

다음과 같이 주소에 의한 호출을 하면 set_both_link 함수의 스택 프레임에서도 주소에 접근해 값을 바꿔 main 함수의 스택 프레임 속 test1과 test2의 값을 변경할 수 있다.

함수 포인터

EDDI
Electronic Design Development Institute

- 앞서 배열 포인터를 다룰때 C언어 특성에 따른 독특한 선언 방식을 보았다.
- 함수 포인터도 마찬가지로 독특한 선언 방식을 가지고 있다.

```
void (*p)(void) = print_test;
// void (*p)(void) ---> void (*)(void) p
p();
//print_test();
```

- 괄호 속 포인터 변수 뒤에 파라미터가 오면

  ->변수명의 오른쪽 괄호를 풀고 변수명을 제일 뒤쪽(오른쪽)으로 옮긴 뒤 다시 괄호를 닫는다.

※ 함수 이름도 결국 포인터다.

함수 포인터

EDDI
Electronic Design Development Institute

```
void (*p)(void) = print_test;
// void (*p)(void) ---> void (*)(void) p
p();
//print_test();
```

- 그럼 위 예시는 다음과 같이 자연스럽게 해석된다.

  ->void 데이터 타입을 반환하고 입력 파라미터가 void 데이터 타입인 함수의 포인터 변수 p

함수 포인터 반환 함수

EDDI
Electronic Design Development Institute

- 이번엔 함수 포인터를 반환하는 함수를 만들어본다.

```
int (* arbiter (void))(void)
{
        printf("조건이 만족되었으므로 이 포인터를 리턴합니다!\n");

        return ret_int_test;
}
```

- 다음과 같이 해석을 고려해 선언해야 한다. 해석해보면 다음과 같이 풀이된다.

  int (*)(void)arbiter(void)

- 즉, int형을 반환하고 입력 파라미터가 void 형인 함수의 포인터를 반환하는 입력 파라미터가 void형인 함수 arbiter 라고 해석할 수 있다.

배열 포인터 반환 함수

EDDI
Electronic Design Development Institute

```
int (* mat_add(int (*src)[2]))[2]
```

- 함수 포인터를 반환하는 함수를 선언할 때와 마찬가지로 배열 포인터를 반환하는 함수를 선언할 때도 해석을 고려해서 선언해야 한다.
- 풀이해보면

  int[2] *mat_add(int[2] *src)
- 즉, int 2개짜리 배열의 포인터를 반환하고 int 2개짜리 배열의 포인터를 입력 파라미터로 하는 함수 mat_add 라고 해석될 수 있다.

Static

EDDI
Electronic Design
Development Institute

- 함수 내부에 static 변수를 선언하면 반드시 해당 함수를 통해서만 static 변수의 값을 얻을 수 있다.
- 완벽하진 않지만 C++이나 Java의 private 역할을 수행할 수 있다.

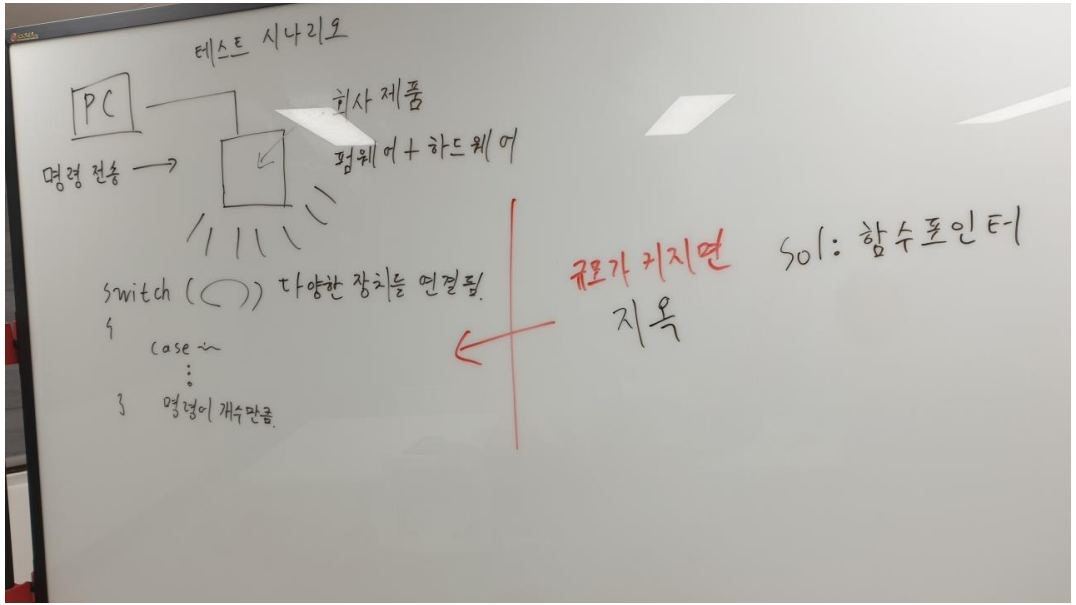
enum

EDDI
Electronic Design Development Institute

```
enum protocol
{
        CAMERA,
        DCMOTOR,
        BLDC,
        PMSM,
        ACIM,
        LED,
        I2C,
        SPI,
        CAN,
        ECAP
};
```

0부터 시작해 1씩 증가하며 값을 지정해준다.

위임자



- 컴퓨터가 입력하는 명령에 따른 동작을 설정할 때, 모두 switch case 문을 통해 설정한다면 관리해야 할 명령어 수가 많아질수록 case는 많아지고 코드가 길어져 코드의 유지 및 보수에 적합하지 않다.

위임자

- 명령어를 입력받으면 그에 맞는 함수가 실행되도록 함수 포인터를 반환하는 위임자 함수를 만든다면 코드의 유지 및 보수에 효과적이다.