EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv1 과정

Ch6.

제 3기

2021. 12. 28

여건영

CONTENTS

struct_ex.c

* 구조체

구조체의 [커스텀 데이터 타입]을 만들어 관리 및 활용 할 수 있다.
가장 중요한 부분은 구조체도 구조체라는 특정한 타입의 데이터를 저장할 수 있는 공간(변수)이다.
변수나 배열 등을 만들 때 적용하였던 규칙이 그대로 적용될 수 있다.

```c
#include <stdio.h>
#include <stdlib.h>

// typedef 는 실제 C 타입을
// 우리가 원하는 간소화된 타입으로 변경하는 작업을 지원함
typedef struct array_list list;

struct array_list
{
        int data;
        struct array_list *link;
        // struct array_list type 데이터의 주소값을 저장하는 공간(변수) link
};

void print_list_struct(list target)
{
        // 구조체 내부에 접근할 때는 '.' 연산자를 활용합니다.
        // '.' 연산자는 구조체 내부에 접근하세요!
        // 라는 의미를 가지는 연산자
        printf("target.data = %d\n", target.data);
        printf("target.link = %d\n", target.link);
}

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

        print_list_struct(test1);
        print_list_struct(test2);

        return 0;
}
```

```
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch6$ ./a.out
target.data = 3
target.link = 0
target.data = 7
target.link = 0
```

struct_ex2.c

* set_both_link(test1, test2) 함수를 사용하여 test1.link와 test2.link에 각 상대 구조체의 주소값을 넣어보자

```
void set_both_link(list target1, list target2)
{
        target1.link = &target2;
        target2.link = &target1;
}
```

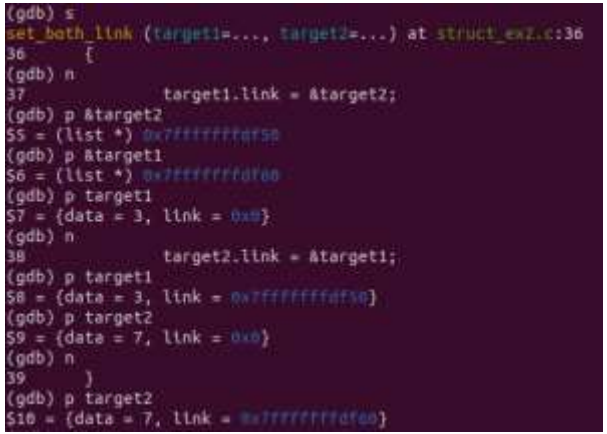
```
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch6$ ./a.out
target.data = 3
target.link = 0
target.data = 7
target.link = 0
```

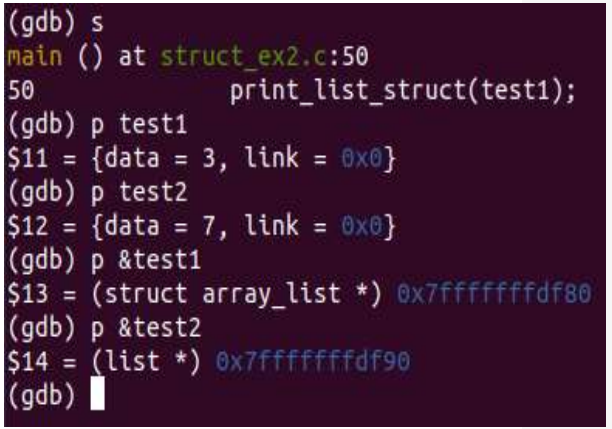
* 결과는? 그대로인 것을 볼 수 있다.
  set_both_link 라는 함수 아래 만들어진 구조체 target1, target2는
  main 함수의 구조체 test1, test2와 엄연히 다르다. (data 값은 복사되어 같을지라도)
  set_both_link 함수 수행 중, target1과 target2는 stack 안에서 만들어지고 다시 사라진다.
  해당 함수 내에서 data가 변경되는 것을 확인할 수 있지만 함수가 종료되면 그 세상은 사라진다.

  아래와 같이 확인해보자.

struct_ex2.c

1. set_both_link 함수를 실행 하기 전, 구조체 test1, tset2의 정보 확인
2. set_both_link 함수 내에서 구조체 target1, target2 정보 확인
3. set_both_link 함수 실행 후, 구조체 test1, tset2의 정보 변화 확인





struct_ex3.c

* set_both_link(&tset1, &test2) 함수를 사용하여 test1.link와 test2.link에 각 상대 구조체의 주소값을 넣어보자

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

```
void print_list_struct(list target)
{
        printf("target.data = %d\n", target.data);
        printf("target.link = 0x%x\n", target.link);

        if(target.link != NULL)
        {
                printf("target.link->data = %d\n", target.link->data);
                // 여기서 test1에 link 에는 test2의 주소값이,
                // test2에 link 에는 test1의 주소값이 들어가 있으므로
                // 주소값을 통해 구조체 내부에 data에 접근한다.
        }
}
```

* 결과는? 변한 것을 볼 수 있다.

test1, test2의 주소값을 받아 접근한다.
set_both_link 함수는 struct 타입 데이터의 주소값을 저장하는 매개변수를 입력 받는다.
여기서 target1, target2는 그냥 주소값을 갖고있는 변수 한 칸이다.
포인터 변수를 통해 해당 주소로 직접 접근하여 구조체 데이터를 바꿀 수 있었다.

아래와 같이 확인해보자.

struct_ex3.c

1. 구조체 test1, test2 data 값 출력
2. set_link 함수 실행 후, 구조체 test.link에 각 구조체 자신의 주소값이 들어갔는지 확인하고
   그 주소값을 통해 해당 구조체 내부에 data를 출력하여 본다.
3. set_both_link 함수 실행 후, 구조체 test.link에 각 상대 구조체의 주소값이 들어갔는지 확인하고
   그 주소값을 통해 해당 구조체 내부에 data를 출력하여 본다.

```c
int main(void)
{
        // 1.
        struct array_list test1 = { 3, NULL };
        // 2.
        list test2 = { 7, NULL };
        print_list_struct(test1);
        print_list_struct(test2);
        printf("\n");
        set_link(&test1, &test2);
        print_list_struct(test1);
        print_list_struct(test2);

        set_both_link(&test1, &test2);
        print_list_struct(test1);
        print_list_struct(test2);

        return 0;
}
```

```
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch6$ ./a.out
target.data = 3    target.link = 0x0
target.data = 7    target.link = 0x0

target.data = 3    target.link = 0xddcf4f60
target.link->data = 3

target.data = 7    target.link = 0xddcf4f70
target.link->data = 7

target.data = 3    target.link = 0xddcf4f70
target.link->data = 7

target.data = 7    target.link = 0xddcf4f60
target.link->data = 3
```

struct_ex4.c

```c
int main(void)
{
        int cnt = 0;
        // 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);

        while(test1.link && cnt++ < 10)
        {
                printf("data = %d\n", test1.link->data);
                test1.link = test1.link->link;
        }
```

```
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch6$ ./a.out
target.data = 3      target.link = 0xbe959910
target.link->data = 7
target.data = 7      target.link = 0xbe959900
target.link->data = 3
data = 7
data = 3
data = 3
data = 3
data = 3
data = 3
data = 3
data = 3
data = 3
data = 3
```

* set_both_link(&test1 ,&test) 함수 실행 후, 각 구조체.link에 상대 구조체의 시작 주소값이 들어간다.
  // test1 = { 3, test2 시작주소}, test2 = { 7, test1 시작주소}

  While 문 동작에서, 처음 한번만 test1.link, 0xbe959910(test2 시작주소)가 가리키는 data 7이 출력되고,
  다시 test1.link 안으로 test2.link 값 0xbe959900(test1 시작주소) 값이 들어간다.
  // 현재 test1 = { 3, test1 시작주소}

  반복을 실행하며, 현재 test1.link(test1 시작주소)가 가리키는 data 값 3이 출력되며
  다시 test1.link 안으로 test1.link가 가리키는 link 값(test1 시작주소)이 들어간다.
  // test1 = { 3, test1 시작주소}
  반복 실행..

struct_ex5.c

```c
int main(void)
{
        int cnt = 0;
        // 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);

        while(test1.link && cnt++ < 10)
        {
                printf("data = %d\n", test1.link->data);
                test1.link = test1.link->link;
        }
```

```
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/37|/GYY/c/ch6$ ./a.out
target.data = 3      target.link = 0x34ff71b0

target.link->data = 7
target.data = 7      target.link = 0x34ff71a0

target.link->data = 3
data = 3
data = 7
data = 3
data = 7
data = 3
data = 7
data = 3
data = 7
data = 3
data = 7
```

* test1 = { 3, test1 시작주소}, test2 = { 7, test2 시작주소}
  tmp = { test1 시작주소 }
  set_both_link(&test1, &test2) 실행 후,
  test1 = { 3, test2 시작주소}, test2 = { 7, test1 시작주소}

  While 문 동작에서, tmp가 가리키는 data 3이 출력되고,
  tmp에 tmp가 가리키는 link, test2 시작주소 가 들어간다. // tmp = { test2 시작주소 }

  다음, tmp가 가리키는 data 7이 출력되고,
  tmp에 tmp가 가리키는 link, test1 시작주소 가 들어간다. // tmp = { test1 시작주소 }
  반복 수행...

pof.c

```c
#include <stdio.h>

void print_test(void)
{
        printf("Hello Test\n");
}

int main(void)
{

        void (*p)(void) = print_test;
        // void (*p)(void) ---> void (*)(void) p
        // 함수도 주소값을 가진다.
        // void 함수명(void) type data의 주소값을 저장하는 공간(변수) p에
        // print_test 함수의 이름(주소)를 넣는다.
        p();
        // print_test();

        return 0;
}
```

pof2.c

```c
#include <stdio.h>

int ret_int_test(void)
{
        return 3;
}

int main(void)
{
        int (*p)(void) = ret_int_test;
        // int 함수명(void) type data의 주소값을 저장하는 공간(변수) p에
        // ret_int_test 함수의 이름(주소)를 넣는다.
        printf("res = %d\n", p());

        return 0;
}
```

pof3.c

```c
#include <stdio.h>

int ret_int_test(void);
// 위임자
// int (*)(void) arbiter (void)
int (* arbiter (void))(void);
// int (함수명)(void) type data의 주소값을 저장하는 공간(변수) arbiter()
// arbiter() << 안에는 함수의 시작 주소가 들어간다.
// ret_int_test의 시작주소가 0x00 이라 가정
// 변수 arbiter()의 값은 [ 0x00 ]

// arbiter()를 출력하면 ret_int_test의 시작주소 0x00 이 출력될 것이고,
// arbiter()()를 출력하면 ret_int_test() 값이 출력될 것이다.
//
int (* arbiter (void))(void)
{
        printf("조건이 만족되었으므로 이 포인터를 리턴합니다!\n");

        return ret_int_test;
}

// int (*)(void)
int ret_int_test(void)
{
        return 3;
}

int main(void)
{
        printf("res = 0x%x\n", arbiter()());
        // arbiter() ===> ret_int_test
        // arbiter()() ===> ret_int_test()
        printf("res = 0x%x\n", arbiter());

        return 0;
}
```

* arbiter()를 단순히 변수라고 생각하여
  arbiter() => p로 치환하여,
  Int (* p)(void);

  코드를 작성하였을 때,
  Compile error 부분
  그리고 그 이유에 대해 좀더 고민해 보겠습니다;