TI DSP,MCU및 Xilinux Zynq FPGA 프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/2/28
수업일수	6일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

1. 문제풀이

- 2_2.c 풀이

2. 정수 2004016을 변수에 저장하고 이것을 char형 포인터로 받는다.
그리고 정수형은 총 4byte로 구성되므로 총 4개의 byte를 볼 수 있을것이다.
각 byte에 숫자가 어떻게 배치되었는지 확인해보자.

*삼항연산자" 조건 ? 참인경우 실행 : 거짓인경우 실행 "

```
#include <stdio.h>

void compute_int_bit(char *val, int size)
{
int i, j;
unsigned char comp = 0;

printf("val = ");

for(i = size - 1; i >= 0; i--)
{
comp = 128;
for(j = 1; j <= 8; j++)
{
//비트검사를하겠다?
if(i == size - 1 && j == 1)
{
//j부호비트를 체크
printf("%d ", val[i] & comp ? 1 : 0);
}
else
{
printf("%d", val[i] & comp ? 1 : 0);
}
comp >>= 1;
//2로나눈다:
}
printf(" ");
}
}

int main(void)
{
int size = sizeof(int);
int val = 2004016;
```

```
compute_int_bit((char *)&val, size);
//캐릭터형 배열로 보겠다는 의미 > 무슨소리인지
return 0;
}
```

~결과

```
val = 0 0000000 00011110 10010100 00110000
```

main함수에서 int의 sizeof를 size라는 변수에 넣고 val이라는 변수에 2004016을 넣는다. compute_int_bit((char *)&val, size); 를 보면 인자로 int형이었던 val를 (char *)&val를 인자로 보내어 char형 포인터로 받게 된다. compute_int_bit에서는 int형 변수 i, j를 선언하고 char형 변수 comp에 0의 초기 값을 넣는다. 그리고 for문을 2개 이용하여 i는3부터 1씩 감소하고 j는 0부터 1씩 증가하여 i == size - 1 와j == 1가 둘 다 1의 값이 나온다면 삼항연산자를 통해 val[i] & comp가 참이면 1을, 거짓이면 0을 출력한다.

comp >>= 1;의 의미를 모르겠음...ㅠㅠ

-2_4.c 풀이

풀이

```
#include <stdio.h>

void pointer_juggling(void)
{
int n1 = 2, n2 = 4, n3 = 7;
int *tmp = NULL;
int *n_p[3] = {&n1, &n2, &n3};
int **npp = NULL;

int i = 0, j = 0;

printf("n1 = %d₩n", *n_p[0]);
printf("n2 = %d₩n", *n_p[1]);
printf("n3 = %d₩n₩n", *n_p[2]);

for(;;)
{
npp = &n_p[i];
tmp = *npp;
*npp = n_p[i + 1];
n_p[i + 1] = n_p[i + 2];
n_p[i + 2] = tmp;

printf("n1 = %d₩n", *n_p[0]);
printf("n2 = %d₩n", *n_p[1]);
printf("n3 = %d₩n₩n", *n_p[2]);

if(j++ == 3)
break;
}
}

int main(void)
{
pointer_juggling();
return 0;
}
```

~결과

```
n1 = 2
n2 = 4
n3 = 7

n1 = 4
n2 = 7
n3 = 2
```

n1 = 7
n2 = 2
n3 = 4

n1 = 2
n2 = 4
n3 = 7

n1 = 4
n2 = 7
n3 = 2

n1, n2, n3 에 각각 2, 4 ,7을 넣고 포인터 배열 n_p에 n1, n2, n3의 주소를 각각 넣는다. 포인터 tmp와 포인터npp에는 NULL을 입력하고 변수 i, j에 초기 값 0을 지정하고 n1, n2, n3의 값을 출력해보면 아까 지정했던 값들이 출력된다.
for문을 이용하여 무한반복 루프를 생성하고 n_p[0]>npp>tmp로 n1의 주소를 옮기고 n_p[1]>npp로 n2의 주소를 옮기고 n_p[2]를 n_p[1]로 옮기고 아까 tmp에 옮겼던 n1의 주소를 n_p[2]로 옮겨 n1, n2, n3를 각각 출력해보면 4,7,2가 출력된다 위의 과정을 반복하면 저글링을 출력할 수 있고 j가 3이 될 때까지 반복한다.

- 문제은행 1번

배열에 문자열을 입력받고 각 배열요과 짝수인 경우만을 출력하는 함수를 작성하라.

```
include <stdio.h>


void print_even_arr_elem(char *str)
{
int i;


for(i = 0; str[i]; i++)
{
if(!(str[i] % 2))
{
printf("res = %d : %c₩n", str[i], str[i]);
}
}
}
int main(void)
{
char str[32] = "Hello Embedded World!₩n";
print_even_arr_elem(str);
return 0;
}
```

~결과

```
res = 72 : H
res = 108 : l
res = 108 : l
res = 32 :
res = 98 : b
res = 100 : d
res = 100 : d
res = 100 : d
res = 32 :
res = 114 : r
res = 108 : l
res = 100 : d
res = 10 :
```

str이라는 배열에 Hello Embedded World!을 입력하고
print_even_arr_elem라는 함수에 배열str을 인자로 보낸다.
변수 i를 선언하고 for문을 이용하여 str의 각 요소를 2로 나누어 나머지가 0일 때에만, 즉 짝수일 때 str의 그 요소를 int형과 char형을 출력하게 한다.

-문제은행 3번

3 77 10 7 4 9 1 8 21 33 아래오 같은 숫자들이 배열에 들어 있다고 가정한다. 이 요소들을 배열에 거꾸로 집어넣어보자.

```
#include <stdio.h>
void rev_order(int *arr1, int *arr2, int size)
{
int i, j;
for(i = size - 1, j = 0; i >= 0; i--, j++)
{
arr2[j] = arr1[i];
}
}
void print_arr(int *arr, int size)
{
int i;
for(i = 0; i < size; i++)
{
if(i != size - 1)
printf("%d ", arr[i]);
else
printf("%d₩n", arr[i]);
}
}
int main(void)
{
int arr[12] = {3, 77, 10, 7, 4, 9, 1, 8, 21, 33};
int reverse_order[12] = {0};
int size = (sizeof(arr) / sizeof(int)) - 1;
print_arr(arr, size);
rev_order(arr, reverse_order, size);
print_arr(reverse_order, size);
return 0;
}
```

~결과

3 77 10 7 4 9 1 8 21 33 0

0 33 21 8 1 9 4 7 10 77 3

arr이라는 배열에 3, 77, 10, 7, 4, 9, 1, 8, 21, 33의 값들을 입력한다. reverse_order라는 배열에는 0으로 초기화 해주고 size라는 변수를 선언하여 배열의 마지막 요소인 숫자를 넣는다. print_arr 함수에 arr과 size를 인자로 보내고 0으로 초기화 된 i가 for문을 통해 1씩 증가하고 size보다 작은 동안 실행되는데, size-1과 다를 때 arr의 요소값을 출력하고 같을 때 arr의 요소 값을 출력한다. rev_order에서는 변수 i, j를 선언하고 for 문을 통해 i는 배열의 마지막요소부터 시작하고 j는 배열의 첫번째 요소부터 시작하여 1씩 감소, 증가하면서 arr1를 arr2에 넣어 배열의 요소를 바꾼다. 그리고 다시 print_arr를 통해 reverse_order의 배열의 요소를 각 출력하면 처음과 거꾸로 된 결과가 출력된다.

-문제은행 4번

3 77 10 7 4 9 1 8 21 33에서 홀수요소 번째 요소의 합과 짝수 번째 요소의 합을 곱하시오.

```
#include <stdio.h>
int mult_even_odd_sum(int *arr, int size)
{
int i, esum = 0, osum = 0;
for(i = 0; i < size; i++)
{
if(i % 2)
{
osum += arr[i];
}
else
{
esum += arr[i];
}
}
return osum * esum;
}
int main(void)
{
int arr[12] = {3, 77, 10, 7, 4, 9, 1, 8, 21, 33};
int size = sizeof(arr) / sizeof(int) - 1;
int res;
res = mult_even_odd_sum(arr, size);
printf("res = %d₩n", res);
return 0;
}
```

~결과

res = 5226

배열arr에 초기 값을 넣고 변수 size에는 배열의 마지막 요소 숫자를 넣는다. 변수 res에는 mult_even_odd_sum(arr, size)의 값을 넣는다. mult_even_odd_sum(arr, size)에서는 배열과 변수를 인자로 받아 for문을 이용하여 i가 10일 때 까지 i값마다 짝수일 때 arr의 i값을 osum에 더해 저장하고 짝수가 아닐때 esum에 arr을 더하고 저장하여 osum과 esum를 곱한 값을 리턴한다 즉 짝수번째 요소끼리 더하고 홀수번째 요소끼리 더하여 곱한 값이다. 리턴한 값(res)을 출력하면 5226이 나오게 된다.

-문제은행 6번

행렬의 곱셈, 덧셈, 나눗셈, 뺄셈에 대해 조사하시오. 숫자를 예로 들어서 계산도 해보시오.

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

void init_2x2_matrix(int (*mat)[2])
{
int i, j;

for(i = 0; i < 2; i++)
{
for(j = 0; j < 2; j++)
{
// 1 ~ 4
mat[i][j] = rand() % 4 + 1;
}
}
}

void print_mat(int (* mat)[2])
{
int i, j;

printf("mat:₩n");

for(i = 0; i < 2; i++)
{
for(j = 0; j < 2; j++)
{
printf("%4d", mat[i][j]);
}
printf("₩n");
}
printf("₩n");
}

void add_2x2_matrix(int (*A)[2], int (*B)[2], int (*R)[2])
{
int i, j;

for(i = 0; i < 2; i++)
{
for(j = 0; j < 2; j++)
{
R[i][j] = A[i][j] + B[i][j];
}
}
}
```

```
void sub_2x2_matrix(int (*A)[2], int (*B)[2], int (*R)[2])
{
int i, j;

for(i = 0; i < 2; i++)
{
for(j = 0; j < 2; j++)
{
R[i][j] = A[i][j] - B[i][j];
}
}
}
void sub_2x2_matrix(int (*A)[2], int (*B)[2], int (*R)[2])
{
int i, j;

for(i = 0; i < 2; i++)
{
for(j = 0; j < 2; j++)
{
R[i][j] = A[i][j] - B[i][j];
}
}
}
```

```
void mult_2x2_matrix(int (*A)[2], int (*B)[2], int (*R)[2])
{
/* a b e f
c d g h

ae + bg af + bh
ce + dg cf + dh */
R[0][0] = A[0][0] * B[0][0] + A[0][1] * B[1][0];
R[0][1] = A[0][0] * B[0][1] + A[0][1] * B[1][1];
R[1][0] = A[1][0] * B[0][0] + A[1][1] * B[1][0];
R[1][1] = A[1][0] * B[0][1] + A[1][1] * B[1][1];
}

int main(void)
{
int mat1[2][2] = {0};
int mat2[2][2] = {0};
int res[2][2] = {0};

// It's for use rand() function.
srand(time(NULL));

init_2x2_matrix(mat1);
print_mat(mat1);

init_2x2_matrix(mat2);
```

```
print_mat(mat2);

add_2x2_matrix(mat1, mat2, res);
print_mat(res);

sub_2x2_matrix(mat1, mat2, res);
print_mat(res);

mult_2x2_matrix(mat1, mat2, res);
print_mat(res);

return 0;
}
```

~결과

```
mat:
1 4
2 4

mat:
4 4
4 4

mat:
5 8
6 8

mat:
-3 0
-2 0

mat:
20 20
24 24
```

2.typedef

자료형에 새로운 이름을 부여하고자 할 때 사용한다. 주로 구조체나 함수 포인터에 사용한다.

예제1)

```
#include<stdio.h>

typedef int INT;
typedef int* PINT;

int main(void)
{
INT num=3;
PINT ptr=&num;

printf("num=%d₩n",*ptr);
return 0;

}
```

~결과

```
num=3
```

typedef를 이용해서 int형을 INT로 사용할 수 있게 되었고, int*대신에 PINT라는 포인터를 사용할 수 있게 되었다. main함수를 보게 되면 INT라는 자료형의 num이라는 변수가 3으로 초기값을 입력받고, ptr이라는 포인터가 num의 주소를 받게 된다. 이를 printf로 출력하게 되면 *ptr은 num의 주소를 가르키고 num은 3의 값을 가지니 3의 결과가 출력된다.

예제2)

```
#include<stdio.h>

typedef int INT[5];

int main(void)
{
int i;
INT arr={1,2,3,4,5};


for(i=0;i<5;i++)
printf("arr[%d]=%d₩n",i,arr[i]);


return 0;
}
```

~결과

```
arr[0]=1
arr[1]=2
arr[2]=3
arr[3]=4
arr[4]=5
```

INT옆에 []가 없어도 배열처럼 사용할 수 있다. Typedef int INT[5]; 라고 위에서 지정했기 때문이다. 배열을 처음 typedef로 자료형을 바꿀 때에는 []를 붙여주어야 한다. 하지만 사용할 때에는 []가 없어도 사용할 수 있다.

위의 예제를 보면 typedef로 int형 배열을 INT라는 자료형으로 바꾸었다. main함수에서 i라는 변수를 선언하고 INT형 배열에 초기값을 넣었고 for문에서 배열의 요소를 하나하나 출력하니 그 값들이 나오게 되었다.

-malloc() Function, free Function

malloc()은 메모리구조상 힙에 데이터를 할당한다. 데이터가 계속해서 들어올 경우 얼만큼의 데이터가 들어오는 건지 알 수 없기 때문에 들어올 때 마다 동적으로 할당할 필요성이 있다. 동적으로 메모리가 할당 될 때 사용된다.

ex) (자료형 *)malloc(sizeof(자료형) *할당할 개수)
    char *str_ptr=(char *)malloc(sizeof(char) *20);

free()는 메모리 구조상 힙에 데이터를 할당 해제한다 malloc()의 반대 역할을 수행한다.

ex) free(이름);
    free(str_ptr);

calloc() 은 malloc과 완전히 동일하다. 사용방법에 차이가 있는데 첫번재 인자는 할당할 개수, 두번째 이자는 할당할 크기이다. 즉 call(2,sizeof(int))는8tyed 공간을 할당한다.

ex) (자료형 *)calloc(할당할 개수, sizeof(자료형))
    int *num_ptr=(int*)calloc(2,sizeof(int));

위의 예제에서 밑줄 친 부분은 강제형변환으로 형을 지정해주는 역할을 한다. malloc과 calloc은 본인의 취향에 따라 사용하면 된다.

예제3)

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

int main(void)
{
char *str_ptr=(char *)malloc(sizeof(char) *20);

printf("Input String:");
scanf("%s",str_ptr);

if(str_ptr !=NULL)
printf("string=%s₩n", str_ptr);

free(str_ptr);

return 0;
}
```

~결과
20바이트의 동적 변수가 선언되어 19개의 문자와 하나의 널문자를 입력 가능하다.
calloc과 malloc을 사용하기 위해서는 #include<stidlib.h>를 사용해 헤더 파일을 불러와야한다. 그리고 main함수에서 char형의 메모리를 20개씩 동적할당하는 malloc을 str_ptr포인터에 지정한다. 그리고 scanf로 str_ptr에 값을 입력받고 malloc을 그만 사용하기 위해 free를 사용한다.

예제4)

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

int main(void)
{
int *num_ptr=(int*)calloc(2,sizeof(int));

printf("Input Interger:");
scanf("%d%d",&num_ptr[0],&num_ptr[1]);

if (num_ptr !=NULL)
printf("Integer=%d,%d₩n", num_ptr[0], num_ptr[1]);

free(num_ptr);

return 0;

}
```

~결과

```
Input Interger:1 3
Integer=1,3
```

calloc또한 #include<stdlib.h>로 헤더파일을 불러와 사용한다. int형의 메모리를 2개 할당받아 이를 포인터 num_ptr에 지정하고 scanf로 사용자에게서 값을 받아 num_ptr이 빈 공간이 아니면 입력받은 값 2개를 출력한다. calloc또한 free를 이용하여 메모리를 할당받는걸 그만두게 한다.

-구조체

사용하는 이유 : 문자열, 숫자를 한번에 묶어서 관리하고 싶을 때 사용한다.

예제5)

```
#include<stdio.h>

struct pos
{
double x_pos;
double y_pos;
};

int main(void)
{
double num;
struct pos position;
//pos까지가 데이터 타입이다.
num=1.2;
position.x_pos=3.3;
position.y_pos=7.7;

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

return 0;
}
```

~결과

```
3.300000
7.700000
sizeof(position)=16
```

main함수에서 num이라는 더블형 변수를 하나 선언하고 pos라는 position의 구조체를 선언한다. pos는 double형 x_pos, y_pos변수로 구성되어있다.
num에 1.2의 값을 넣고 구조체 position에 위차한 x_pos에 3.3의 값을 입력하고 position에 위차한 y_pos에 7.7을 입력한다. 이를 printf로 출력하면 3.3과 7.7이 출력되고 position의 크기는 double형의 변수가 2개 있으므로 16byte가 출력된다.

예제6)

```
#include<stdio.h>
#define NAME_LEN 30
#define ID_LEN 15

typedef struct __id_card{
char name[NAME_LEN];
char id[ID_LEN];
unsigned int age;
} id_card;


int main(void)
{
int i;
id_card arr[2]={
{"Marth Kim","800903-1012589",34},
{"July Eun","830108-1023417",31}
};

for(i=0;i<2;i++)
{ printf("name=%s,id=%s,age=%d₩n",
arr[i].name, arr[i].id, arr[i].age);
}
return 0;
}
```

~결과

```
name=Marth Kim,id=800903-1012589,age=34
name=July Eun,id=830108-1023417,age=31
```

#define을 하여 NAME_LEN은 30으로 치환하고 ID_LEN은 15로 치환한다. main함수에서 i라는 변수를 선언하고 __id_card라는 구조체의 타입을 id_card로 지정한다. 구조체는 name이라는 배열과 id라는 배열, 양수만이 저장되는 age라는 변수가 있다.
다시main함수를 보면 id_card의 자료형인 배열 arr에 초기 값을 설정하고 for문으로 각 배열의 요소를 출력한다.

예제7)

```
#include<stdio.h>
#include<stdlib.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;
}
```

~결과

city=Seoul, name=Marth Kim, id=800903-1012589, age=34

__id_card라는 구조체를 id_card라는 이름의 타입으로 지정하고 __city라는 구조체를 city라는 이름의 타입으로 지정한다. main함수를 보면 int형 변수 i가 선언되어있고 city라는 타입의 info라는 이름의 구조체에 초기화 되어있다. 그리고 이를 printf로 출력하게되는데 info라는 구조체에 city라는 요소 값, info의 구조체에서 card의 구조체 안의 name이라는 요소, id, age라는 요소를 찾아가 값을 출력해낸다.

예제8)

```
#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;
}
```

~결과

```
city=Seoul, name=Marth Kim, id=800903-1012589, age=33
```

__id_card라는 구조체의 데이터 타입을 id_card로 선언하고 __city라는 구조체 데이터 타입을 city로 지정한다. main함수를 보면 int형 변수 i와 데이터타입 city의 구조체 info가 초기화 되어있고, malloc으로 id_card크기만큼을 동적할당한다. 총 20byte로 예상된다.
또한 info구조체에서 card에 접근하고 포인터인 name은 포인터를 사용하였기 때문에 '.' 대신 '->'를 이용하여 접근하였. 그리하여 그 주소 값에 값들을 넘기고 이를 출력해보면 넘긴 값들이 출력되고 malloc을 이용하였기 때문에 free를 사용하여 동적할당을 그만둔다.

예제9)

```
#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)++; //val=4
data_p=data_p->data_ref;
if(!(i%2))
printf("₩t"); //2개마다 tap넣어라
}
printf("₩n");
return 0;
}
```

*data_p >> 포인터
d1, d2 >>구조체
~결과
3 7 4 8 5 9 6 10 7 11

__data라는 구조체를 data라는 자료형으로 지정하고 main함수에서는 int 형 변수 i와 data형 포인터 *data_p를 선언한다. data타입의 구조체 d1과 d2에 초기값을 설정하고 d1의 구조체의 data_ref라는 요소에 d2의 주소 값을 저장하고 d2의 구조체의 data_ref라는 요소에 d1의 주소값을 저장한다. data형의 data_p라는 포인터에는 d1의 주소 값을 넣고 이를 for문을 이용하여 data_p에 저장된 주소값을 찾아가 출력하고,

>>모르겟음

-enum

#define 하는걸 편하게 한다.

예제10)

```
#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;
}
```

~결과

```
enum num =0
enum num =1
enum num =2
enum num =3
enum num =4
enum num =5
enum num=44
enum num=45
enum num=46
```

>>모르겟음 머라는걸가..?

-함수포인터

함수포인터도 주소이다. 함수 또한 주소를 저장할 수 있음.

예제11)

```
#include<stdio.h>

void aaa(void){
printf("aaa called₩n");
}

void bbb(void(*p)(void)){
p();
printf("bbb called₩n");
}

int main(void){
bbb(aaa);
return 0;
}
```

~결과

```
aaa called
bbb called
```

main함수에서 bbb라는 함수에 aaa라는 인자를 넣는데 aaa또한 함수이다. aaa를 먼저 살펴보면 aaa called라는 문장을 출력하는 역할을 하고 리턴하는 값이 없고 인자로 받는 것도 없다. bbb라는 함수를 살펴보면 void로 값을 리턴하는 것이 없고, 리턴값과 인자를 내보내고 받는것이 없는 포인터를 인자로 받는다. 그리고 포인터 p를 부르고 bbb called라는 문장을 출력한다. main에서 aaa를 인자로 받았으니 p();에서 함수 aaa를 실행하게 되어 aaa called를 출력하고 아래에 bbb called도 출력하게 된다.

* 함수 프로토타입이란?
리턴, 함수명, 인자에 대한 기술서이다.

void (* signal(int signum, void (* handler)(int)))(int); 라는 함수가 있다.
int (*p)[2]; 는 int (*)[2] p 와 같은 것인데 이를 위의 함수에도 적용하면

리턴: void (*)(int)
함수명: signal
인자: int signum 과 void (* handler)(int) 주황색으로 표시한 것으로 구분할 수 있다.

void (*p)(void):
예제 11에서 나온 위의 함수는 void 를 리턴하고 void 를 인자로 취하는 함수의 주소값을 저장할 수 있는 변수 p이다.

>>의문점
int (*p)[2]; 는 int (*)[2] p 와 같다는데 먼소린지 머르겟슴

예제12)

```
#include <stdio.h>

typedef struct test_class
{
int in1;
int in2;
double dn1;
double dn2;
int (*int_op)(int, int);
double (*double_op)(double, double);
} tc;

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

int imul(int n1, int n2)
{
return n1 * n2;
}

double dadd(double n1, double n2)
{
return n1 + n2;
}

int main(void)
{
int res;
double dres;
tc tc_inst = {3, 7, 2.2, 7.7, NULL, NULL};

tc_inst.int_op = iadd;
res = tc_inst.int_op(tc_inst.in1, tc_inst.in2);
printf("res = %d₩n", res);

tc_inst.int_op = imul;
res = tc_inst.int_op(tc_inst.in1, tc_inst.in2);
printf("res = %d₩n", res);

tc_inst.double_op = dadd;
dres = tc_inst.double_op(tc_inst.dn1, tc_inst.dn2);
printf("dres = %lf₩n", dres);

return 0;
}
```

~결과

```
res = 10
res = 21
```

dres = 9.900000

test_class라는 구조체를 tc라는 자료형으로 지정하고 iadd, imul, dadd라는 함수들이 지정되어 있다. 이 함수들은 각각 n1과 n2를 더하고, 곱하고 double형으로 더하는 함수들이다.
main함수를 살펴보면 int형의 res라는 변수와 double형의 dres라는 변수가 지정되어있고 tc라는 자료형의 tc_inst라는 이름의 구조체가 초기화되어있다.
tc_inst라는 구조체 요소의 int_op로 접근하여 iadd라는 함수를 넣는다. 변수 res에는 tc_inst라는 구조체의 int_op함수에 tc_inst.in1, tc_inst.in2라는 구조체 인자를 전달하여 리턴받은 값을 넣는다. 그리고 이를 printf로 res 값을 출력하면 iadd는 인자로 int n1과 int n2를 받았고 두개를 합한 값을 리턴하여 res에저장되었다. 따라서 10이 출력되게 되고 밑의 6줄도 마찬가지로 imul, dadd의 함수들의 리턴 값이 출력될 것이다.