TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

-5일차-

팩토리얼 소스

```
 1 #include<stdio.h>
 2
 3 int fact(int num){
 4       int first = 1;
 5       while(num>0)
 6             first = first*num--;
 7       return first;
 8 }
 9
10 int main(void){
11       int result, fact_val;
12       printf("계산할 팩토리얼을 입력하시오:");
13       scanf("%d",&fact_val);
14       result=fact(fact_val);
15       printf("%d번째 항의 수는 = %d₩n", fact_val, result);
16       return 0;
17 }
```

결과값 : (입력) 5
5번째 항의 수는 = 120

배열이 필요한 이유는?

천개정도의 값을 저장하려면 변수가필요한데
변수 1000개를 선언하려면 무엇이 필요한가?

100개의 변수가 필요한 경우를 생각해보자
학교의 학생 관리 System에서 명단에 문자열을 담음
다수의 변수에 값을 할당 시 for문을 이용할 수 있음

배열 소스-1

```
#include<stdio.h>
int main(void)
{
      int i;
      int num [7] ;

      for(i=0; i<7; i++)
      {
```

```
            num[i]=i;
            printf("num[%d]=%d₩n",i,num[i]);
        }
        return 0;
 }
```

결과값 :

num[0]=0
num[1]=1
num[2]=2
num[3]=3
num[4]=4
num[5]=5
num[6]=6

설명 :

num
| || | ||||
0 1 2 3 4 5 6

i는 0부터 시작해서
num[0] 이 출력
i++되서 i가 1이되고
num[1] 이 출력

이와 마찬가지로 나머지 2 3 4 5 6 도 출력

배열 소스-2

```
#include<stdio.h>
int main(void)
{
            int i;
            int num1_arr[]={1,2,3,4,5};
            int num2_arr[3]={1,2,3};

            int len1=sizeof(num1_arr)/sizeof(int);
            int len2=sizeof(num2_arr)/sizeof(int);

            printf("num1_arr length = %d₩n",len1);
            printf("num2_arr length = %d₩n",len2);
```

```
	for(i=0; i<len1; i++) //위쪽이 어떻든 정상적으로 for문은 동작
	{
		printf("num1_arr[%d] = %d₩n",i, num1_arr[i]);
	}

	for(i=0; i<len2; i++)
	{
		printf("num2_arr[%d] = %d₩n",i, num2_arr[i]);
	}
	return 0;
}
```

결과값 :

```
num1_arr length = 5
num2_arr length = 3
num1_arr[0] = 1
num1_arr[1] = 2
num1_arr[2] = 3
num1_arr[3] = 4
num1_arr[4] = 5
num2_arr[0] = 1
num2_arr[1] = 2
num2_arr[2] = 3
```

예제소스

```
 1 #include<stdio.h>
 2
 3 int main(void)
 4 {
 5       int i;
 6       int num1_arr[7] = {1,2,3};
 7
 8       for(i=0; i<7; i++)
 9       {
10            printf("num1_arr[%d]=%d₩n",i,num1_arr[i]);
11       }
12
13       return 0;
14 }
```

결과값:

num1_arr[0]=1
num1_arr[1]=2
num1_arr[2]=3
num1_arr[3]=0
num1_arr[4]=0
num1_arr[5]=0
num1_arr[6]=0

비는칸은 0으로 기입됨

---

character type array
char형 배열이 필요한 이유?
string인 문자열 "I'm Marth kim"은 변경이 불가능
char형 배열은 내부 데이터 변경이 가능하다
마지막 data에 Null Character가 필요함
//어디까지가 문자의 끝인지 알수가없어서 null값을 넣는다.

Null Character는 무엇인가?

Null문자는 문자열의 마지막을 의미
//단위행렬
등차적표기의 단위행렬
예제소스 arr3.c

```
 1 #include<stdio.h>
 2
 3 int main(void)
 4 {
 5       int arr[4][4];
 6       int i, jcl;
 7
 8       for(i=0; i<4; i++)
 9       {
10             for(j=0; j<4; i++)
11                   {
12                         if(i==j)
13                               arr[i][j]=1;
14                         else
15                               arr[i][j]=0;
16                   }
```

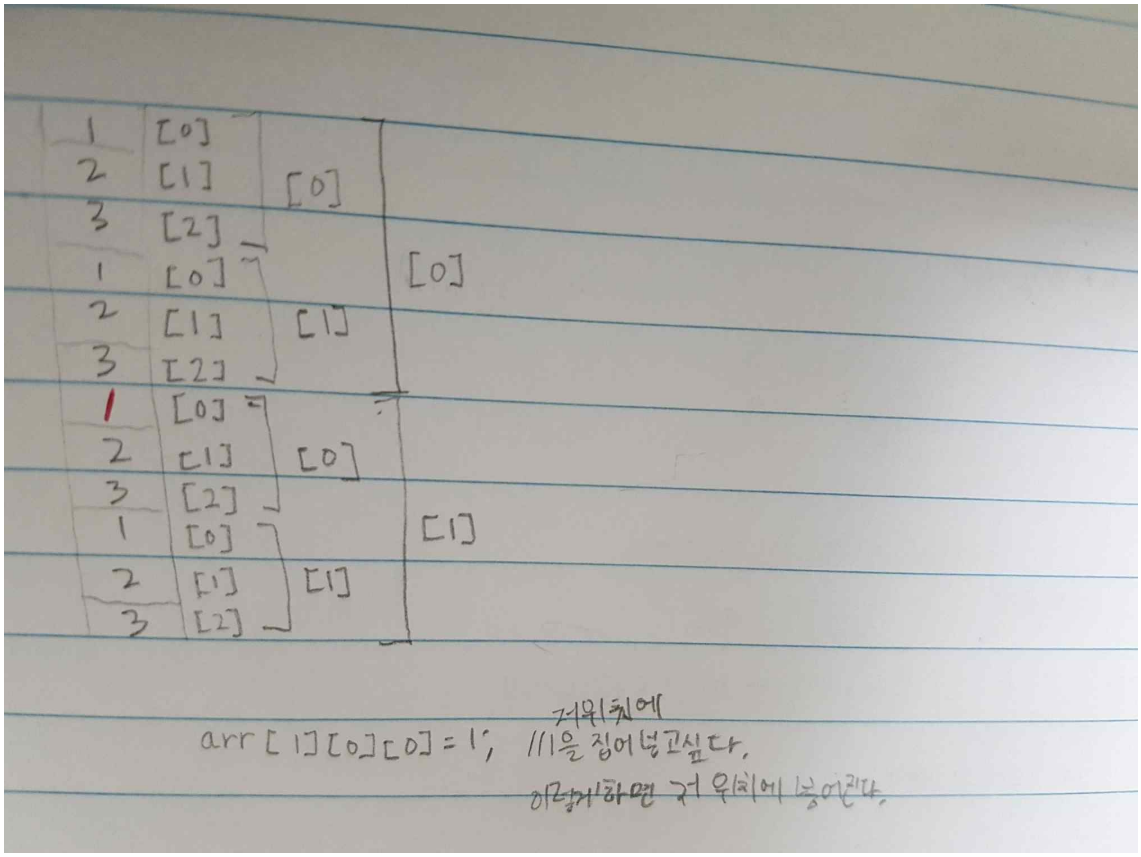
```
17        }
18        for(i=0; i<4; i++)
19        {
20              for(j=0; j<4; j++)
21              {
22                    printf("%d",arr[i][j]);
23              }
24              printf("₩n");
25        }
26        return 0;
27  }
```

결과값:

```
num1_arr length = 5
num2_arr length = 3
num1_arr[0] = 1
num1_arr[1] = 2
num1_arr[2] = 3
num1_arr[3] = 4
num1_arr[4] = 5
num2_arr[0] = 1
num2_arr[1] = 2
num2_arr[2] = 3
```

메모리상에 위 arr3.c가 어떻게 위치해있는지

다중 배열을 사용하는 이유는?
사실 C너어에서 배열에 차원 개념은 없음
교육의 편의상 차원 개념을 도입 //차원은 없
2차원 배열의 경우 행렬의 표현에 용이함
2차원 배열의 경우 [x][y]로 x명의 y개 과목을 관리가능
3차원 배열은 위 경우에서 반이 z개 있을경우 관리 가능




```
arr[2][2][3] =  {
                    {{1,2,3},{1,2,3}},
                    {{1,2,3},{1,2,3}}
                    };
```

예제소스 arr4.c

```
 1 #include<stdio.h>
 2
 3 int main(void)
 4 {
 5       int arr[2][2] = {{10,20},{30,40}};
 6       int i, j;
 7
 8       for(i=0; i<2; i++)
 9       {
10             for(j=0; j<2; j++)
11             {
12                   printf("arr[%d][%d] = %d₩n",i,j,arr[i][j]);
```

```
 13              }
 14        }
 15
 16        return 0;
 17 }
```

결과값 :

```
arr[0][0] = 10
arr[0][1] = 20
arr[1][0] = 30
arr[1][1] = 40
```

예제소스 arr5.c

```
#include<stdio.h>

int main(void)
{
        int arr[4] = {10,20,30,40};
        int i;

        printf("address = %p₩n",arr); //%p는 주소값뿌릴때쓰는거

        for(i=0; i<4; i++)
        {
                printf("address = %p, arr[%d] = %d₩n", &arr[i]->이거랑, i, arr[i]);
        }

        return 0;
}
```

결과값: //숫자값들 차이 4씩나고 첫번째와 두번째 출력주소값이 같음
배열이 int형이니까 4바이트씩 1열로 나열되있으니까

```
address = 0x7ffdb632d8a0
address = 0x7ffdb632d8a0, arr[0] = 10
address = 0x7ffdb632d8a4, arr[1] = 20
address = 0x7ffdb632d8a8, arr[2] = 30
```

address = 0x7ffdb632d8ac, arr[3] = 40

-포인터
예제소스 pointer.c

```
#include<stdio.h>

int main(void)
{
        int arr[3] = {1,2,3};
        int *p = arr;
        int i;

        for(i = 0; i<3; i++)
                printf("p[%d] = %d₩n", i, p[i]);
 // p[%d] 안에는 i값이 들어가고 p[i]값은 배열값이 들어간다

        return 0;
}
```

결과값:
p[0] = 1
p[1] = 2
p[2] = 3

다시 배열쪽

일반배열과 이중배열의 관계는?
[2][3]의 경우 {{1,2,3},{4,5,6}}과 같은 방식으로함
이중배열은 일반배열이 배열형태로 모인 것

예제소스 arr6.c

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

```
int main(void)
{
	int arr[3][4];

	printf("arr address = %p₩n",arr);
	printf("arr[0] address = %p₩n",arr[0]);
	printf("arr[1] address = %p₩n",arr[1]);
	printf("arr[2] address = %p₩n",arr[2]);

	return 0;
}
```

결과값:

```
arr address = 0x7ffc65132200
arr[0] address = 0x7ffc65132200
arr[1] address = 0x7ffc65132210
arr[2] address = 0x7ffc65132220
```

//10씩늘음

arr[3]은 main()함수내의 지역 변수 arr_print로 위 배열의 주소가 전달됨

즉, arr_print 함수에서는 main()함수의 지역변수인
arr[3]에 접근하여 값을 수정할 수 있음

```
 1 #include <stdio.h>
 2
 3 void arr_print(int arr[])
 4 {
 5       int i;
 6       for(i=0; i<3; i++)
 7             printf("%4d",arr[i]); //4자리에 맞춰서 나와라
 8 }
 9
10 int main(void)
11 {
12       int arr[3] = {3,33,333};
13       arr_print(arr);
14       return 0;
15 }
```

결과값: 3 33 333

예제소스 pointer2.c

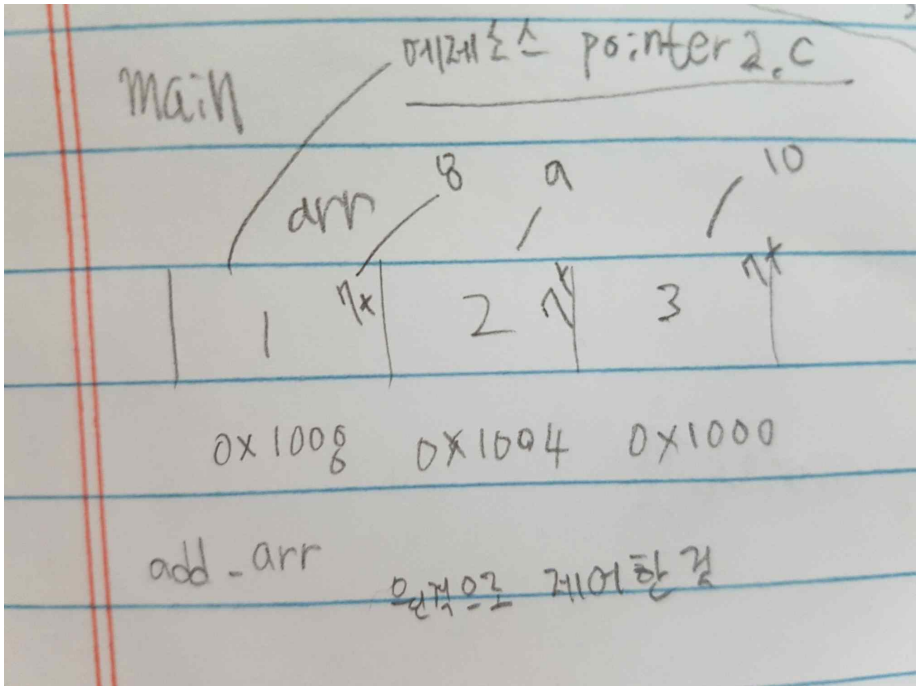
```
#include<stdio.h>

void add_arr(int *arr)
{
	int i;
	for(i=0; i<3; i++)
	{
		arr[i] += 7;
	}
}

void print_arr(int *arr)
{
	int i;
	for(i=0; i<3; i++)
	{
		printf("arr[%d] = %d₩n", i, arr[i]);
	}
}

int main(void)

{
	int arr[3] = {1,2,3};
	add_arr(arr);
	print_arr(arr);

	return 0;
}
```

결과값 :

```
arr[0] = 8
```

arr[1] = 9
arr[2] = 10

//이와같이 주소를 모르면 못들어가지만 주소를 알면 어디에있든 해당 변수를 제어할수있다



재확인.

```
 1 #include<stdio.h>
 2
 3 void add_arr(int *arr)
 4
 5
 6 {
 7
 8       int i;
 9       for(i=0; i<3; i++)
10       {
11             arr[i] += 7;
12       }
13
14 }
15
16 void print_arr(int *arr)
17 {
```

```
18        int i;
19        for(i=0; i<3; i++)
20        {
21              printf("arr[%d] = %d₩n", i, arr[i]);
22        }
23  }
24
25
26  int main(void)
27
28  {
29        int i;
30        int arr[3] = {1,2,3};
31        add_arr(arr);
32        print_arr(arr);
33
34        printf("real₩n");
35
36        for(i=0; i<3; i++)
37        {
38              printf("arr[%d] = %d₩n", i, arr[i]);
39        }
40
41        return 0;
42  }
```

결과값:
arr[0] = 8
arr[1] = 9
arr[2] = 10
real
arr[0] = 8
arr[1] = 9
arr[2] = 10

포인터는 무엇일까?

주소
조금 더 엄밀하게 주소를 저장할 수 있는 변수

무언가를 가르키는 녀석
pointer의 크기는 HW가 몇 bit를 지원하느냐에 따름
그런데 왜 HW에 따라 달라질까?

다같이 담을순없지만 가장 큰사이즈가 필요

최상위 메모리 주소 = 0xffffffff
32bit(4byte)를 가정함

최하위 메모리주소 0x0

포인터는 어떻게 선언하는가?
가르키고 싶은 녀석의 자료형을 적음
주소값을 저장할 변수명(이름)을 적음
변수명 앞에 '*'을 붙임
자료형 없이 변수명 앞에 '*'만 붙은 경우
해당 변수가 가르키는 것의 값을 의미함

```
int num=7;
int * = &num; //포인터 선언
printf("*p
//나는 int형의 어떤 데이타를 받을수있는 변수p다
 &num

//포인터 받을려는 주소를 적기
```

포인터? 주소
배열? 주소

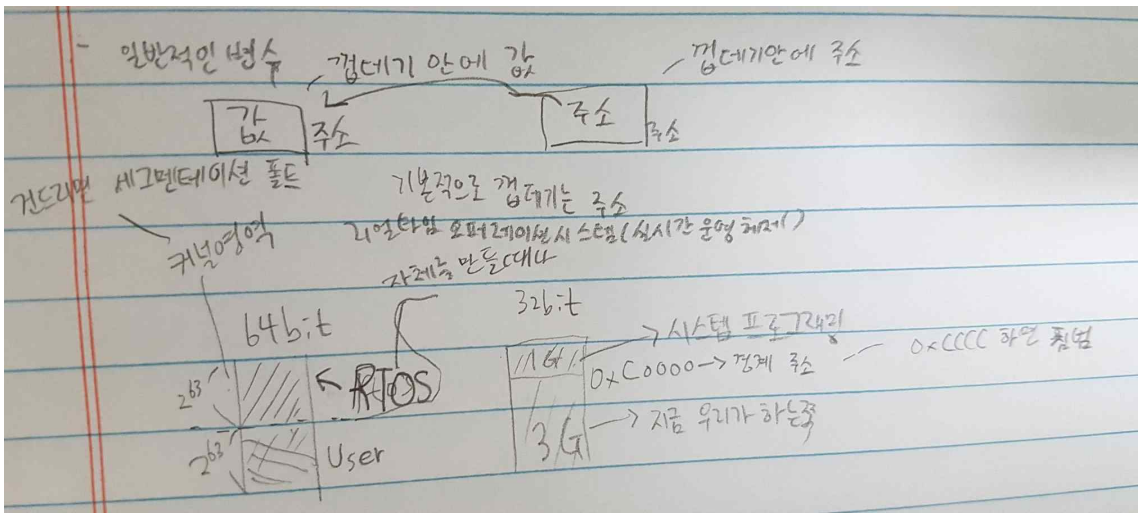
Example

```
1 #include<stdio.h>
2
3 int main(void)
4 {
5       int *ptr;
6
7       printf("ptr = %p₩n", ptr);
8       printf("ptr value = %d₩n", *ptr);
9
```

```
10      *ptr = 27;
11
12      printf("ptr value = %d₩n", *ptr);
13      return 0;
14 }
```

//세그멘테이션 폴트가 나는이유 : 접근하지말아야할곳에 접근을 해서

결과값:
ptr = (nil)
Segmentation fault (core dumped)



기계어를 알면 포인터, 배열, 변수등의
모든것을 주소 관점에서 접근 할 수 있어서 매우 쉬워짐

*Segmentation Fault가 나는 이유?

우리가 기계어를 보면서 살펴봤던 주소값들이 사실은 전부 가짜 주소

이 주소값은 엄밀하게 가상 메모리 주소에 해당하고
운영체제의 paging 메커니즘을 통해서
실제 물리 메모리의 주소로 변환된다.
(윈도우도 가상 메모리 개념을 베껴서 사용한다)
그렇다면 당연히 맥(유닉스)도 사용함을 알 수 있다.

가상 메모리는 리눅스의 경우

32비트 버전과 64비트 버전이 나뉜다.

32비트 시스템은 2^32=4GB 의 가상 메모리 공간을 가짐
여기서 1:3으로 1을 커널이 3을 유저가 가져간다.
1은 시스템(HW, CPU, SW 각종 주요 자원들)에
관련된 중요한 함수 루틴과 정보들을 관리하게 된다.
3은 사용자들이 사용하는 정보들로
문제가 생겨도 그다지 치명적이지 않은 정보들로 구성됨

64비트 시스템은 1:1로 2^63 승에 해당하는 가상메모리를 각각 가진다.
문제는 변수를 초기화하지 않았을 경우 가지게 되는 쓰레기값이 0xCCCCC...CCC로 구성됨이다.

32비트의 경우에도 1:3 경계인 0xc0000000 을 넘어가게됨
64비트의 경우엔 시작이 C이므로 이미 1:1경계를 한참 넘어감

그러므로 접근하면 안되는 메모리 영역에 접근하였기에
엄밀하게는 Page Fault(물리 메모리 할당되지 않음) 가
발생하게 되고 원래는 Interrupt가 발생해서
Kernel 이 Page Handler(페이지 제어기)가 동작해서
가상 메모리에 대한 Paging처리를 해주고
실제 물리 메모리를 할당해주는데
문제는 이것이 User쪽에서 들어온 요청이므로
Kernel쪽에서 강제로 기각해버리면서
Segmentatuin Fault가 발생하는 것이다.

실제 Kernel 쪽에서 들어온 요청일 경우에는
위의 메커니즘에 따라서 물리 메모리를 할당해주게 된다.

예제소스

```
1 #include<stdio.h>
2 int main(void){
3       int num=3;
4
5       *(&num) += 30;
6       printf("num=%d₩n",num);
7       return 0;
8 }
```

결과값 : num=33

예제소스

```
 1 #include<stido.h>
 2
 3 int main(void)
 4       char str1[33] = "Pointer is important!";
 5       char *str2 = "Pointer is important!";
 6
 7       printf("str1 = %s₩n",str1);
 8       printf("str2 = %s₩n",str2);
 9       return 0;
10       }
```

데이타에 저장되지만 어쨋든 주소값을 가지고갈수있을거고 주소값을 1바이트마다 순차적으로 포인터는 주소값을 받아오는것
배열처럼 쓸 수 있고 %s를 쓰면 양쪽다 문자열이 나온다

그 주소값은 Pointer에 대한 Pointer로 '*'

int *p =#//나는 주소값을 받겠습니다
int **p = &p; //나는 주소에 주소를 받겠습니다

3뿌리는 방법
p(num)
p(*p)
p(**pp) //

예제소스 pointer5.c

```
 1 #include<stdio.h>
 2
 3 int main(void)
```

```
 4 {
 5        int  num=3;
 6        int  *p=&num;
 7        int  **pp=&p;
 8        int  ***ppp  =  &pp;
 9
10        printf("num=%d₩n",num);
11        printf("*p=%d₩n",*p);
12        printf("**pp=%d₩n",**pp);
13        printf("***ppp  =  %d₩n",  ***ppp);
14
15        return  0;
16 }
```

결과값:

```
num=3
*p=3
**pp=3
***ppp  =  3
----------------------
```

예제소스 pointer6.c

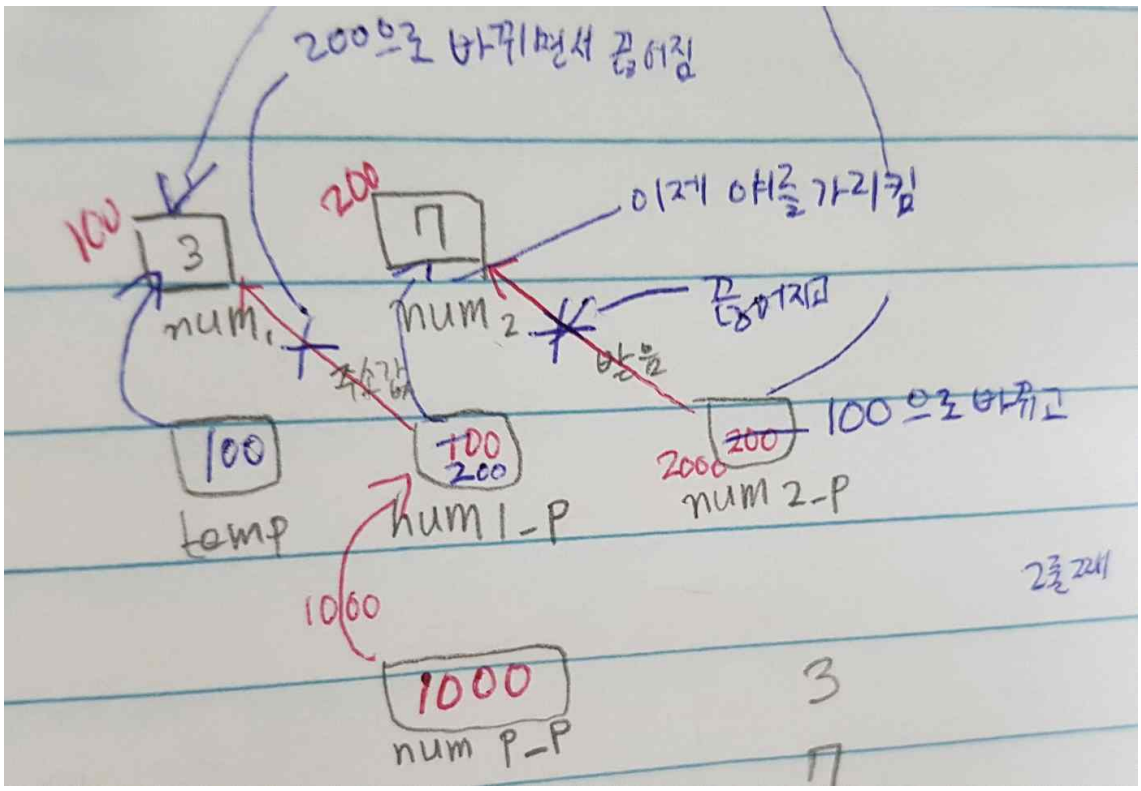
```
 1 #include<stdio.h>
 2
 3 int  main(void){
 4        int  num1=3,  num2=7;
 5        int  *temp=NULL;
 6        int  *num1_p=&num1;
 7        int  *num2_p=&num2;
 8        int  **num_p_p=&num1_p;
 9
10
11        printf("*num1_p=%d₩n",*num1_p);
12        printf("*num2_p=%d₩n",*num2_p);
13
14        temp=*num_p_p;
15        *num_p_p=num2_p;
16        num2_p=temp;
17
18        printf("*num1_p=%d₩n",*num1_p);
19        printf("*num2_p=%d₩n",*num2_p);
20
```

```
21          return 0;
22  }
```

결과값 :
*num1_p=3
*num2_p=7
*num1_p=7
*num2_p=3

이중포인터를 사용하면 재귀호출을 안쓰고도 트리를 구현할수있다
원격으로 두개를 바꿔버릴수있음

왜 3 7 7 3이 나온지 원인을 파악



예제소스 pointer7.c

```
1  #include<stdio.h>
2  int main(void)
3       {
4            int i, j, n1, n2, n3;
5            int a[2][2] = {{10,20},{30,40}};
6            int *arr_ptr[3]={&n1,&n2,&n3}; //포인터를 저장하는주소, 포인터배열
```

```
 7              int(*p)[2]=a;
 8
 9              for(i=0; i<3; i++)
10                      *arr_ptr[i]=i;
11              for(i=0; i<3; i++)
12                      printf("n%d=%d₩n",i,*arr_ptr[i]);
13
14              for(i=0; i<2; i++)
15                      printf("p[%d]=%d₩n",i,*p[i]);
16
17              return 0;
18      }
```

결과값:

```
n0=0
n1=1
n2=2
p[0]=10
p[1]=30
```

```
 1 #include<stdio.h>
 2 int main(void)
 3      {
 4              int i, j, n1, n2, n3;
 5              int a[2][2] = {{10,20},{30,40}};
 6              int *arr_ptr[3]={&n1,&n2,&n3}; //포인터를 저장하는주소, 포인터배열
 7              int(*p)[3]=a;
 8
 9              for(i=0; i<3; i++)
10                      *arr_ptr[i]=i;
11              for(i=0; i<3; i++)
12                      printf("n%d=%d₩n",i,*arr_ptr[i]);
13
14              for(i=0; i<2; i++)
15                      printf("p[%d]=%d₩n",i,*p[i]);
16
17              return 0;
18      }
```

결과값 :
n0=0
n1=1
n2=2
p[0]=10
p[1]=40

1. 배열에 문자열을 입력 받고,
각 배열 요소가 짝수인 경우만을 출력하는 함수를 작성하라.

```c
#include<stido.h>

int main(void)
{
    int s[100];
    printf("문자열 입력: ");
    scanf("%d", s);

    printf("%d\n", s);

    return 0;
}
```

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

4. 위의 숫자 3, 77, 10, 7, 4, 9, 1, 8, 21, 33에서
홀수 번째 요소의 합과 짝수 번째 요소의 합을 곱하시오.

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

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.

```c
#include<stdio.h>

int main(void)
{
        char *ptr;
        char num =2004016;

        ptr = &num;

        printf("%p\n",*ptr);

        return 0;
}
```

4. 우리는 예제에서 주소값을 교환하여 값을 변경하는 것을 해보았다.

그렇다면 변수 3개를 놓고, 이것에 대해서 무한 Loop를 돌면서 저글링을 해보자!

```c
#include <stdio.h>

int main()
{
        int num1=1;
        int num2=2;
        int num3=3;
    for (;;)
    {
        printf("%d\n",num1);
        printf("%d\n",num2);
        printf("%d\n",num3);
    }

    return 0;
}
```

삼각형의 넓이 구하는 문제

case 1) 밑변, 높이

```c
#include<stdio.h>

//삼각형의 넓이 공식 : 넓이 = ½(밑변의 길이x높이)

int main(void)
{

float num1;

float num2;

printf("삼각형의 밑변의 길이와 높이를 입력하세요: ");
scanf("%f %f",&num1, &num2);

printf("%f*%f/2=%fcm²\n",num1,num2,num1*num2/2);


return 0;
}
```

case 2) 밑변, 밑변과 다른 변이 이루는 각도

* 제공된 교재의 279 페이지 퀴즈들 풀어보기

```c
#include<stdio.h>
int main(void)
{
int i;
int j=0.4;

for(i=1000000; i<5; i++)
{

printf("%d+(%d*0.4)=%d\n",i,j,i+(i*j));

}


return 0;
```