배열을 왜 쓰나 , 편의를 위해

배열 = 주소

[]사이의 수를 index(인덱스)라고 하며
배열의 첫 번째 요소는 [0]에서 시작한다.
100개의 요소를 원하면 int number[100]
그러나 0부터 시작하므로 마지막은 [99]이다

string인 문자열 “I’m Marth Kim”은 변경이 불가능
char형 배열은 내부 데이터 변경이 가능하다
마지막 data에 Null Character가 필요함

Null Character는 무엇인가 ?
NULL 문자는 문자열의 마지막을 의미
배열에 값을 1개씩 직접 설정할 경우 ‘\0’으로
어느 부분이 배열의 마지막인지를 명시해주는 것이 좋음

262p,

문자열

```
#include<stdio.h>

int main(void)
{
char str1[5] = “AAA”;    // 5칸에,  A A A \0 0 이들어감 \0 이 끝을 내주니 AAA만나옴
char str2[ ] = “BBB”;      //  4칸에 B B B \0 (\0이 자동 들어감)
char str3[ ] = {‘A’, ‘B’, ‘C’}; // 4칸에 A B C \0
char str4[ ] = {‘A’, ‘B’, ‘C’, ‘\0’};  // 4칸에
```

```
printf(“str1 = %s\n", str1 );
printf(“str2 = %s\n", str2 );
printf(“str3 = %s\n", str3 );
printf(“str4 = %s\n", str4);



str1[0] = ‘E’;
str2[1] = ‘H’;
printf(“str1 = %s\n”, str1);
}
return 0;
```

2중배열

```
#include <stdio.h>
int main(void)
{
int arr[2][2] = {{10, 20}, {30, 40}};
int i, j;
for(i =0; i < 2; i++)
{
for( j = 0; j < 2; j++)
{
printf(“arr[%d][%d] = %d\n”, i, j, arr[i][j]);
}
}
}
return 0;
```

3중 배열

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

```
int main(void)
{
int arr[3][3][3];
int i, j, k, num = 1;
}
for(i = 0; i < 3; i++)
{
for( j = 0; j < 3; j++)
{
for(k = 0; k < 3; k++)
{
arr[i][j][k] = num++;
printf("arr[%d][%d][%d] = %d\n", i, j, k, arr[i][j][k]);
}
}
}
return 0;
```

함수의 인자로 배열을 전달하려면 ?
배열을 통째로 전달 할 수 없음
함수 호출시 배열의 주소값을 전달함
함수 자체에서 배열의 index를 생략하고 []만 전달해도됨
되도록이면 index를 모두 정확하게 적는것이 편함
2차원, 3차원 배열도 모두 동일한 방식임

```
#include <stdio.h>
void arr_print(int arr[])
{
int i;
for(i = 0; i < 3; i++)
printf("%4d", arr[i]);
printf("\n");
}
int main(void)                          //배열의 이름을 보내 (주소 를 보내) 주소를 알고 있다 함수 어디서든지
                                        //변수를 조정할수있다,,,,,,,,,,,,,,,,,,
{
int arr[3] = {3, 33, 333};    //arr[3]은 main ()함수내의 지역 변수 arr_print로 위 배열의 주소가 전달됨
arr_print(arr);        // 즉 arr_print 함수에서는 main()함수의 지역변수인 arr[3]에 접근하
```

여 값을 조정할수 있음,
return 0;
}

포인터는 무엇일까 ?

주소를 저장 할 수 있는 변수

조금 더 엄밀하게 주소를 저장할 수 있는 변수
무언가를 가르키는 녀석
Pointer의 크기는 HW가 몇 bit를 지원하는냐에 따름
그런데 왜 HW에 따라 달라질까 ?

Pointer는 이 Memory의 어떤 주소값이든 접근할 수 있어야함
고로 Pointer의 크기는 4byte이다

왜냐면 요

32bit(4byte)를 가정하고
최상위 메모리 주소 = 0xffffffff 고,
최하위 메모리 주소 = 0x0 이라고 하면 이를 접근하기 위해 4byte크기가 필요

포인터 선언 방법

int num =7;

int* : 인트형의 주소를 저장할 수 있는 타입,

이 자체가 데이터 타입

int* p : 인트형의 주소를 저장 할 수 있는 변수 p

자료형 없이 변수명 앞에 '*'만 붙은 경우
해당 변수가 가르키는 것의 값을 의미함

```
int main(void){
int *ptr;
printf("ptr = %p\n", ptr);
printf("ptr value = %d\n", *ptr);
*ptr = 27;
}
printf("ptr value = %d\n", *ptr);
return 0
```

```
ptr = (nil)
```

NULL Pointer는 무엇인가 ?
지금 당장 가르킬 녀석이 없을 경우 사용
위 예제처럼 엉뚱한 값을 가지지 않도록 초기화하기 위함

Segmentation fault (core dumped) 접근하면 안될곳에 접근,

nil =값이상해

```
 #include<stdio.h>

int main(void){

        int num = 3;

        *(&num) +=30;
          printf("num = %d\n", num);
        return 0;

}
```

```
num = 33
```

↙[[[0x1000]]]p

[[[3]]] num &num =0x1000 *(&num) =3 => [33]

0x1000주소라고하면 0x1000주소에 있는 3이라는 변수를 가져온다,

결국 num += 30;

이라는 말과 같음,,

* Segmentation Fault 가 나는 이유?

우리가 기계어를 보면서 살펴봤던 주소값들이 사실은 전부 가짜 주소라고 말했었따
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 = 4GB의 가상 메모리 공간을 가짐
여기서 1:3으로 1을 커널이 3을 유저가 가져간다.
1은 시스템(HW , CPU, SW 각종 자원들)에 관련된 중요한 함수 루틴과 정보들을 관리하게 된다.
3은 사용자들이 사용하는 보들로
문제가 생겨도 그다지 치명적이지 않은 정보들로 구성됨
64비트 시스템은 1:1로 2^63 승에 해당하는 가상메모리를 각각 가진다.
문제는 변수에 초기화하지 않았을 경우 가지게 되는 쓰레기값이 0xCCCC.....CCCC로 구성됨이다

다
(맨 바닥이 0 이라고 생각하면 맨위는 ffffffff 중간은 888888888 (16진수))

32비트의 경우에도 1:3 경계인 0xC00000000000을 넘어가게됨
64비트의 경우엔 시작이 c이므로
이미 1:1 경계를 한참 넘어

그러므로 접근하면 안되는 메모리 영역에 접근하였기에 엄밀하게는 Page Fault(물리 메모리 할당되지 않음)
발생하게 되고 원래는 Interrupt가 발생해서 Kernel 이 Page Handler(페이지 제이기)(유저쪽에서 잘못된 정보처리 하다가 오류가 났다)가 동작해서
가상 메모리에 대한 Paging 처리를 해주고
실제 물리 메모리를 할당해주는데
문제는 이것이 User 쪽에서 들어온 요청이므로
Kernel 쪽에서 강제로 기각해버리면서
Segmentation Fault 가 발생하는 것이다

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

'&'은 무엇인가 ?
'&' 뒤에 붙은 변수명의 주소값을 가져옴
상수형 Pointer

Pointer to Pointer
Pointer에 대한 Pointer는 ?
Pointer도 Stack에 할당되는 지역변수
즉, Pointer에 대한 주소값도 존재함
얻는법은 동일함('&'를 이용)
그 주소값은 Pointer에 대한 Pointer로 '**' 형태임

3 100 200

num p pp

int num =3 ;

int *p => #

int**pp => &p;

p(num)

p(*p)

p(**pp)

```
int main(void){
int num = 7;
int *ptr1 = &num;
int **ptr2 = &ptr1;
printf("ptr1 = %p\n", ptr1);
printf("ptr2 = %p\n", ptr2);
printf("ptr2가 가리키는것(ptr1)이 가리키는 것은 ? %d\n", **ptr2);
}
return 0;
```

이중 포인터

```
int main(void){
int num1 = 3, num2 = 7;
int *temp = NULL
int *num1_p = &num1;
int *num2_p = &num2;
int **num_p_p = &num1_p;
printf("*num1_p = %d\n", *num1_p);
printf("*num2_p = %d\n", *num2_p);
temp = *num_p_p;
*num_p_p = num2_p;
```

```
num2_p = temp;
printf(“*num1_p = %d\n”, *num1_p);
printf(“*num2_p = %d\n”, *num2_p);
}
return 0;
```

```
num1_p = 3
num2_p = 7
*num1_p = 7
*num2_p = 3
```

```
#include <stdio.h>
  2
  3 int main(void)
  4 {
  5
  6
  7           int num=3 ;
  8           int *p =&num;
  9           int **pp = &p;
 10           int ***ppp =&pp;
 11
 12
 13           printf("num =%d\n" , num);
 14           printf("*p = %d\n" , *p);
 15           printf("**pp = %d\n" , **pp);
 16           printf("***ppp = %d\n" , ***ppp);
 17           return 0 ;
 18
 19 }
```

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

int *p[2] int*데이터타입이고
int(*p)[2] int*p 자체를 두개(= int*[2]p) ,, = int(*)[2] 인트형 4byte 두개짜리 포인터 8byte형 포인터 p