

Pointer

C 중급 세미나 - 김혜윤 -

Contents

1. 포인터

- 주소 / 연산자 *, & / 포인터 연산

2. 배열과 포인터

- 배열과 포인터의 관계 / 배열 이름 / 연산자 []

3. 배열 포인터 vs. 포인터 배열

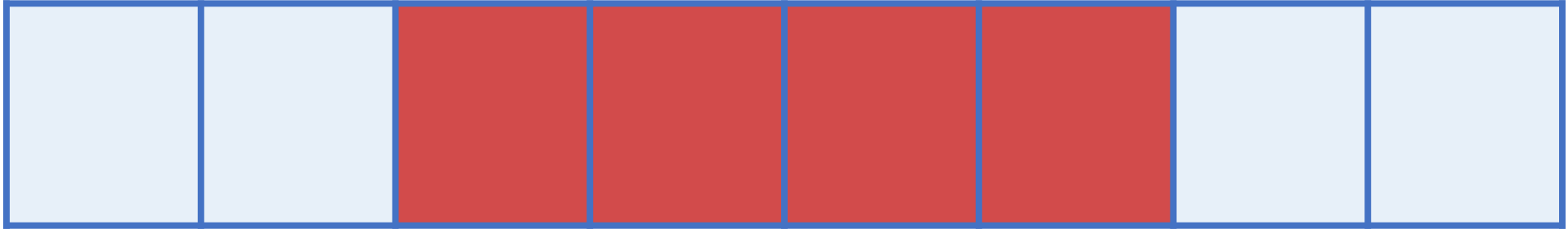
- 포인터 배열
- 이중 포인터
- 배열 포인터 / 2차원 배열

포인터

주소

```
int a = 1; // 4 바이트 (int)
```

4바이트로 주소 할당



0000007C0510F6D4

64비트 운영체제에서
포인터는 8바이트

연산자 *, &

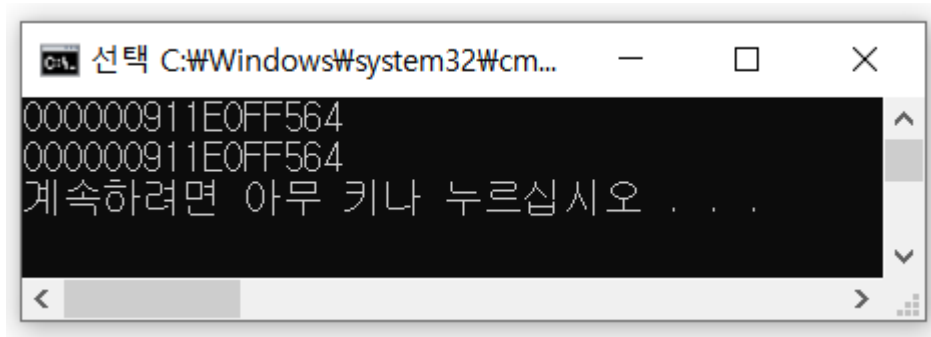
- 위 : p만 포인터
- 아래 : p, a, b 포인터

- 포인터 : 주소값을 저장하는 변수
- 자료형 *포인터이름 : 포인터 변수 선언
- 포인터 = &변수
: 변수의 주소값 대입
- %p: 16진수로 출력

```
int *p;  
int* p;
```

```
int *p, a, b;  
int* p, a, b;
```

```
#include <stdio.h>  
  
int main()  
{  
    int *ptr;    // 포인터 변수 선언  
    int n = 10;  
  
    ptr = &n;    // n 주소값 ptr에 대입  
  
    printf("%p\n", &n);    // 변수 n의 주소값  
    printf("%p\n", ptr);    // ptr에 저장된 주소값  
  
    return 0;  
}
```



연산자 * (역참조)

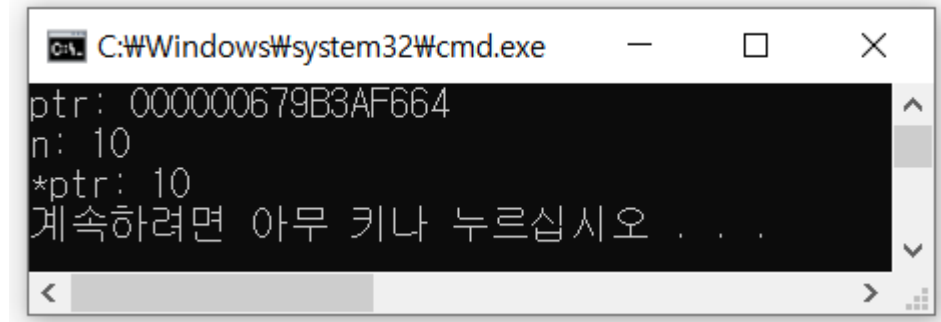
- *(포인터) : 포인터에 저장된 주소에 저장된 값을 역참조

```
#include <stdio.h>
int main()
{
    int *ptr;    // 포인터 변수 선언
    int n = 10;

    ptr = &n;    // n 주소값 ptr에 대입

    printf("ptr: %p\n", ptr);    // ptr에 저장된 주소값
    printf("n: %d\n", n);        // 변수 n에 저장된 값 (10)
    printf("*ptr: %d\n", *ptr);  // ptr에 저장된 주소에 저장된 값

    return 0;
}
```



```
C:\Windows\system32\cmd.exe
ptr: 000000679B3AF664
n: 10
*ptr: 10
계속하려면 아무 키나 누르십시오 . . .
```



연산자 * (포인터 선언 vs. 역참조)

- 자료형 *포인터이름 : 포인터 변수 선언
- *(포인터) : 포인터에 저장된 주소에 저장된 값을 역참조

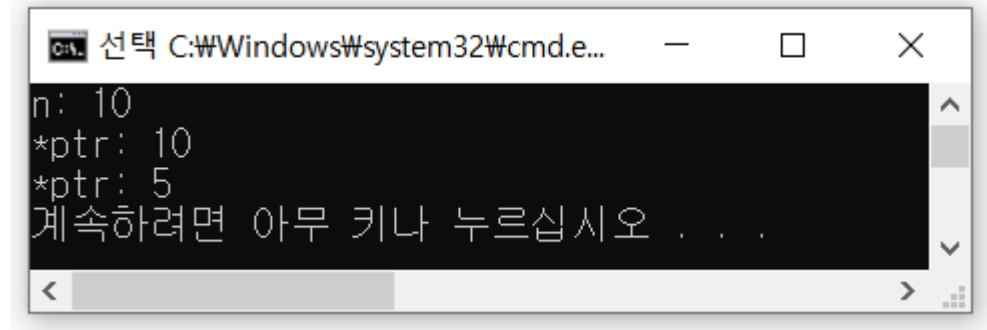
```
#include <stdio.h>
int main()
{
    int *ptr;    // 포인터 변수 선언
    int n = 10;

    ptr = &n;    // n 주소값 ptr에 대입

    printf("n: %d\n", n);        // 변수 n에 저장된 값 (10)
    printf("*ptr: %d\n", *ptr);  // ptr에 저장된 주소에 저장된 값

    *ptr = 5;
    printf("*ptr: %d\n", *ptr);  // ptr에 저장된 주소에 저장된 값

    return 0;
}
```



```
선택 C:\Windows\system32\cmd.e...
n: 10
*ptr: 10
*ptr: 5
계속하려면 아무 키나 누르십시오 . . .
```

포인터 연산

- 포인터에 저장된 자료형만큼 곱해서 연산 (뺄셈도 가능)

```
#include <stdio.h>
int main()
{
    int *ptr1;    // 포인터 변수 선언
    char *ptr2;   // 포인터 변수 선언
    int a = 10;
    char b = 'b';

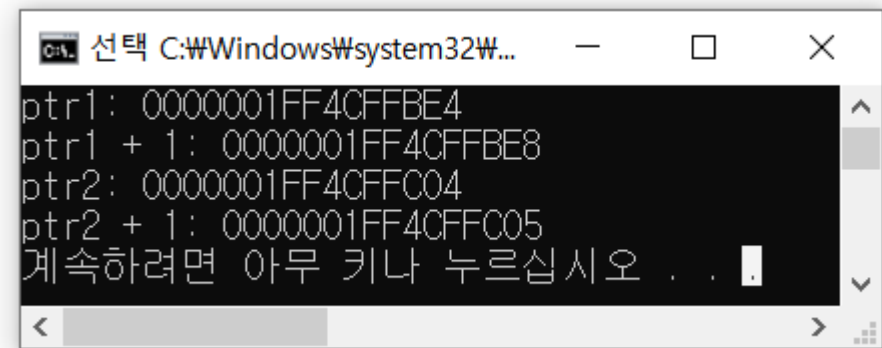
    ptr1 = &a;    // a 주소값 ptr1에 대입
    ptr2 = &b;    // b 주소값 ptr2에 대입

    printf("ptr1: %p\n", ptr1);        // ptr1에 저장된 주소값
    printf("ptr1 + 1: %p\n", ptr1 + 1); // ptr1에 저장된 주소값 + 1

    printf("ptr2: %p\n", ptr2);        // ptr2에 저장된 주소값
    printf("ptr2 + 1: %p\n", ptr2 + 1); // ptr2에 저장된 주소값 + 1

    return 0;
}
```

int 형: 4바이트
char 형: 1바이트

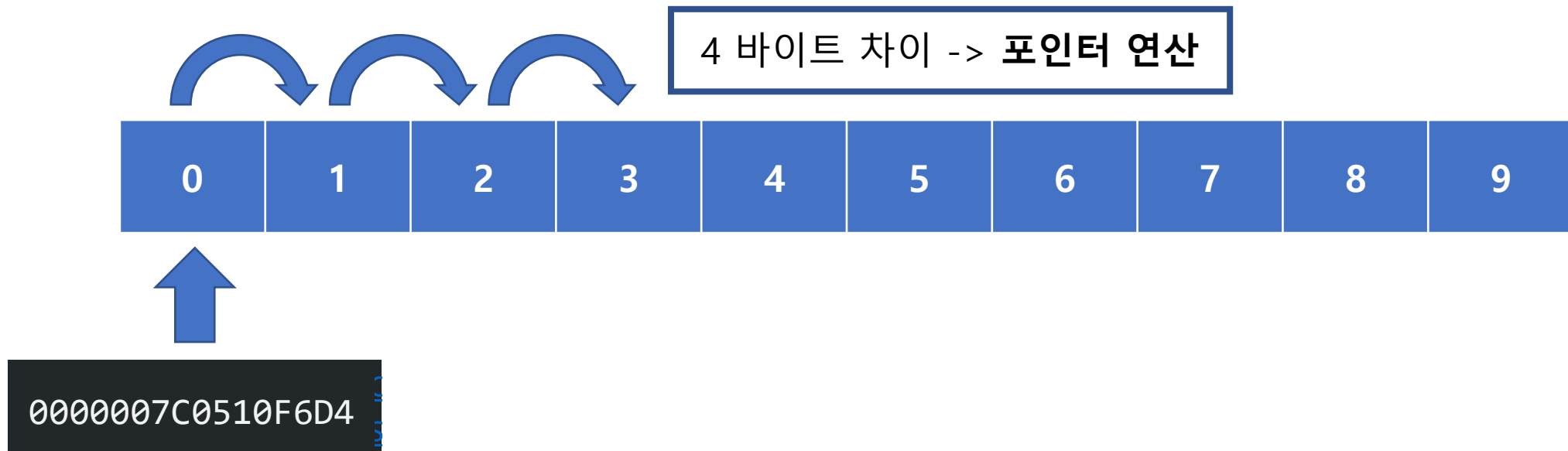


```
선택 C:\Windows\system32\W...
ptr1: 0000001FF4CFFBE4
ptr1 + 1: 0000001FF4CFFBE8
ptr2: 0000001FF4CFFC04
ptr2 + 1: 0000001FF4CFFC05
계속하려면 아무 키나 누르십시오...
```


배열과 포인터

배열과 포인터의 관계

```
int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```



배열과 포인터의 관계

포인터 연산 -> 배열 주소 접근

```
#include <stdio.h>

int main() {

    int array[5] = {0, 1, 2, 3, 4};
    int *p = &array[0];

    for (int i = 0; i < 5; i++) {
        printf("array[%d] : %p\n", i, &array[i]);
        printf("array + %d : %p\n", i, p + i);
    }

    return 0;
}
```

1	array[0]	: 0x7ffd962b84e0
2	array + 0	: 0x7ffd962b84e0
3	array[1]	: 0x7ffd962b84e4
4	array + 1	: 0x7ffd962b84e4
5	array[2]	: 0x7ffd962b84e8
6	array + 2	: 0x7ffd962b84e8
7	array[3]	: 0x7ffd962b84ec
8	array + 3	: 0x7ffd962b84ec
9	array[4]	: 0x7ffd962b84f0
10	array + 4	: 0x7ffd962b84f0

배열과 포인터의 관계

포인터 연산 -> 배열값 접근

```
#include <stdio.h>

int main() {

    int array[5] = {0, 1, 2, 3, 4};
    int *p = &array[0];

    for (int i = 0; i < 5; i++) {
        printf("array[%d] : %d\n", i, array[i]);
        printf("array + %d : %d\n", i, *(p + i));
    }

    return 0;
}
```

1	array[0]	:	0
2	array + 0	:	0
3	array[1]	:	1
4	array + 1	:	1
5	array[2]	:	2
6	array + 2	:	2
7	array[3]	:	3
8	array + 3	:	3
9	array[4]	:	4
10	array + 4	:	4

배열 이름

```
#include <stdio.h>

int main() {

    int array[5] = {0, 1, 2, 3, 4};
    int *p = &array[0];

    printf("array[0] : %p\n", p);
    printf("array      : %p\n", array);

    return 0;
}
```

array = array[0] 주소값

-> 배열 이름이 첫번째 원소를
가리키는 포인터??

```
array[0] : 0x7ffcabeca900
array     : 0x7ffcabeca900
```

배열 이름 \neq 포인터

```
#include <stdio.h>

int main() {

    int array[5] = {0, 1, 2, 3, 4};
    int *p = &array[0];

    printf("sizeof(array[0]) : %d\n", sizeof(p));
    printf("sizeof(array)      : %d\n", sizeof(array));

    return 0;

}
```

- sizeof : 크기 알려주는 연산자 (바이트)
- &array[0]: 주소 -> 8바이트
- array: 배열 이름
배열 크기($5 \times 4 = 20$)
- sizeof, &와 사용될 때를 제외
하면 첫번째 원소를 가리키는
포인터로 타입 변환됨
- 그냥 array -> &array[0]
- array에 다른 주소값 대입 X

```
1  sizeof(&array[0]) : 8
2  sizeof(array)      : 20
```

연산자 [] : $array[i] = *(array + i)$


```
#include <stdio.h>

int main() {

    int array[5] = {0, 1, 2, 3, 4};

    for (int i = 0; i < 5; i++) {
        printf("array[%d] : %d\n", i, array[i]);
        printf("array + %d : %d\n", i, *(array + i));
    }

    return 0;
}
```



`&array[0]`

1	<code>array[0]</code>	: 0
2	<code>array + 0</code>	: 0
3	<code>array[1]</code>	: 1
4	<code>array + 1</code>	: 1
5	<code>array[2]</code>	: 2
6	<code>array + 2</code>	: 2
7	<code>array[3]</code>	: 3
8	<code>array + 3</code>	: 3
9	<code>array[4]</code>	: 4
10	<code>array + 4</code>	: 4

연산자 [] p[i]로도 배열 접근 가능

```
#include <stdio.h>

int main() {

    int array[5] = {0, 1, 2, 3, 4};
    int *p = &array[0];

    for (int i = 0; i < 5; i++) {
        printf("array[%d] : %d\n", i, array[i]);
        printf("array + %d : %d\n", i, p[i]);
    }

    return 0;
}
```

*(array + i), *(p + i)와 같음

1	array[0]	:	0
2	array + 0	:	0
3	array[1]	:	1
4	array + 1	:	1
5	array[2]	:	2
6	array + 2	:	2
7	array[3]	:	3
8	array + 3	:	3
9	array[4]	:	4
10	array + 4	:	4

배열 포인터 vs. 포인터 배열

포인터(를 담는) 배열

```
#include <stdio.h>

int main() {

    int a = 1, b = 2, c = 3;
    int *pa = &a;
    int *pb = &b;
    int *pc = &c;
    int *ppp[3] = {pa, pb, pc};

    for (int i = 0; i < 3; i++) {
        printf("ppp[%d] : %p\n", i, ppp[i]); // ppp배열의 포인터 값
        printf("*ppp[%d] : %d\n", i, *ppp[i]); // ppp배열의 포인터 역참조
    }

    return 0;
}
```

우선순위	연산자(연산기호)
1	() [] -> .
2	! ~ ++ -- * & sizeof() (자료형)
3	* / %



*ppp[i]에서 *보다
[]먼저 연산

1	ppp[0] : 0x7ffd862b44a4
2	*ppp[0] : 1
3	ppp[1] : 0x7ffd862b44a8
4	*ppp[1] : 2
5	ppp[2] : 0x7ffd862b44ac
6	*ppp[2] : 3

이중 포인터

```
#include <stdio.h>

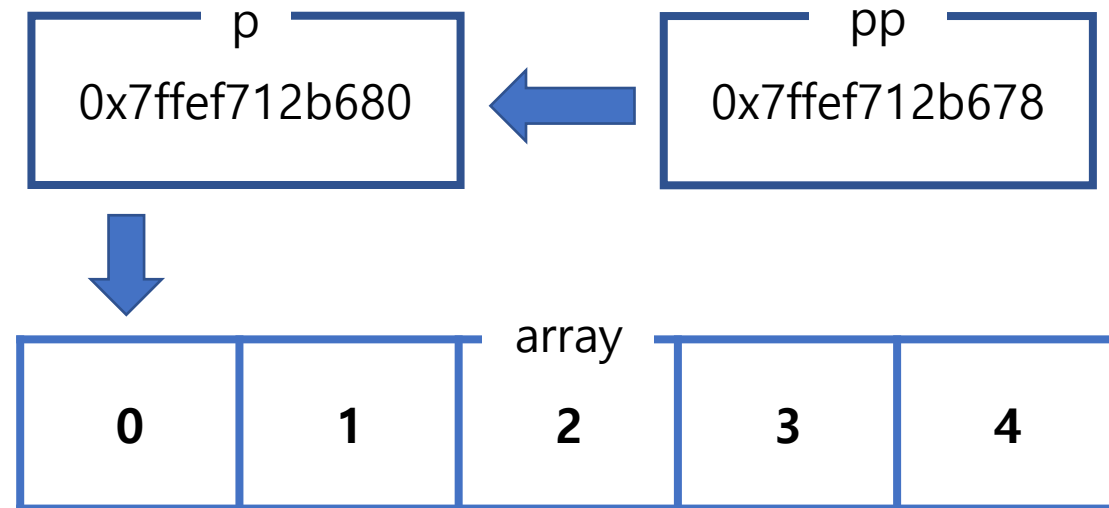
int main() {

    int array[5] = {0, 1, 2, 3, 4};
    int *p = &array[0];
    int **pp = &p;

    printf("*p : %d\n", *p);    // array[0]
    printf("p : %p\n", p);      // array[0] 주소
    printf("*pp : %p\n", *pp);  // p 값 = array[0] 주소
    printf("pp : %p\n", pp);    // p 주소

    return 0;
}
```

1	*p	:	0
2	p	:	0x7ffef712b680
3	*pp	:	0x7ffef712b680
4	pp	:	0x7ffef712b678



배열(을 가리키는) 포인터

```
#include <stdio.h>

int main() {

    int array[5] = {0, 1, 2, 3, 4};
    int (*parray)[5] = &array;          // 크기 5 배열을 가리키는 포인터

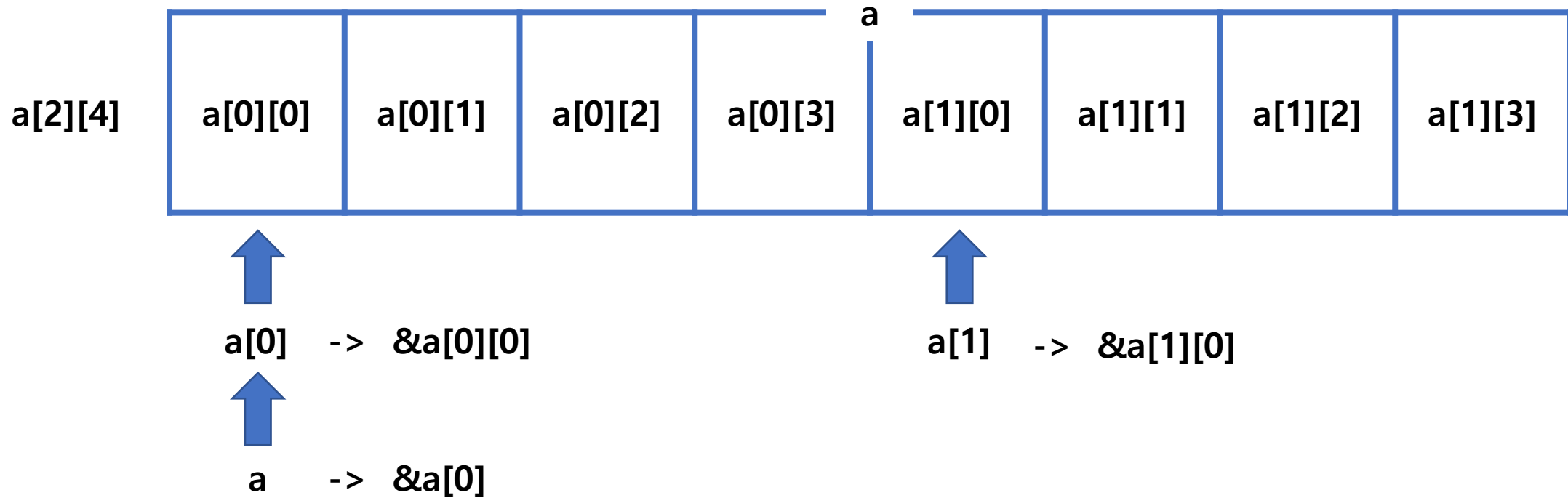
    printf("array[3] : %d \n", array[3]);
    printf("parray[3] : %d \n", (*parray)[3]);

    return 0;
}
```

- &array : &연산자와 사용됨
-> array 배열 나타냄
(array[0]로 변환 X)
- int (*parray)[5] 에서 괄호 필수

1	array[3]	:	3
2	parray[3]	:	3

2차원 배열



2차원 배열

```
#include <stdio.h>

int main() {

    int array[2][4] = {0,};

    printf("array      : %d\n", sizeof(array));
    printf("array[0]    : %d\n", sizeof(array[0]));
    printf("array[0][0] : %d\n", sizeof(array[0][0]));

    return 0;
}
```

1	array	:	32
2	array[0]	:	16
3	array[0][0]	:	4

2차원 배열 이름을 담은 포인터

```
#include <stdio.h>

int main() {
    int array[2][4] = {0,};

    printf("array          : %p\n", array);
    printf("array + 1      : %p\n", array + 1);
    printf("array[0]         : %p\n", array[0]);
    printf("array[0] + 1 : %p\n", array[0] + 1);

    return 0;
}
```

- array + 1
array[0] 크기만큼 커짐
-> array를 담은 포인터는
int [4]를 가리키는 포인터

1	array	:	0x7ffd6c3fb1d0
2	array + 1	:	0x7ffd6c3fb1e0
3	array[0]	:	0x7ffd6c3fb1d0
4	array[0] + 1	:	0x7ffd6c3fb1d4

2차원 배열 이름을 담은 포인터

```
#include <stdio.h>

int main() {
    int array[2][4] = {0,};
    int (*p)[4] = array; // int **p = array X

    printf("array          : %p\n", array);
    printf("p + 1          : %p\n", p + 1);
    printf("array[0] + 1 : %p\n", array[0]);
    printf("p + 1          : %p\n", p[0] + 1);

    return 0;
}
```

1	array	: 0x7ffccbcf3cc0
2	p + 1	: 0x7ffccbcf3cd0
3	array[0] + 1	: 0x7ffccbcf3cc0
4	p + 1	: 0x7ffccbcf3cc4