

2차시 - 포인터(2)

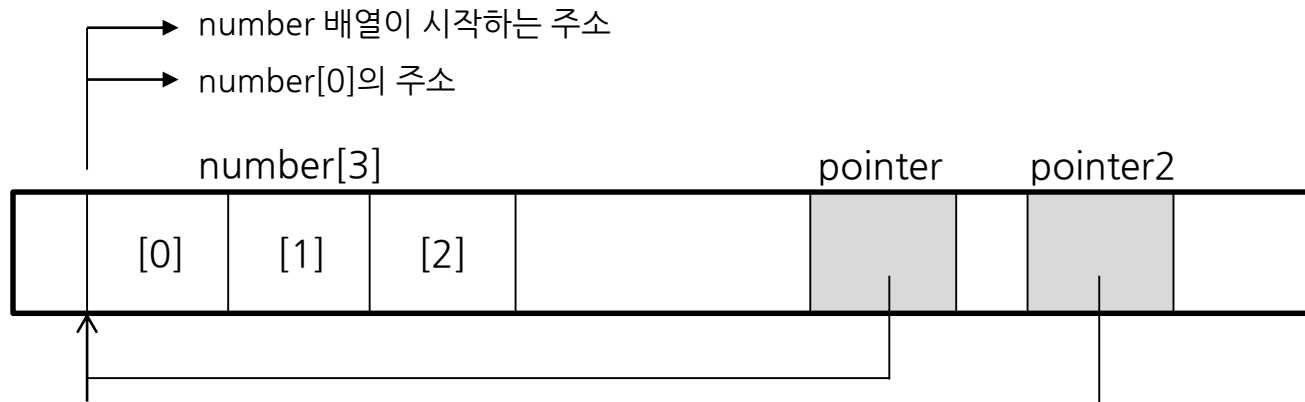
포인터를 통한 배열 참조

◆ 배열과 포인터

- 배열의 이름만 사용하면 **포인터 상수**가 된다. (상수라는 점에 유의)

```
int  number[3];  
int  *pointer = &number[0];  
int  *pointer2 = number ; // = &number[0] ;  
                배열의 주소 (=포인터)
```

number = 10; (X) 상수이기 때문에 값으로만 사용된다.



포인터를 통한 배열 참조

◆ 배열과 포인터

- 배열의 이름만 사용하면 포인터 상수가 된다.

```
int *pointer2 = number ; // = &number[0] ;  
                배열의 주소 (=포인터)
```

- number를 단독으로 사용할 때의 자료형은?
- int *형인 pointer2 변수에 number 값을 저장할 수 있을까?(int *, int [3])
- number는 포인터 상수이지만 &을 붙일 수 있다.
 - int (*ptr)[3] = &number ;
// number 의 자료형에 &을 붙였다.
 - &number[0] == number == &number (자료형은 다르지만 값은 같다)

포인터 연산

◆ 포인터(주소)도 숫자이기 때문에 연산이 가능하다.

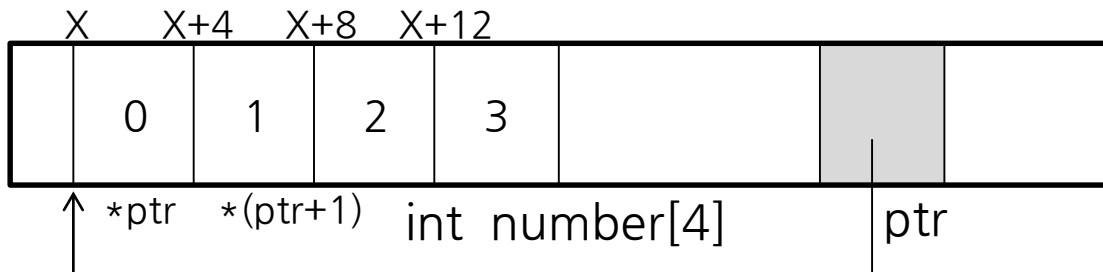
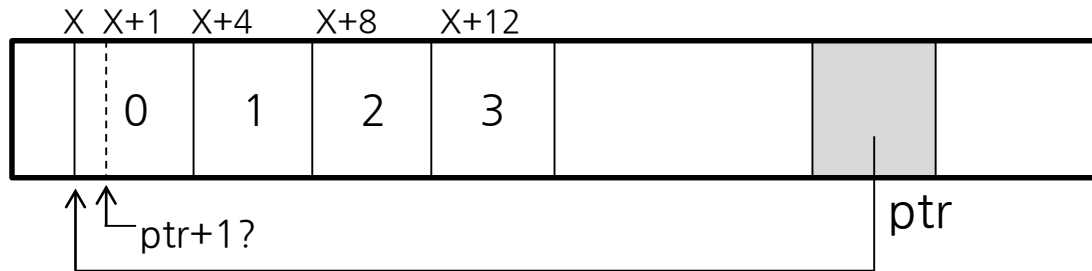
```
int *a = &i;  
a++; a = a + 3;
```

– 주로 배열에 사용한다.(데이터가 연이어 있기 때문에 ++도 의미가 있다)

```
int number [ ] = { 0, 1, 2, 3 };  
int *ptr = &number[0];           // &number[0] = 1000 이라 가정하자  
  
printf("%p  %d\n", ptr, *ptr);  
ptr++;                           // ptr = ptr + 1;  
printf("%p  %d\n", ptr, *ptr);
```

포인터 연산

◆ 포인터 그림 그리고, 연산 결과



$$ptr = ptr + 1$$

$\underbrace{\hspace{1.5cm}}_{(int *)} \quad \underbrace{\hspace{1.5cm}}_{(int)1 \rightarrow sizeof(int) * 1}$

◆ 다른 자료형의 포인터 연산

– 결과는?

```
char *pc = 0;
double *pd = 0;
short **pps = 0;
float *pf = 0;
pc++;
pd += 5;
pps += 2;
pf = (char *) pf + 1;
printf("%x %x %x %x \n", pc, pd, pps, pf);
```

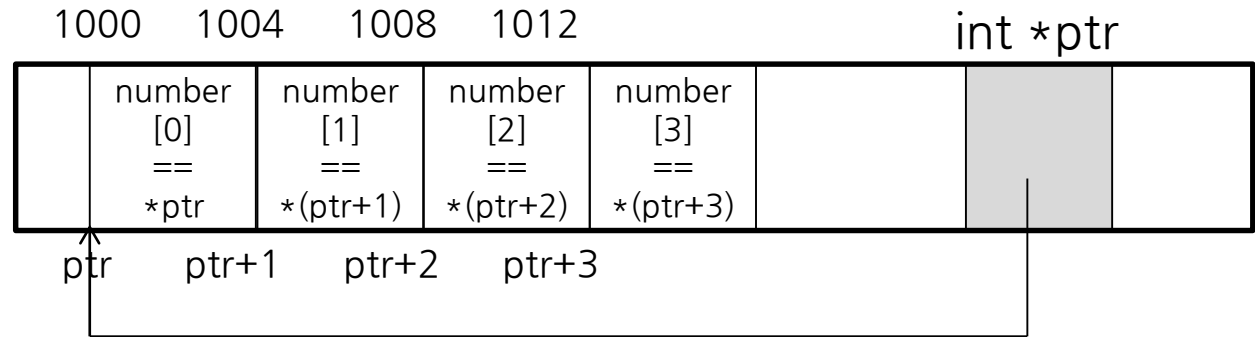
– pc ++ : pc = pc + sizeof(char) * 1

– pd +=5 : pd = pd + sizeof(double) * 5

– pps, pf ?

포인터를 통한 배열 참조

```
int p[10];  
int *ptr = &p[0];  
p[0] == *ptr;  
p[1] == *(ptr + 1);  
p[2] == *(ptr + 2);  
p[n] == *(ptr + n);
```

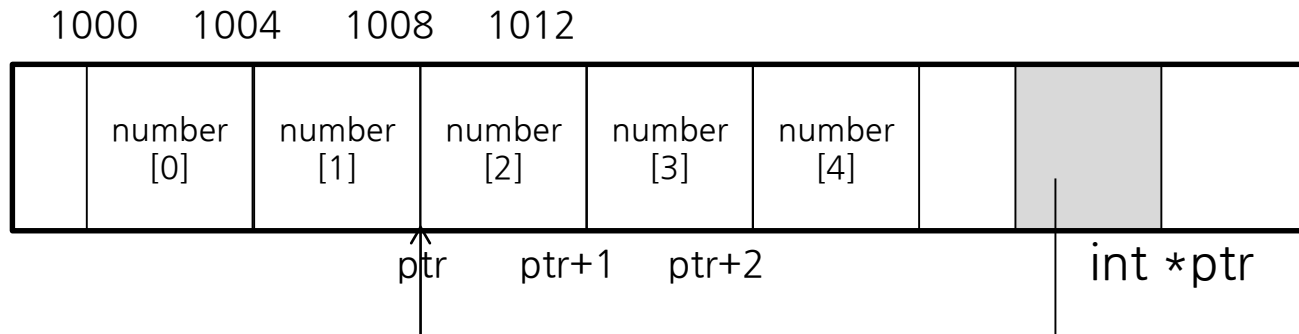


```
for(i=0; i<10; i++)  
    printf("%d \n", number[i]);  
  
for(i=0; i<10; i++)  
    printf("%d \n", *(number + i) );
```

포인터를 통한 배열 참조

◆ 포인터의 활용

- 배열의 기준점을 바꾸어 사용하거나(ptr = &number[2];)
- 배열의 일부분을 배열처럼 사용



- *ptr == number[2]
- *(ptr+1) == number[3]

포인터 연산

```
int number[5];  
int value;  
value = &number[2] - &number[0];    // 2 (8이 아니다)
```

◆ 포인터의 - 연산 결과

- 자료형은 int형
- 결과값은 sizeof(자료형)의 개수
- 포인터의 + 연산 : “포인터 + int형” 만 가능하며,
 - $\text{ptr} = \text{ptr} + 1$ 은 가능하나, $\text{ptr} = \text{ptr} + \text{ptr2}$ 는 불가능(ptr, ptr가 포인터)
- 포인터의 - 연산: “포인터 - 포인터”, “포인터 - int형”이 가능하다.
 - $\text{ptr} = \text{ptr} - 1$, $\text{ptr} = \text{ptr2} - \text{ptr1}$ 모두 가능
- 포인터의 *, / 연산은 불가능

컴퓨터가 보는 배열

◆ 배열이란 개발자(사람)를 위한 개념이다.

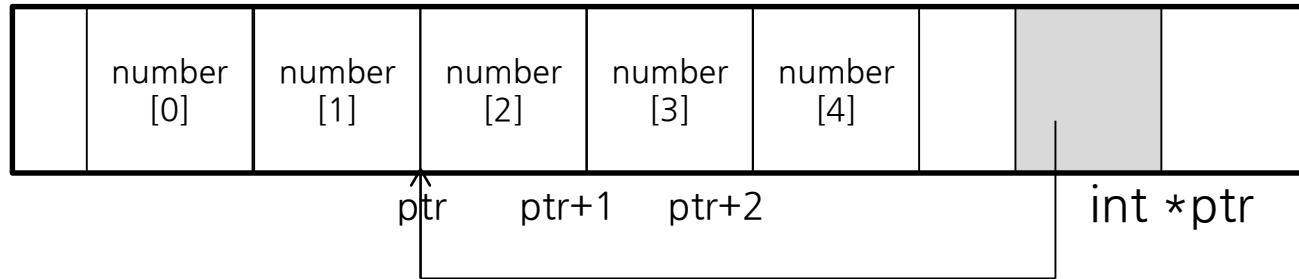
- 컴퓨터에게 배열은 없고 그냥 1차원 데이터다.
- 따라서 모든 배열 사용은 포인터로 바뀌어서 사용한다.
- $\text{array}[x] \rightarrow *(\text{array} + x)$
(사람용) 변환 (컴퓨터용)

◆ 그래서 다음 문장도 에러가 나지 않는다.

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

- $\text{array}[i] = *(\text{array} + i)$ 이고, $i[\text{array}] = *(i + \text{array})$ 로 같다.

배열과 포인터 자료형



◆ 1차원 배열의 구성 요소에 접근할 때에는 1차 포인터 변수를 사용한다.

– 1차원 배열과 1차 포인터는 상호 호환된다.

포인터를 통한 배열 참조(2차원)

```
int    number[2][3];  
int    *ptr = &number[0][0];  
number[0][0] == *ptr;  
number[0][1] == *(ptr + 1);  
number[0][2] == *(ptr + 2);  
number[1][0] == *(ptr + 3);  
number[1][0] == *(ptr + 1 * 3 + 0);
```

number

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]

사실은..



[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]
--------	--------	--------	--------	--------	--------

◆ 컴퓨터에게 2차원배열은 없다.

int number[x][y]; 에서

$\text{number}[a][b] == *(\text{number} + a * y + b);$

포인터를 통한 배열 참조

◆ 배열의 이름은 포인터 상수

1000

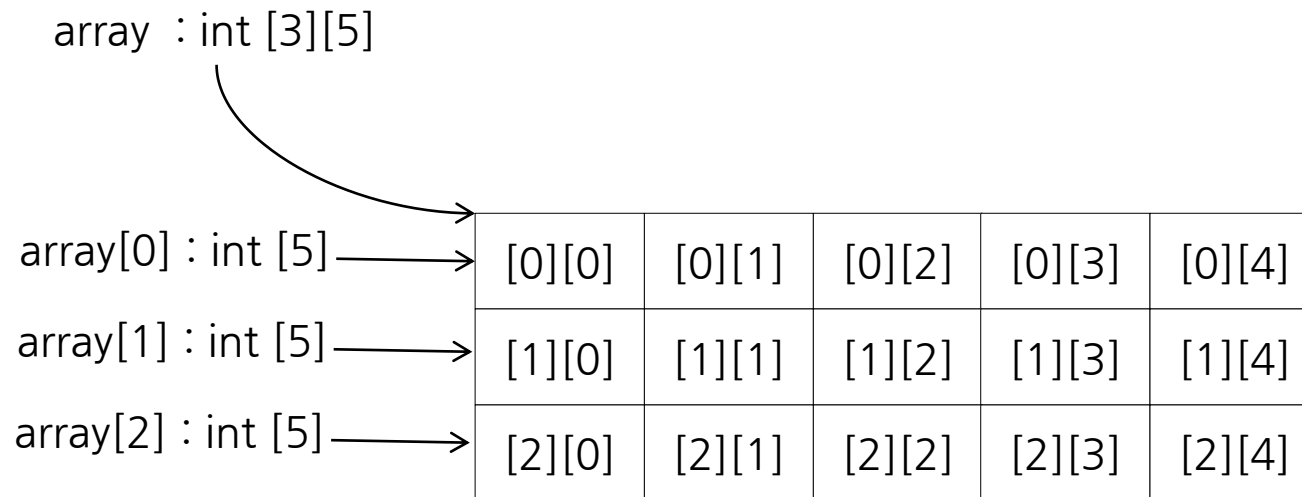
int array[3][5];

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

array[1][1] : int

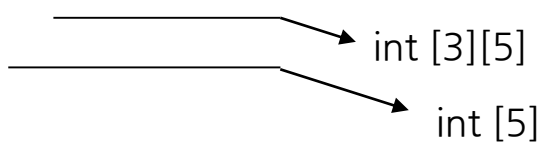
array 의 자료형과 값은?

array[0], array[1] 의 자료형과 값은?



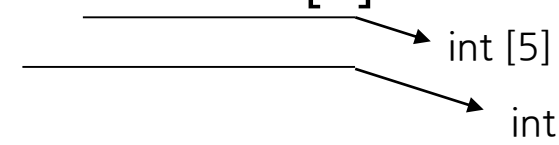
배열의 간접 참조

① * number



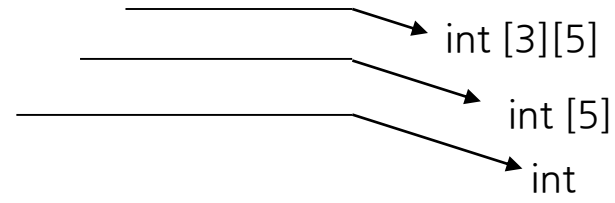
= (int [5]) *(1000) = number[0]

② * number[0]



= (int) *(number[0]) = (int) *(1000) = (int) number[0][0]

③ * * number



= (int) *(1000) = (int) number[0][0]

배열의 간접 참조는
차원을 하나 줄이는 것이다.

포인터를 통한 배열 참조

```
int  number[3][5] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
int  (*pointer)[5];
int  **doubleptr;
int  *singleptr;

pointer = number;
doubleptr = number;
singleptr = number[2];

printf("%d\n", **number);
printf("%d\n", **pointer);
printf("%d\n", *doubleptr);
printf("%d\n", *singleptr);
printf("%d\n", **doubleptr);
```

포인터를 통한 배열 참조

* doubleptr
_____ → int **
_____ → int *

= (int *) *(1000) = (int*) number[0][0]

* * doubleptr
_____ → int **
_____ → int *
_____ → int

= (int) *(*doubleptr) = (int) *(1) = 오류

포인터를 통한 배열 참조

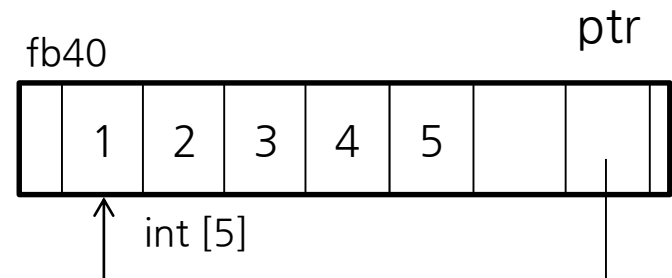
◆ 배열의 참조

- `int *ptr = &number[0][0];`
- `int (*ptr2)[5] = &number[0];`
- `int (*ptr3)[3][5] = &number;`
- &는 변수 앞에 붙지만, 배열 이름(포인터 상수) 앞에 붙을 수 있다.
- 이 때, 포인터 상수는 달라지지 않고 자료형만 1차 포인터형만큼 깊어진다.

직접 해보는 포인터 연구

◆ 간단한 프로그램을 작성

- 컴파일 및 프로시저 단위 실행(F10)
- 종료되기 전까지 F10으로 실행
- 조사식에서 원하는 포인터 수식 입력
- 그림을 그리면 이해가 쉽다.



조사식 1			▼ 🔍 ✕
이름	값	형식	
▶ array	0x00affb40 {0x00affb40 {1, 2, 3, 4, 5}, 0x00affb54 {6, 7, 8, ...}}	int[3][5]	
▶ *array	0x00affb40 {1, 2, 3, 4, 5}	int[5]	
▶ **array	1	int	
▶ pointer	0x00affb40 {1, 2, 3, 4, 5}	int[5] *	
▶ *pointer	0x00affb40 {1, 2, 3, 4, 5}	int[5]	
▶ **pointer	1	int	

자동 지역 스택 모듈 조사식 1

형식에 int [5] * 라고 표시되면 int [5]의 *(포인터)라는 의미이다.

포인터를 통한 배열 참조

◆ 배열의 이름은 포인터 상수

```
int number[3][5];
```

```
int (*ptr)[5] = number;
```

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

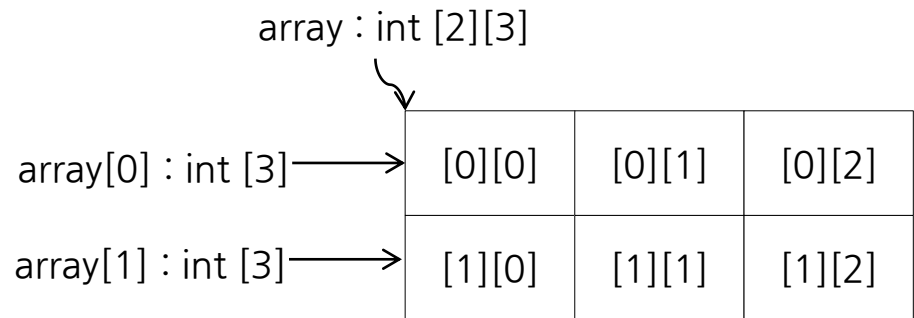
```
number[0][0] == **number == **ptr;
```

```
number[0][1] == *( number[0] + 1 ) == *( *number + 1 ) == * ( * ptr + 1 )
```

```
number[0][2] == *( number[0] + 2 ) == *( *number + 2 ) == * ( * ptr + 2 )
```

```
number[1][0] == *( number[1] + 0 ) == (*(number + 1) + 0 )  
                == *( *number + 5 * 1 + 0 )  
                == * ( * ( ptr + 1 ) + 0 )  
                == * ( * ptr + 5 )
```

2차원 포인터의 연산



$\ast(\text{array} + 1) == \text{array}[1]$

$\overline{\text{int [2][3]}} \quad \overline{\text{sizeof(int[3])} \ast 1}$

$\ast(\text{array}[1] + 1) == \text{array}[1][1]$

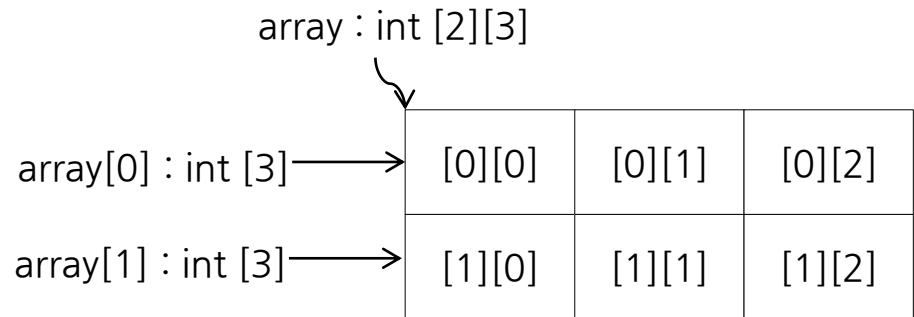
$\overline{\text{int [3] = int} \ast} \quad \overline{\text{sizeof(int)} \ast 1}$

$\ast(\ast(\text{array} + 1) + 1) == \text{array}[1][1]$

$\overline{\text{int [2][3]}} \quad \overline{\text{sizeof(int[3])} \ast 1}$

$\overline{\text{int [3] = int} \ast} \quad \overline{\text{sizeof(int)} \ast 1}$

Quiz)



◆ 다음의 자료형과 값은?

- array
- array + 1
- *array + 1
- array[1] + 1
- *array[1] + 1

Quiz)

`int a[2][3] = { 1, 3, 5, 7, 9, 11 }; // 편의상 이 배열이 1000 번지부터 할당되었다고 가정하자.`

```
printf("%p\n", a);
```

```
printf("%p\n", *a);
```

```
printf("%p\n", **a);
```

```
printf("%p\n", a+1);
```

```
printf("%p\n", *(a + 1));
```

```
printf("%p\n", *a + 1);
```

```
printf("%p\n", *(*a + 1));
```

```
printf("%p\n", **a + 1);
```

```
printf("%d\n", (*a)[1]);
```

```
printf("%d\n", *a[1]);
```

```
printf("%d\n", (*(a + 1))[1]);
```

```
printf("%d\n", (*a + 1)[1]);
```

```
printf("%d\n", *(*a + 1))[1]);
```

```
printf("%d\n", (**a + 1)[1]);
```

Quiz)

```
int array[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

```
int (*arrptr)[4] = array;
```

```
int *iptr = array[1];
```

arrptr과 iptr을 각각 이용하여 3과 8을 화면에 출력하는 방법을 설명하라. 단, 배열 첨자를 사용하지 않아야 한다.