

# 전산 SMP 7주차

2014. 11. 06

김범수

[bskim45@gmail.com](mailto:bskim45@gmail.com)

Special thanks to 박기석 (kisuk0521@gmail.com)

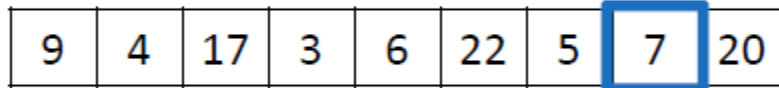
# 지난 내용 복습

Array, Sorting

# 자료구조

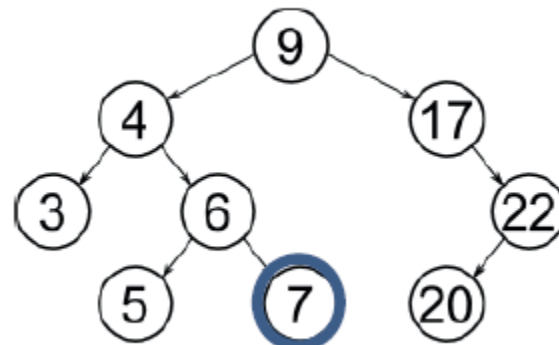
- 많은 데이터를 예쁘게 잘 저장해서
  - 쉽고 빠르게 꺼내 쓰려고
  - 데이터가 많아지면 찾는데도 오래 걸린다.
- 
- Example: finding a number from a set of numbers
    - How many comparisons do we need to retrieve 7?

In linear array



8 comparisons

In binary search tree

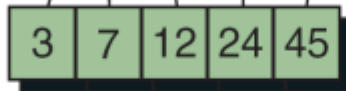


4 comparisons

# 선언, 초기화, 접근

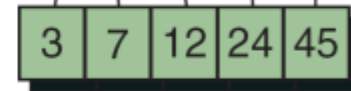
(a) Basic Initialization

```
int numbers[5] = {3, 7, 12, 24, 45};
```



(b) Initialization without Size

```
int numbers[ ] = {3, 7, 12, 24, 45};
```



(c) Partial Initialization

```
int numbers[5] = {3, 7};
```



The rest are filled with 0s

(d) Initialization to All Zeros

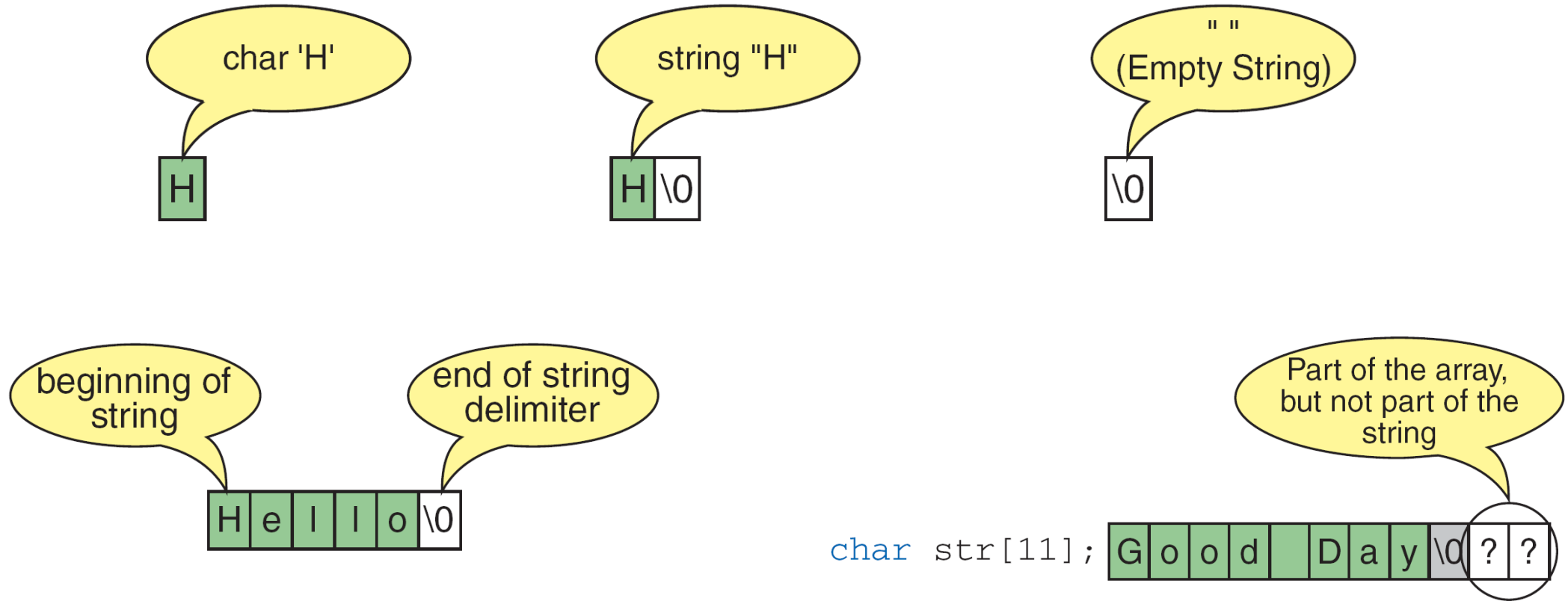
```
int lotsOfNumbers [1000] = {0};
```



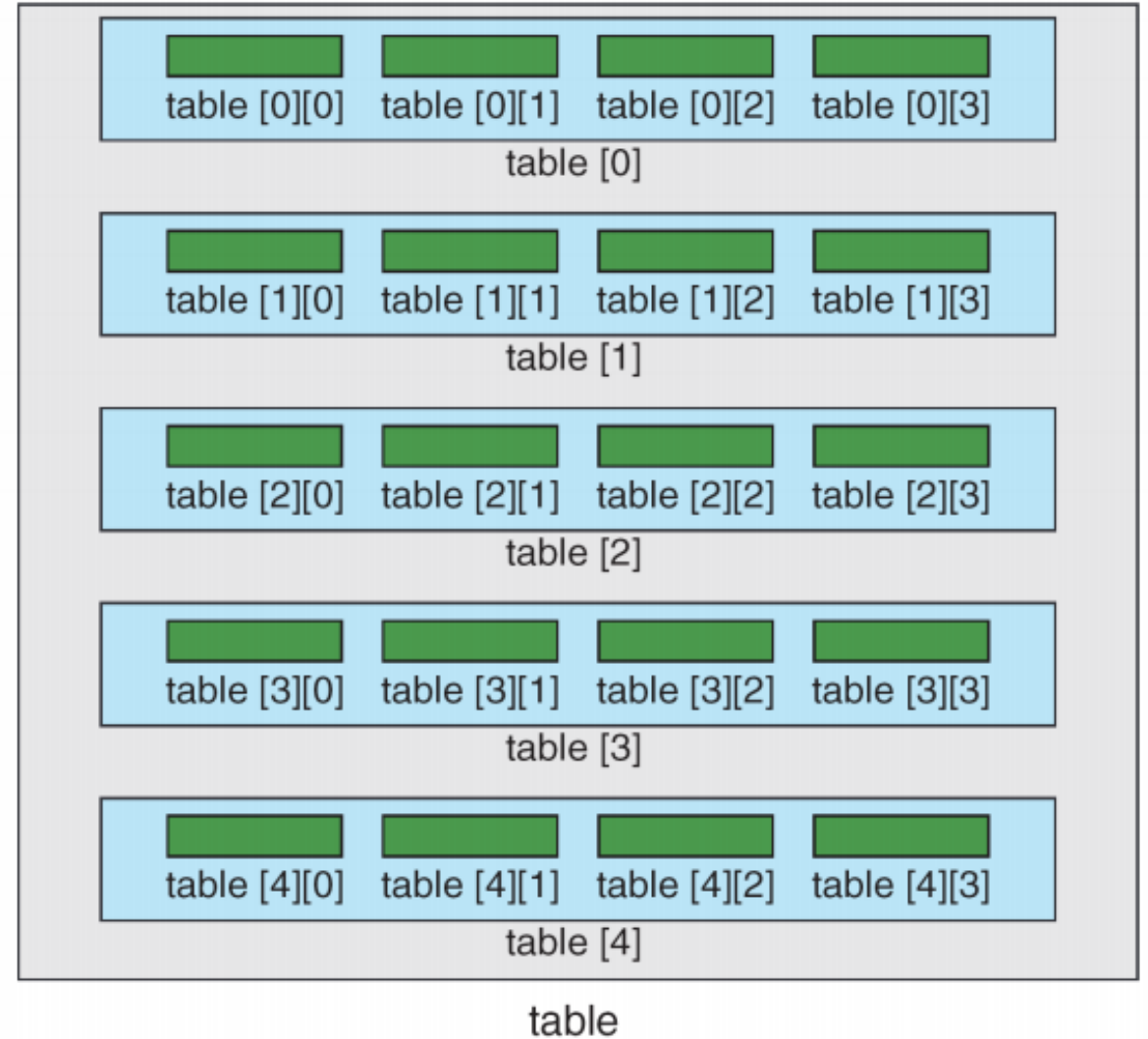
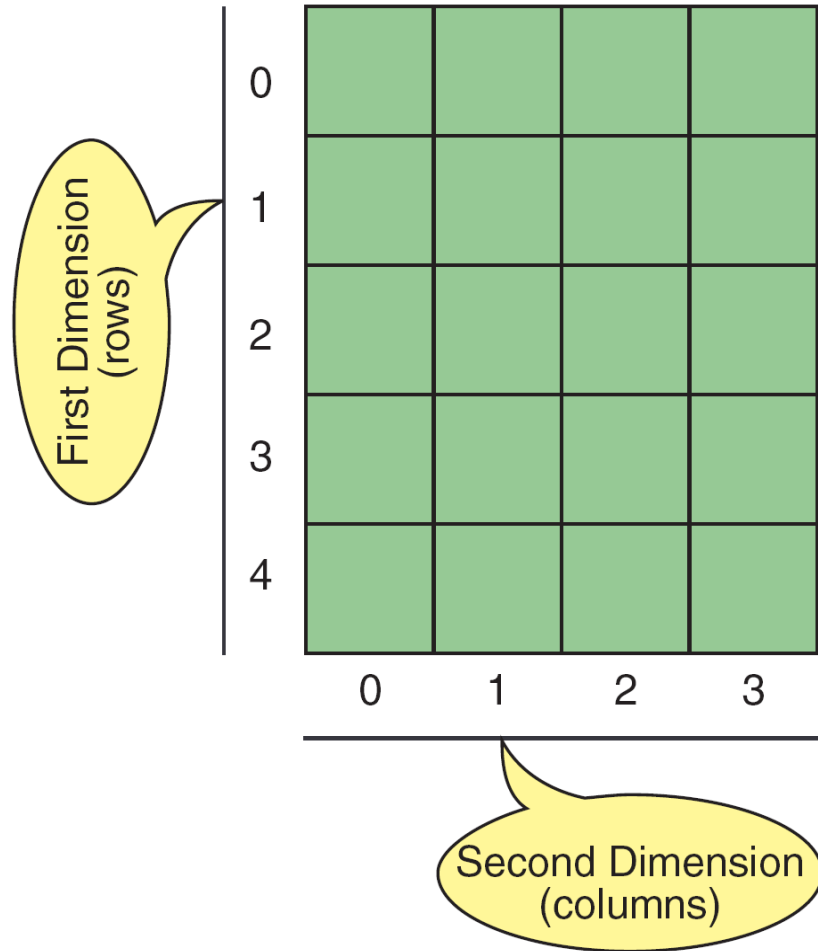
All filled with 0s

# 문자열(string)도 결국은 배열이다

- char 형의 배열 & 끝에 널문자 '\0'
- 배열 & 포인터를 완전하게 배우고 string에 대해서 더 자세히 합니다.



# Two-dimensional Array (Matrix)



# Array Sorting

- Selection Sort
  - Bubble Sort
  - Insertion Sort
- 
- 책에 있는 코드 꼭 보고 한번씩 짜 보세요 – 시험에 나옴

# 오늘 할 것

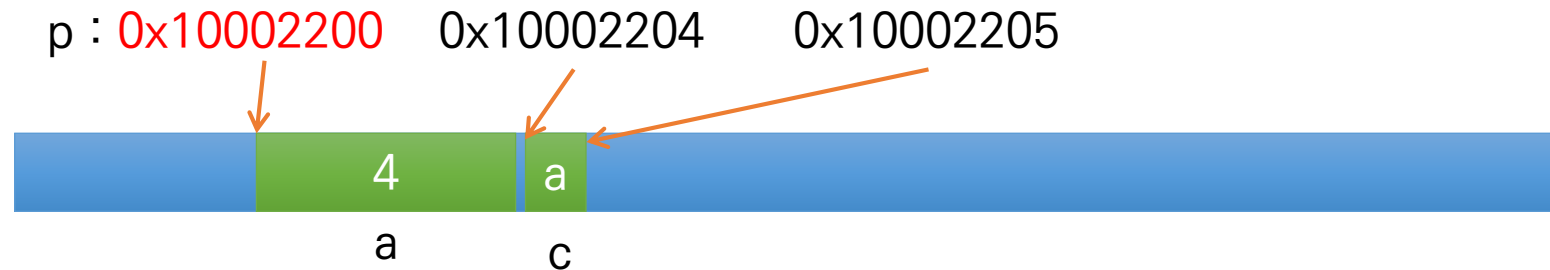
- 대망의 Pointer
- Pointer Applications (relation with Array)
- Dynamic Memory Allocation

Assn #3 질문받기

# Pointer

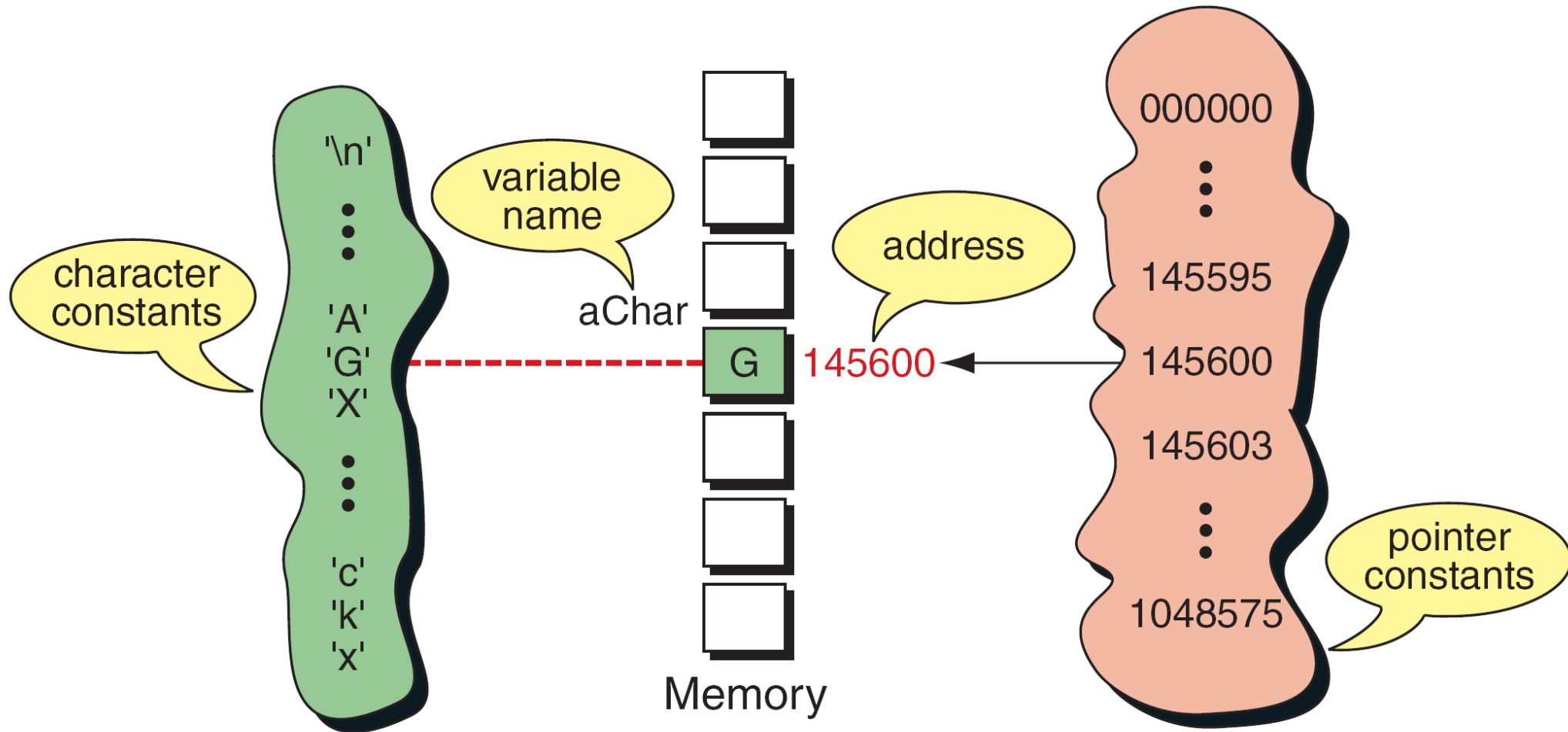
# Pointer란?

- 데이터 접근에 사용되는 **주소**를 저장하는 타입 – 포인터도 타입이다
- `int a = 4; char c = 'a'; int *p = &a;`



- 메모리는 긴 일차원 공간으로 볼 수 있고, 각 byte는 자신만의 주소를 가지고 있다.

# 변수와 주소와의 관계



# Pointer 선언, 사용

- Declaration

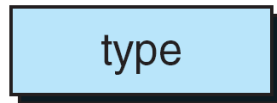
`int *ptr;`

→ 정수형 데이터가 저장된 메모리의 **위치(주소)**를 저장할 수 있는 포인터 변수

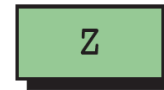
- 포인터의 타입 크기

- 4 byte!(32 bit)      ex) int : 4byte, char : 1 byte

data declaration



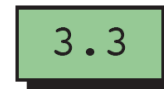
`char a;`



`int n;`



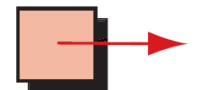
`float x;`



`char* p;`



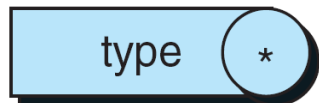
`int* q;`



`float* r;`



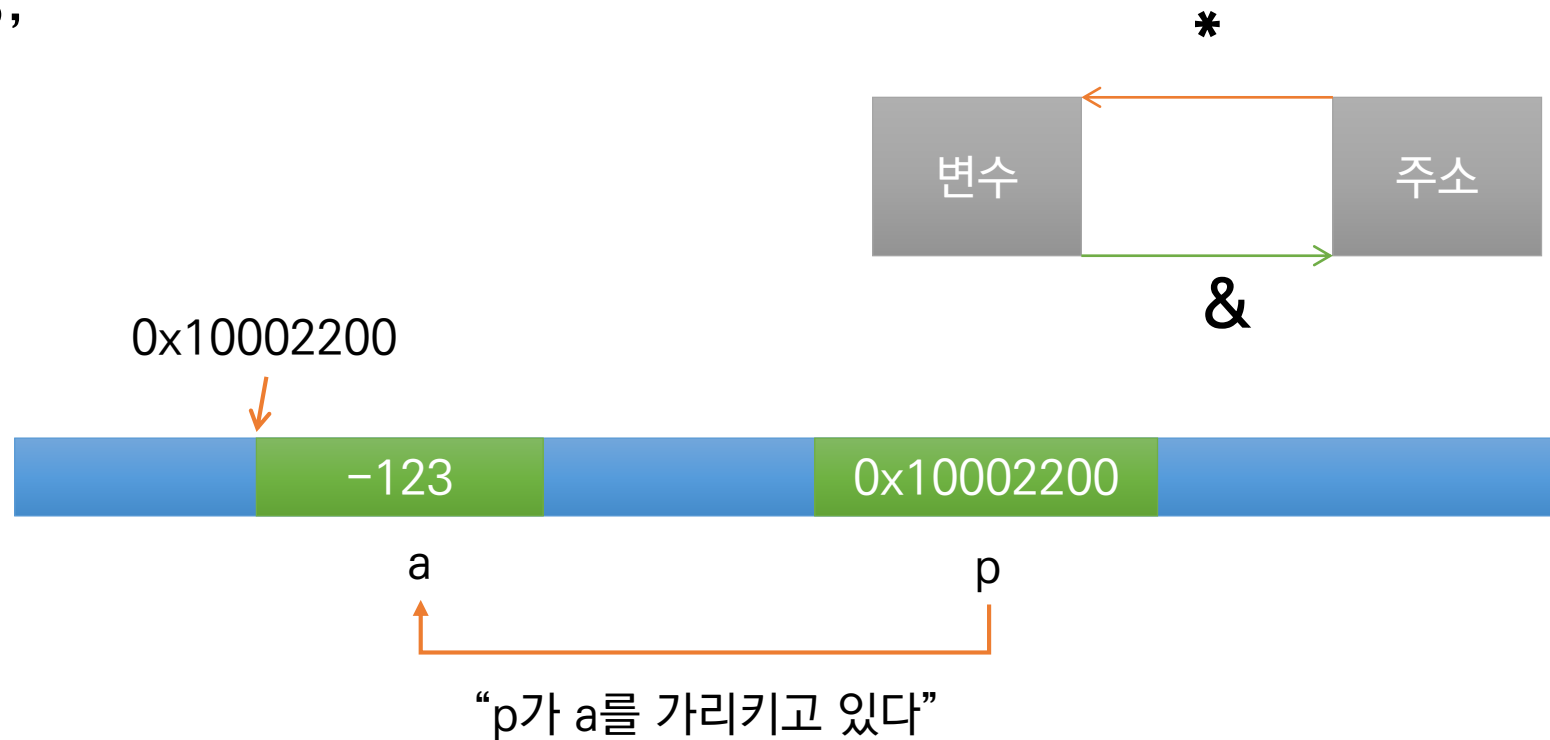
pointer declaration



# Pointer 선언, 사용

```
int a = -123;
```

```
int *p = &a;
```



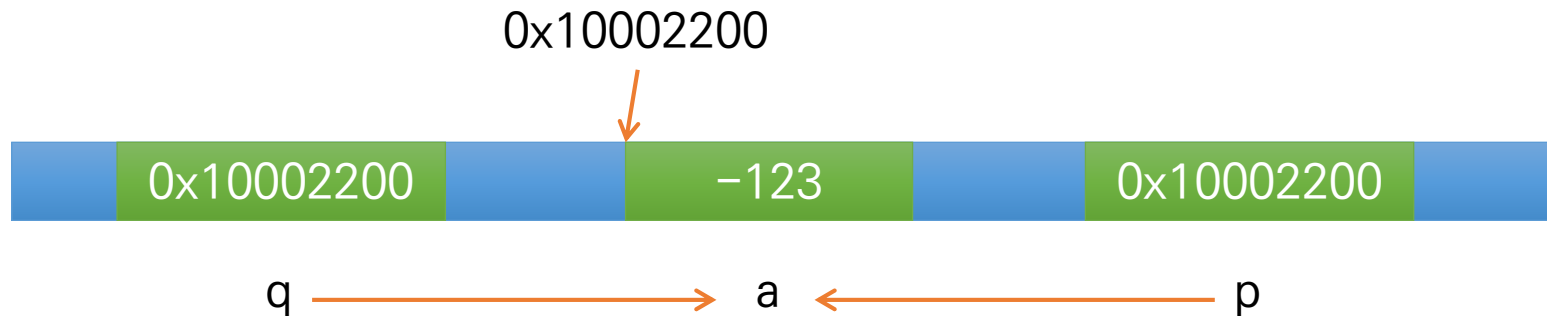
포인터 변수도 결국 '변수'이기 때문에 자신이 가리키는 주소를 메모리 어딘가에 저장하고 있다.

# Pointer 선언, 사용

```
int a = -123;
```

```
int *p = &a;
```

```
int *q = p; (or =&a;)
```



“p,q 둘 다 a를 가리키고 있다”

“a 변수의 데이터는 **하나!** a를 가리키는 포인터는 **두 개!**”

# 예제

```
int a = 14;
```

```
int* p = &a;
```

```
printf("%p %d %d", p, *p, a);
```

```
a = 20;
```

```
printf("%p %d %d", p, *p, a);
```

Results:

```
00135760 14 14
```

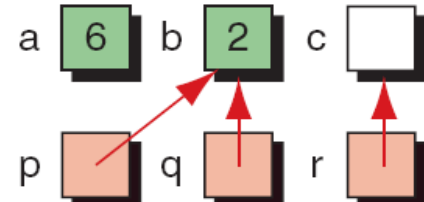
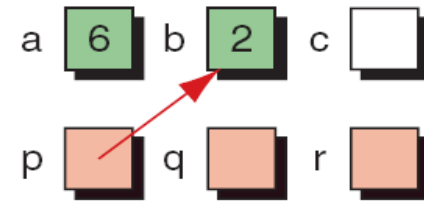
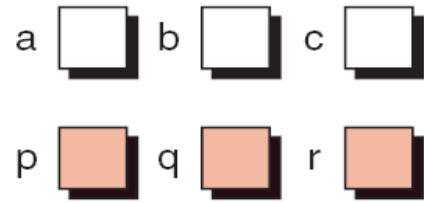
```
00135760 20 20
```

# Fun with Pointers

```
int a, b, c;  
int* p, q, r;
```

```
a = 6;  
b = 2;  
p = &b;
```

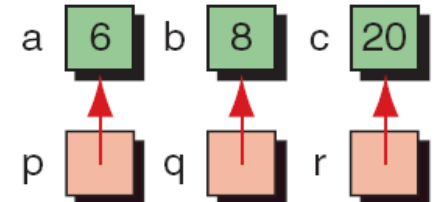
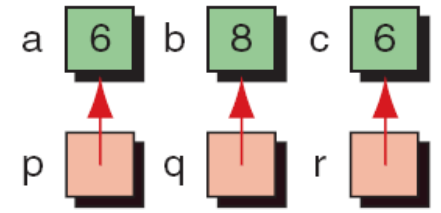
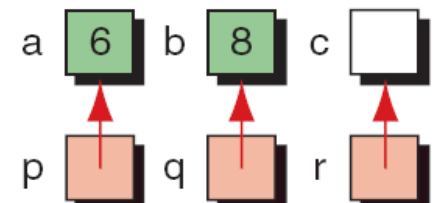
```
q = p;  
r = &c;
```



```
p = &a;  
*q = 8;
```

```
*r = *p;
```

```
*r = a + *q + *&c;
```



# NULL 을 가리키는 포인터

- 포인터 변수를 초기화 하려면?

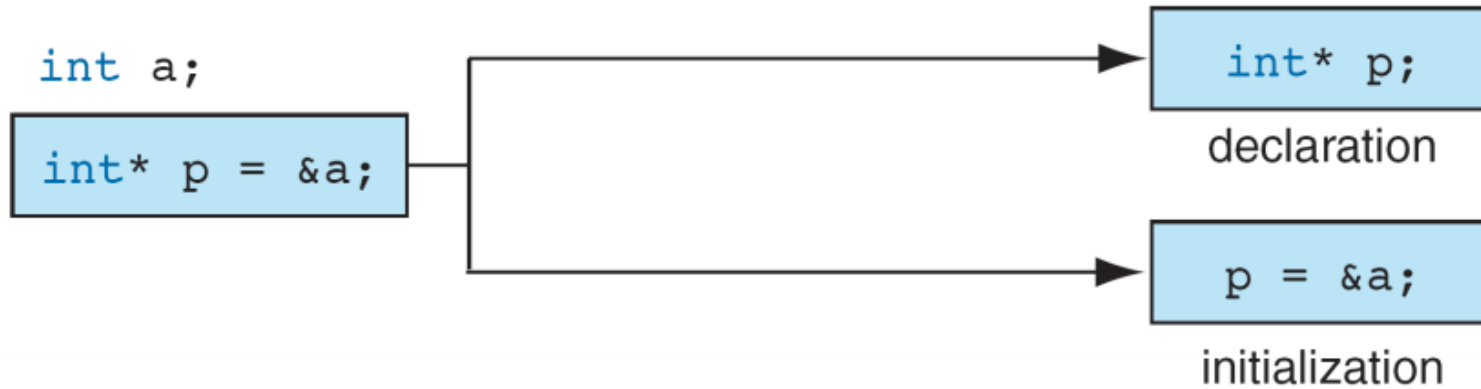
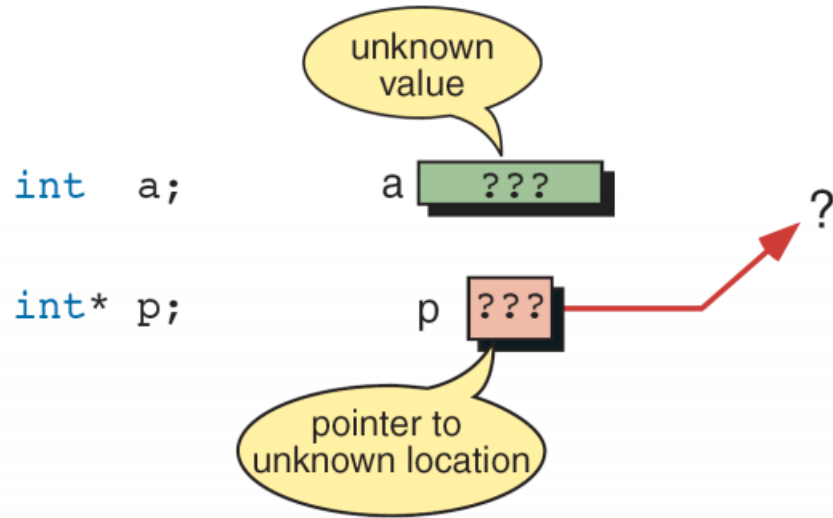


- 아무 수나 넣어서?
  - `int *p = 100;`
  - Segmentation Fault! 혹은 Fatal Error
  - 주소 100의 위치에 뭐가 있는지 모르고, 아무것도 없을 수도...
  - 함부로 건드릴 수 없는 영역에 접근하는 것은 치명적인 위험!
- NULL로 초기화

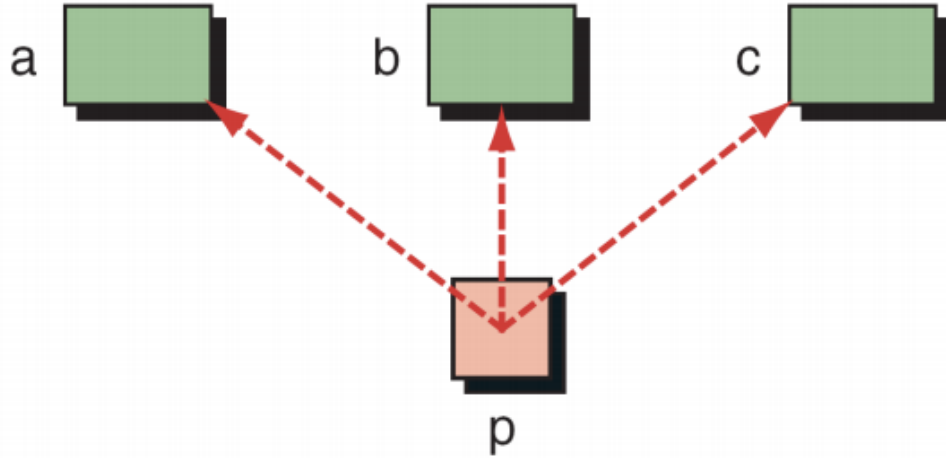
```
int *p = NULL;
```

활용 : `if (p != NULL)`

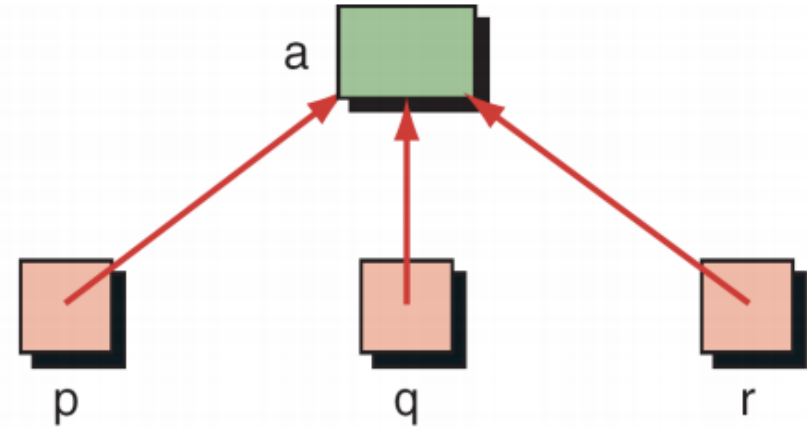
# 초기화 되지 않은 변수를 사용하면 위험하다



# Pointer의 유연성



```
int a, b, c;  
int *p = &a;  
p = &b;  
p = &c;
```



```
int a;  
int *p, *q, *r;  
p = &a;  
q = &a;  
r = &a;
```

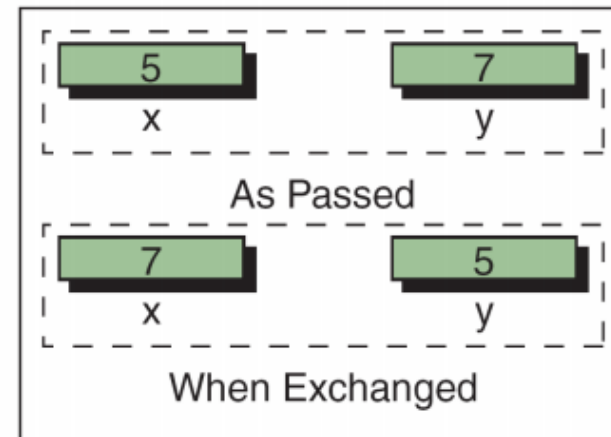
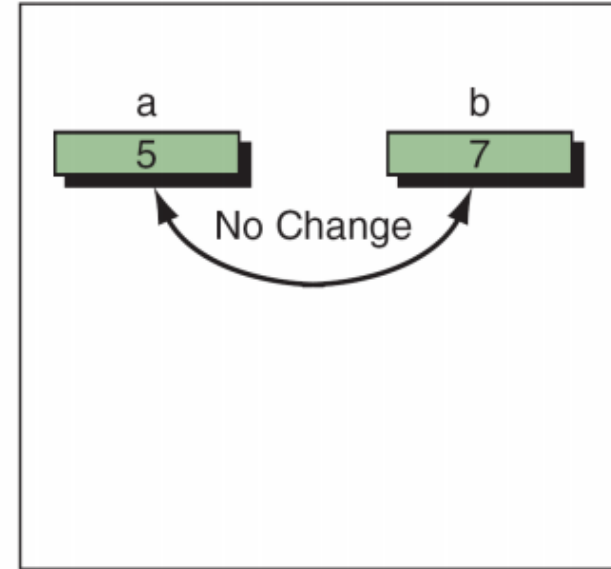
# Call-by-value

```
// Function Declarations
void exchange (int x, int y);

int main (void)
{
    int a = 5;
    int b = 7;
    exchange (a, b);
    printf ("%d %d\n", a, b);
    return 0;
} // main
```

```
void exchange (int x, int y)
{
    int temp;

    temp = x;
    x     = y;
    y     = temp;
    return;
} // exchange
```



# Call-by-reference

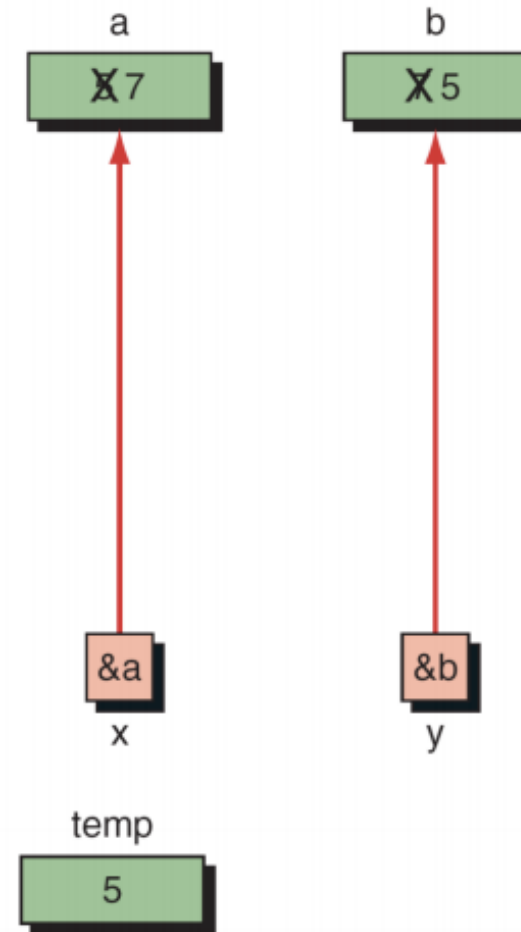
```
// Function Declaration
void exchange (int*, int*);

int main (void)
{
    int a = 5;
    int b = 7;

    exchange (&a, &b);
    printf("%d %d\n", a, b);
    return 0;
} // main
```

```
void exchange (int* px, int* py)
{
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
    return;
} // exchange
```

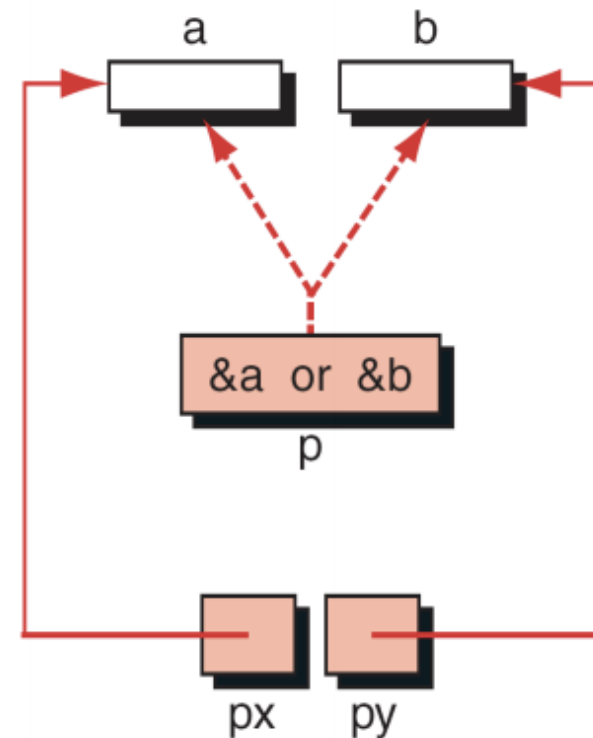


# 리턴 값으로서의 Pointer

```
// Prototype Declarations
int* smaller (int* p1, int* p2);

int main (void)
...
int a;
int b;
int* p;
...
scanf ( "%d %d", &a, &b );
p = smaller (&a, &b);
...
```

```
int* smaller (int* px, int* py)
{
    return (*px < *py ? px : py);
} // smaller
```



# 함수-Pointer 사용 특징

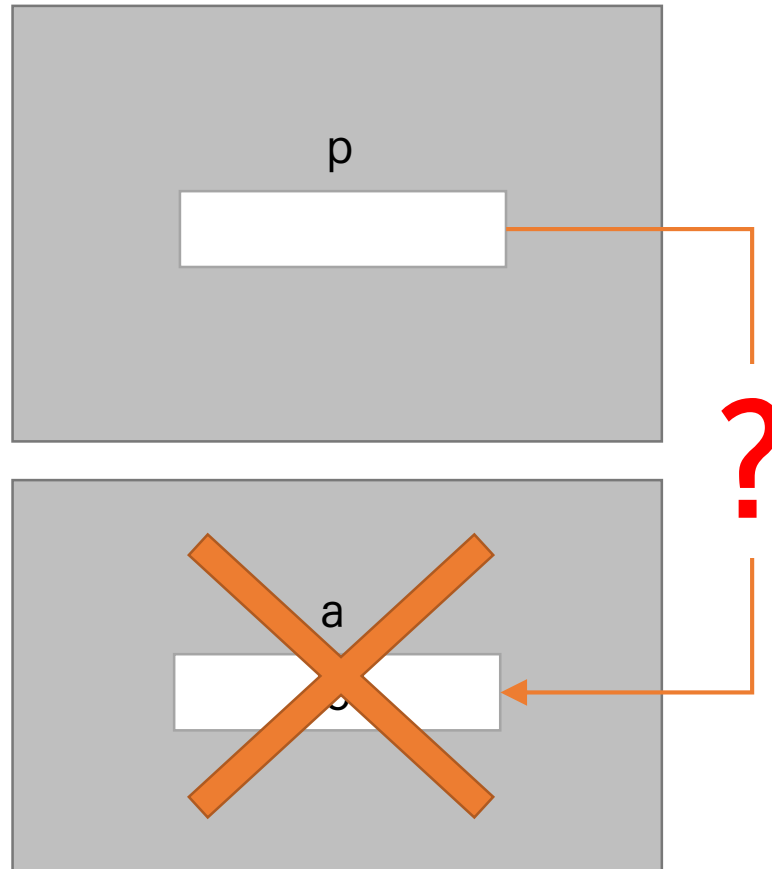
- 포인터 변수도 일반 변수처럼 함수 인자, 반환형으로 사용 가능!
- 함수가 여러 개의 리턴값을 가져야 할 때 - 여러 인자들을 포인터로 받아서 수정할 수 있다!
- 실제 그 메모리 위치를 찾아가 값을 바꿀 수 있다.
- 주의!!
  - 함수 안에서 만들어진 local variable을 가리키는 포인터는 반환할 수 없다.

# Local Variable의 주소를 리턴하는 경우

- Local variable는 그 함수(블럭)이 끝나면 사라진다.  
→ 알 수 없는 곳으로의 포인터를 반환하는 것과 같다.

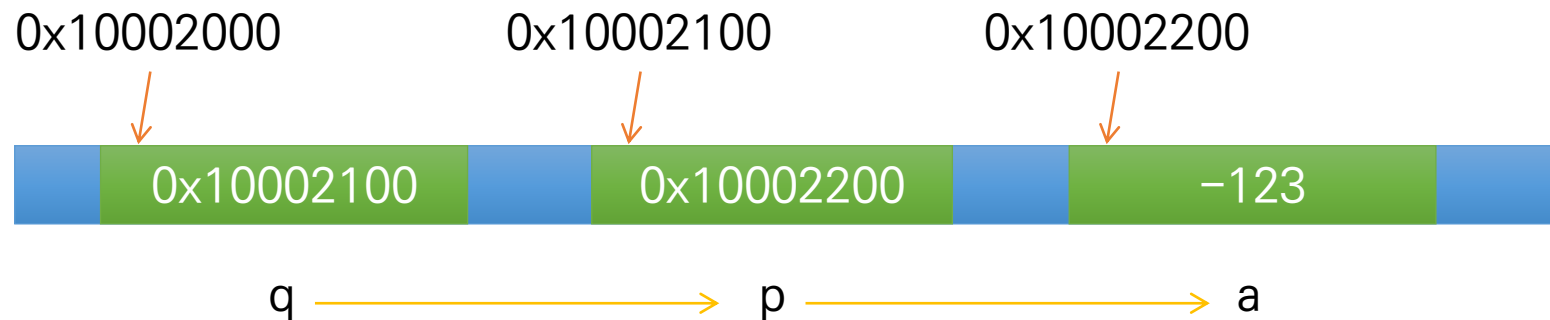
```
int main (void) {  
  
    int* p;  
  
    p = fun();  
    printf("%d", *p);  
}
```

```
int* fun (void) {  
  
    int a = 5;  
  
    return &a;  
}
```



# Pointer to Pointer (다중포인터)

- 포인터를 가리키는 포인터!
- 포인터도 주소를 저장하는 하나의 변수! 이기 때문에 메모리 특정 위치에 저장된다.

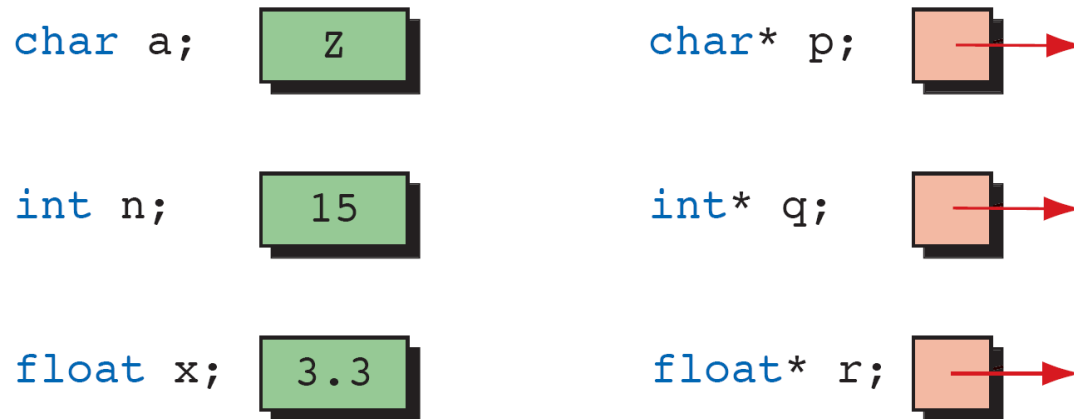


```
int a = -123;  
int *p = &a;  
int **q = &p;
```

# Pointer를 선언할 때 타입이 필요한 이유

```
int *p; char *q;
```

- 어차피 크기는 4byte(32 bit)로 다 똑같은 거 아닌가?
- 컴퓨터가 포인터로 메모리에 접근해 데이터를 읽어올 때!
  - 그 포인터 타입에 따라 가져오는 byte 수가 달라진다.
  - “int \* 포인터면 여기서 부터 4byte 까지 읽어서 정수로 쳐”
  - “char \* 포인터면 여기서 부터 1byte 만 읽어서 문자로 쳐”



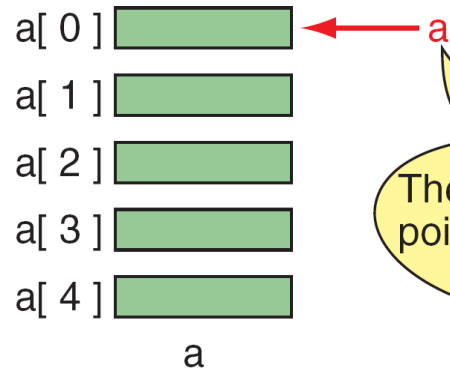
# void \*

- 어떤 데이터 타입과도 연관되지 않는 일반적 형태의 포인터
  - 어떤 포인터 값도 void pointer에 넣어줄 수 있다.
  - 어떤 포인터 값에도 void pointer 값을 넣어줄 수 있다.
- 단, void pointer는 그 상태 그대로는 dereferencing 할 수 없다!  
Why? 타입이 없음 → 내가 보려는 데이터의 타입이 뭐지? → compile error
- **casting**을 통해서 dereferencing 할 수 있다.

```
int a = -123;
void *p;
p = &a;
printf("%d", *(int *)p);
```

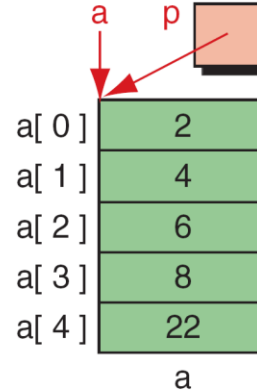
# Pointer Application

# 배열 포인터 (Pointer to Arrays)

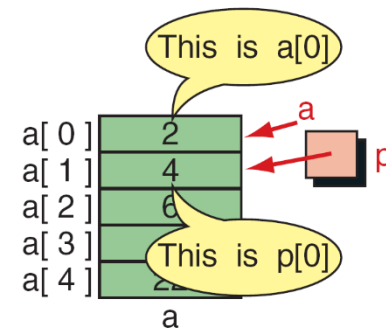
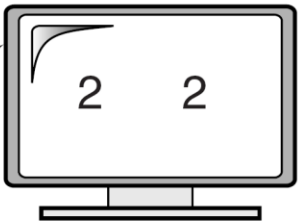


The name of an array is a pointer constant to its first element

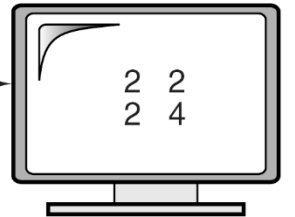
a ↔ &a[0]



```
#include <stdio.h>
int main (void)
{
  int a[5] = {2, 4, 6, 8, 22};
  int* p = a;
  ...
  printf("%d %d\n", a[0], *p);
  ...
  return 0;
} // main
```

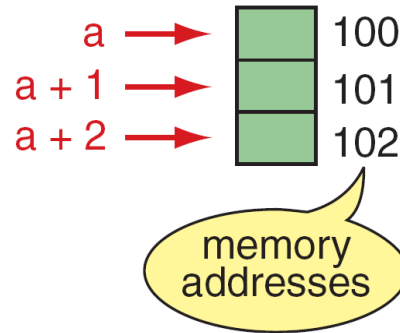
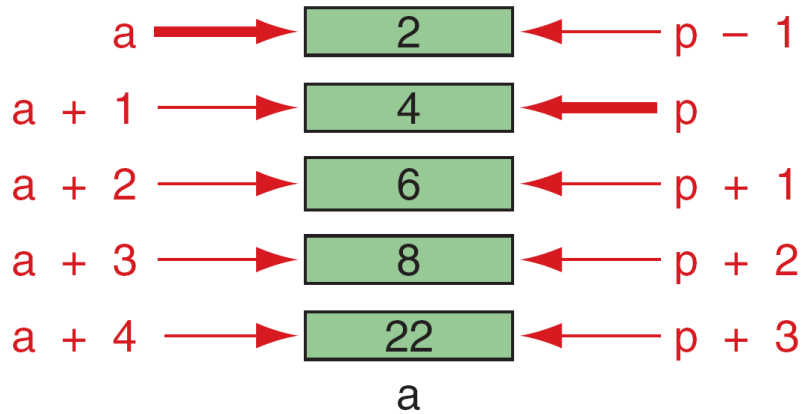


```
#include <stdio.h>
int main (void)
{
  int a[5] = {2, 4, 6, 8, 22};
  int* p;
  p = &a[1];
  printf("%d %d", a[0], p[-1]);
  printf("\n");
  printf("%d %d", a[1], p[0]);
  ...
} // main
```

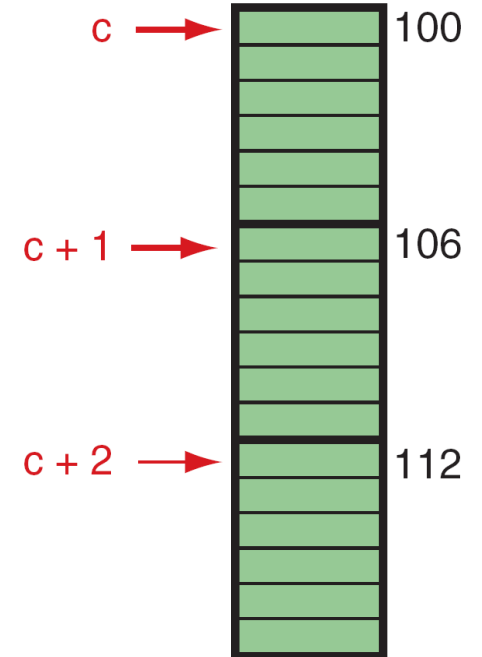
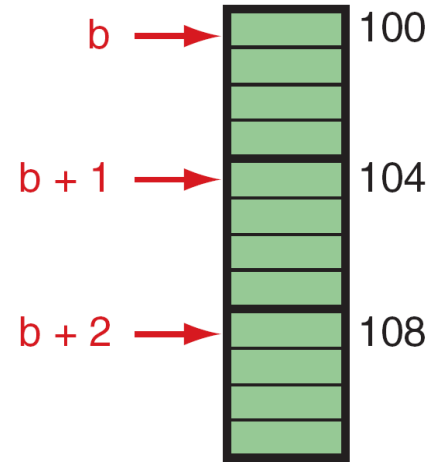


# 포인터 연산

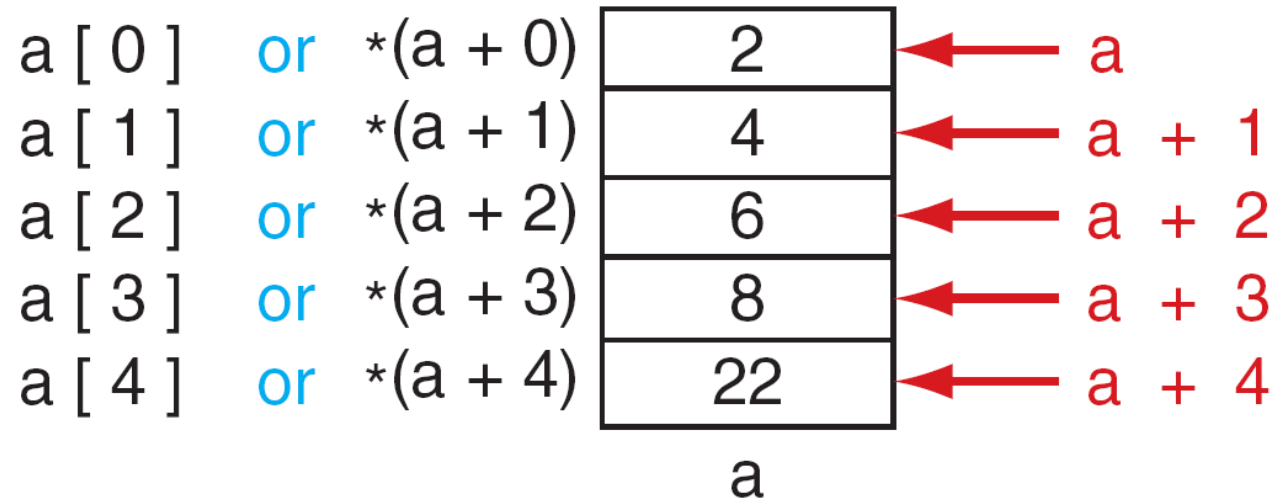
- 포인터  $p \rightarrow p \pm n \rightarrow p + n * (\text{sizeof}(\text{one element}))$



```
char a[3];  
int b[3];  
float c[3];
```



# Dereferencing



`*(a + i)` ↔ `a[i]`

# 2차원 Array

`table[2] == *(table+2)`

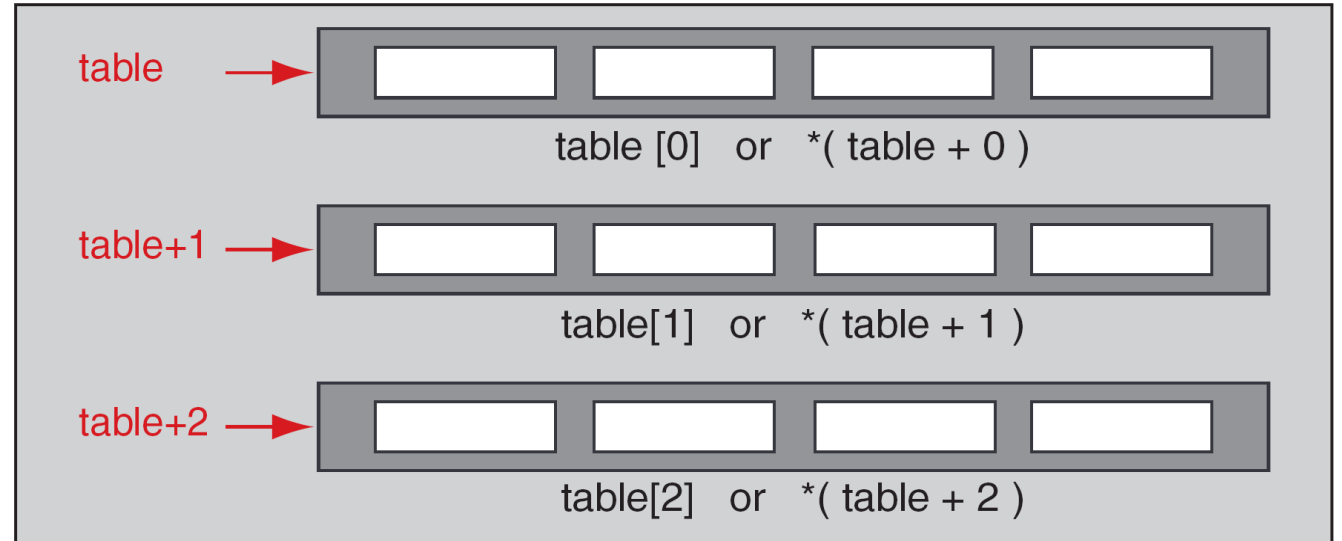
`table[2][1]`

`== (*(table+2))[1]`

`== *(table[2]+1)`

`== *(* (table+2)+1)`

- 헷갈리니 다차원 배열에서는 Index를 사용하자



`int table[3][4];`

```
for (i = 0; i < 3; i++)  
{  
    for (j = 0; j < 4; j++)  
        printf("%6d", *(* (table + i) + j));  
    printf( "\n" );  
} // for i
```

Print Table

# Arithmetic Operations on Pointers

- $p + 5$ ,  $5 + p$ ,  $p - 5$
- $p1 - p2$
- $p++$ ,  $--p$
- $p1 \gt = p2$
- $p1 \neq p2$

Long Form	Short Form
<code>if (ptr == NULL)</code>	<code>if (!ptr)</code>
<code>if (ptr != NULL)</code>	<code>if (ptr)</code>

# 문자열과 포인터

- 다시한번! 여러개의 문자+ '\0' 이 들어있는 배열
- 배열의 이름은 첫번째 element의 주소와 같다.



- 그래서 scanf로 %s 받을 때 & 안 붙인다!

# 문자열 한 줄 다 입력받기

```
char buf[10];
```

- gets(char \*buf)

- gets(buf); //stdin으로 부터 입력 받는다.  
//길이 제한 없음 (보안상 취약)  
//newline 이나 EOF 만날 때 까지  
//끝에 '\0'을 자동으로 붙여준다.

- fgets(char \*buf, unsigned size, FILE \*file)

- Fgets(buf, 10, stdin); //지정 stream으로 부터 입력 받는다.  
//최대 크기를 지정해 줄 수있다.(널 포함)  
//newline 이나 EOF 만날 때 까지  
//끝에 '\0'을 자동으로 붙여준다.  
//문자는 최대 size-1 개 까지 받는다.

# 배열 함수에 넘겨주기

- 1차원 배열

```
int ary1[10];
```

```
void function(int *ary);
```

```
void function(int ary[]);
```

- 2차원 배열

```
int ary2[5][6];
```

```
void function(int (*ary)[6]);
```

```
void function(int ary[][6]);
```

- 3차원 배열

```
(int ary[][3][4])
```

# 배열 받기 연습

- `int* arr1[3];` → `(int** ptr1)`
- `int * arr2[3][5];` → `(int* (*ptr2)[5])`
- `int** arr3[5];` → ?
- `int *** arr4[3][5];` → ?

# 다음시간

- Dynamic Memory Allocation
- Enum, Structure, Union