

2013 Spring Semester SMP

Day 3 – Array, Pointer, Function

Data type with I/O

Function

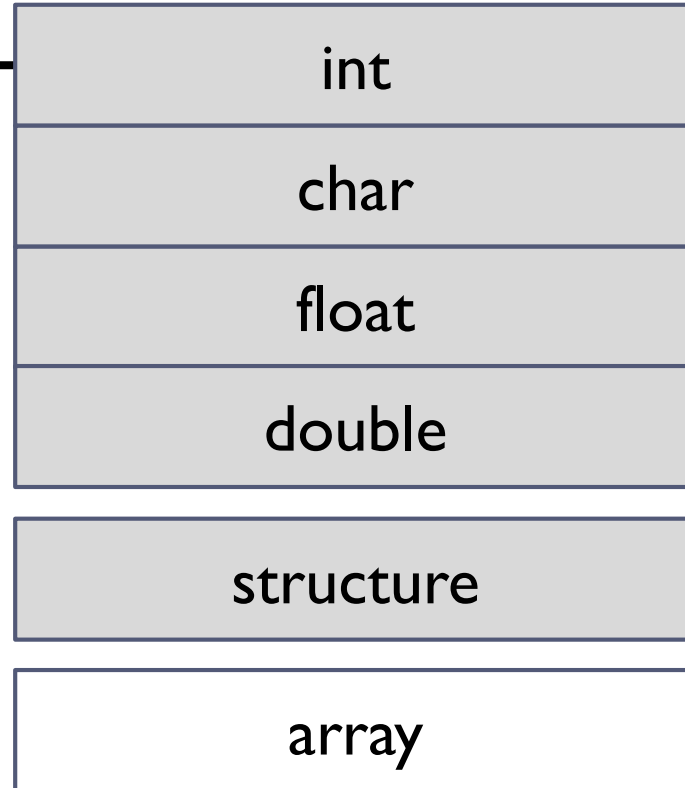
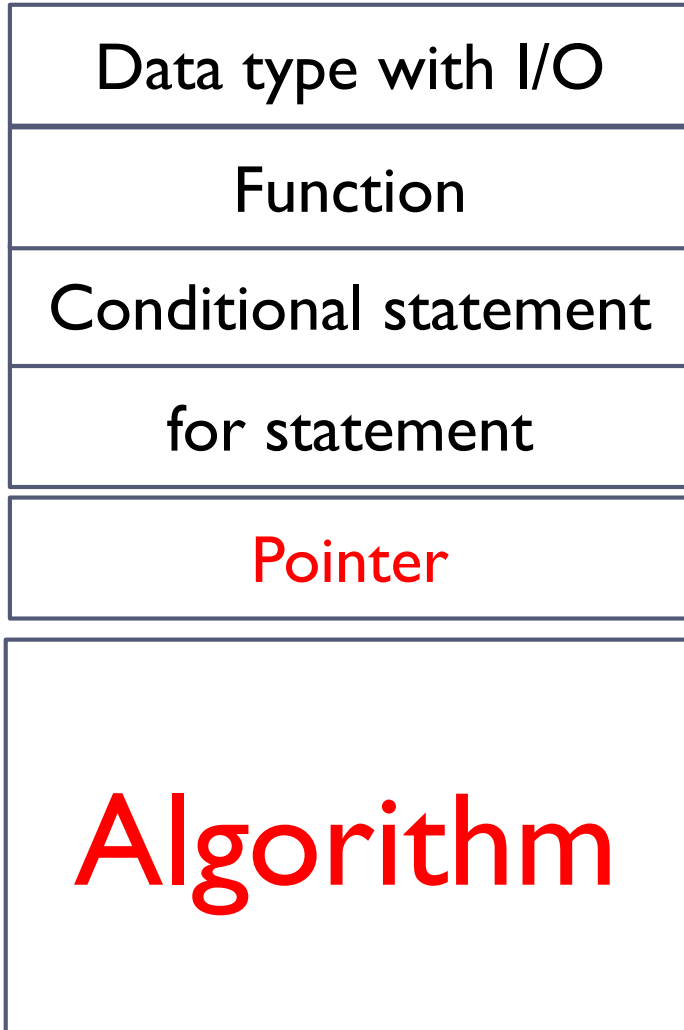
Conditional statement

for statement

Pointer

Algorithm





Pointer

- ▶ C를 이해함에 있어 제 1의 장벽
 - ▶ 제대로 이해하면 전혀 어려울 것 없음
 - (물론 전산 난이도에서...)
- ▶ Pointer를 이해해야 납득이 되는 현상이 많음
 - ▶ 후에 설명 하겠지만, 이런것들은 모르면 절대 에러를 찾지 못함
 - ▶ Casting을 몰라서 평균을 못 구하는 것과 비슷한 현상



Pointer

▶ C에서 값을 알려주는 방법?

- ▶ 변수! → 즉, 값을 갖고 있는 주머니의 이름을 알고있음
- ▶ 값 자체를 그냥 던져줌.
- ▶ `printf("%d",a);`

▶ 매우 비효율적인 매커니즘

- ▶ Ex. 난 숫자가 하나씩 들어있는 주머니를 세 개 갖고있음
너희는 이 주머니에 있는 숫자들을 모두 더해야 함
- ▶ 방법 1. 나한테 숫자가 뭔지 알려달라고 한다
그리고 난 후에 더한다



Pointer

▶ C에서 값을 알려주는 방법?

- ▶ 변수! → 즉, 값을 갖고 있는 주머니의 이름을 알고있음
- ▶ 값 자체를 그냥 던져줌.
- ▶ `printf("%d",a);`

▶ 매우 비효율적인 매커니즘

- ▶ Ex. 난 숫자가 하나씩 들어있는 주머니를 세 개 갖고있음
너희는 이 주머니에 있는 숫자들을 모두 더해야 함
 - ▶ 방법 1. 나한테 숫자가 뭔지 알려달라고 한다
그리고 난 후에 더한다
 - ▶ 방법 2. 너희가 직접 나한테 와서 주머니에 있는 숫자를 확인한다
그리고 난 후에 더한다
-

Pointer

▶ C에서 값을 알려주는 방법?

- ▶ 변수! → 즉, 값을 갖고 있는 주머니의 이름을 알고있음
- ▶ 값 자체를 그냥 던져줌.
- ▶ `printf("%d",a);`

▶ 매우 비효율적인 매커니즘

- ▶ Ex. 난 숫자가 하나씩 들어있는 주머니를 세 개 갖고있음
너희는 이 주머니에 있는 숫자들을 모두 더해야 함

- ▶ 방법 1. 나한테 숫자가 뭔지 알려달라고 한다
그리고 난 후에 더한다

- ▶ 방법 2. 너희가 직접 나한테 와서 주머니에 있는 숫자를 확인한다
그리고 난 후에 더한다

그냥 둘 다 그저 그런 방법

Pointer

▶ C에서 값을 알려주는 방법?

- ▶ 변수! → 즉, 값을 갖고 있는 주머니의 이름을 알고있음
- ▶ 값 자체를 그냥 던져줌.
- ▶ `printf("%d",a);`

▶ 매우 비효율적인 매커니즘

- ▶ Ex. 마트에서 우유 하나를 사 와야 한다. 여기서 난 마트 주인
이제 너희가 마트에서 우유를 사 가면 됨

- ▶ 방법 1. 나한테 마트를 너희한테 가져 오라고 한다(?)
그 후 우유를 삼



Pointer

▶ C에서 값을 알려주는 방법?

- ▶ 변수! → 즉, 값을 갖고 있는 주머니의 이름을 알고있음
- ▶ 값 자체를 그냥 던져줌.
- ▶ `printf("%d",a);`

▶ 매우 비효율적인 매커니즘

- ▶ Ex. 마트에서 우유 하나를 사 와야 한다. 여기서 난 마트 주인
이제 너희가 마트에서 우유를 사 가면 됨

- ▶ 방법 1. 나한테 마트를 너희한테 가져 오라고 한다(?)
그 후 우유를 삼

→ 마트를 너희들 집 앞에다가 정확히 똑같은 것을
하나 더 지어야 함

Pointer

▶ C에서 값을 알려주는 방법?

- ▶ 변수! → 즉, 값을 갖고 있는 주머니의 이름을 알고있음
- ▶ 값 자체를 그냥 던져줌.
- ▶ `printf("%d",a);`

▶ 매우 비효율적인 매커니즘

- ▶ Ex. 마트에서 우유 하나를 사 와야 한다. 여기서 난 마트 주인
이제 너희가 마트에서 우유를 사 가면 됨
- ▶ 방법 2. 너희가 마트에 와서 우유를 사 간다



Pointer

▶ C에서 값을 알려주는 방법?

- ▶ 변수! → 즉, 값을 갖고 있는 주머니의 이름을 알고있음
- ▶ 값을 자체를 그냥 던져줌.
- ▶ `printf("%d",a);`

▶ 매우 비효율적인 매커니즘

- ▶ Ex. 마트에서 우유 하나를
이제 너희가 마트에서
- ▶ 방법 2. 너희가 마트에 와서



Pointer

▶ 엉뚱하지만 정확한 비유

- ▶ 값을 알아내기 위해서 “값을 넘겨주는” 행위가 있고
“값의 위치를 넘겨주는” 행위가 있음
- ▶ 후자가 훨씬 더 일반적이고 오류가 없는 경우
 - 다른 슈퍼에서 물건을 사오게 하고 싶으면 위치만 바꾸면 됨
 - 생각해보면 방법 1.의 경우 내가 갖고있는 슈퍼와 너희가 우유를 사는 슈퍼는 서로 다른 슈퍼
 - 즉, 그 슈퍼에서 우유를 사도 내 마트에 갖고 있는 우유는 변하지 않음
 - → **의도치 않은 Error**

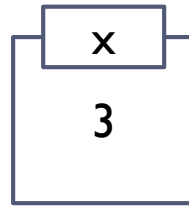


Pointer

- ▶ `int* p;`
 - ▶ “int형을 가리키겠다”는 의미에서 `int*`
 - ▶ `p`가 갖는 값은 `int`형 변수의 주소값

- ▶ **Example**

- ▶ `int x = 3;`
 - ▶ `int* p;`



P

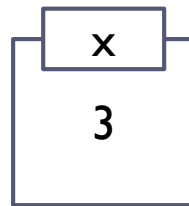


Pointer

- ▶ `int* p;`
 - ▶ “int형을 가리키겠다”는 의미에서 `int*`
 - ▶ `p`가 갖는 값은 `int`형 변수의 주소값

- ▶ **Example**

- ▶ `int x = 3;`
- ▶ `int* p;`



이제 `p`가 `x`를 가리키게 하고 싶으면?

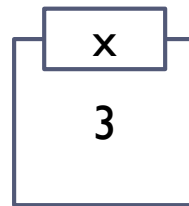


Pointer

- ▶ `int* p;`
 - ▶ “int형을 가리키겠다”는 의미에서 `int*`
 - ▶ `p`가 갖는 값은 `int`형 변수의 주소값

- ▶ **Example**

- ▶ `int x = 3;`
- ▶ `int* p;`



P

이제 `p`가 `x`를 가리키게 하고 싶으면?

`p = &x;`



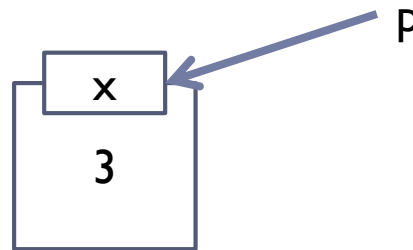
Pointer

- ▶ `int* p;`

- ▶ “int형을 가리키겠다”는 의미에서 `int*`
- ▶ `p`가 갖는 값은 `int`형 변수의 주소값

- ▶ **Example**

- ▶ `int x = 3;`
- ▶ `int* p;`



이제 `p`가 `x`를 가리키게 하고 싶으면?

`p = &x;`



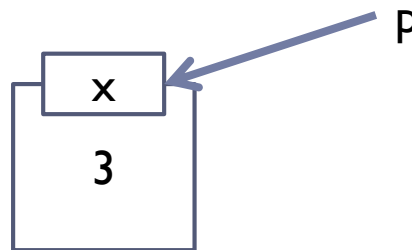
Pointer

▶ `int* p;`

- ▶ “int형을 가리키겠다”는 의미에서 `int*`
- ▶ `p`가 갖는 값은 `int`형 변수의 주소값

▶ Example

- ▶ `int x = 3;`
- ▶ `int* p;`



`p`가 가리키는 값이 무엇인지 알고 싶다면?



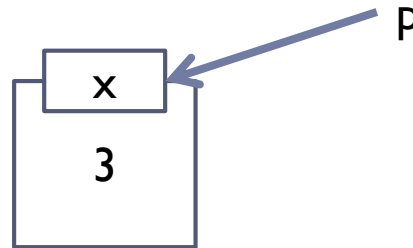
Pointer

- ▶ `int* p;`

- ▶ “int형을 가리키겠다”는 의미에서 `int*`
- ▶ `p`가 갖는 값은 `int`형 변수의 주소값

- ▶ Example

- ▶ `int x = 3;`
- ▶ `int* p;`



`p`가 가리키는 값이 무엇인지 알고 싶다면? `*p = 4;`



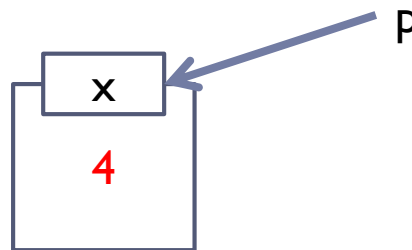
Pointer

▶ `int* p;`

- ▶ “int형을 가리키겠다”는 의미에서 `int*`
- ▶ `p`가 갖는 값은 `int`형 변수의 주소값

▶ Example

- ▶ `int x = 3;`
- ▶ `int* p;`



`p`가 가리키는 값이 무엇인지 알고 싶다면? `*p = 4;`



Pointer

포인터 곳



Pointer

▶ malloc

- ▶ pointer에 공간을 할당해 줌
- ▶ `int *p;`라고 단순히 선언만 해 놓으면 가리키는 애 없음
→ 공간 만들어줄 필요가 있음
- ▶ `p = (int *)malloc(4);` // 4 byte만큼 p에 공간 할당
- ▶ 이제 일반적인 변수처럼 p를 사용할 수 있음

▶ free

- ▶ 할당했던 공간을 지움
 - ▶ `free(p);`
-



Pointer

▶ malloc

- ▶ pointer에 공간을 할당해 줌
- ▶ `int *p;`라고 단순히 선언만 해 놓으면 가리키는 애 없음
→ 공간 만들어줄 필요가 있음
- ▶ `p = (int *)malloc(sizeof(int));` // 4 byte만큼 p에 공간 할당
- ▶ 이제 일반적인 변수처럼 p를 사용할 수 있음

▶ free

- ▶ 할당했던 공간을 지움
 - ▶ `free(p);`
-



Pointer

```
▶ #include <stdio.h>
#include <stdlib.h>
int main(){
    int* p;
    p = (int *)malloc(sizeof(int));
    *p = 4;
    printf("%d\n", *p);
    return 0;
}
```

리키는 애 없음

함 p에 공간 할당

있음

▶ free

- ▶ 할당했던 공간을 지움
- ▶ free(p);

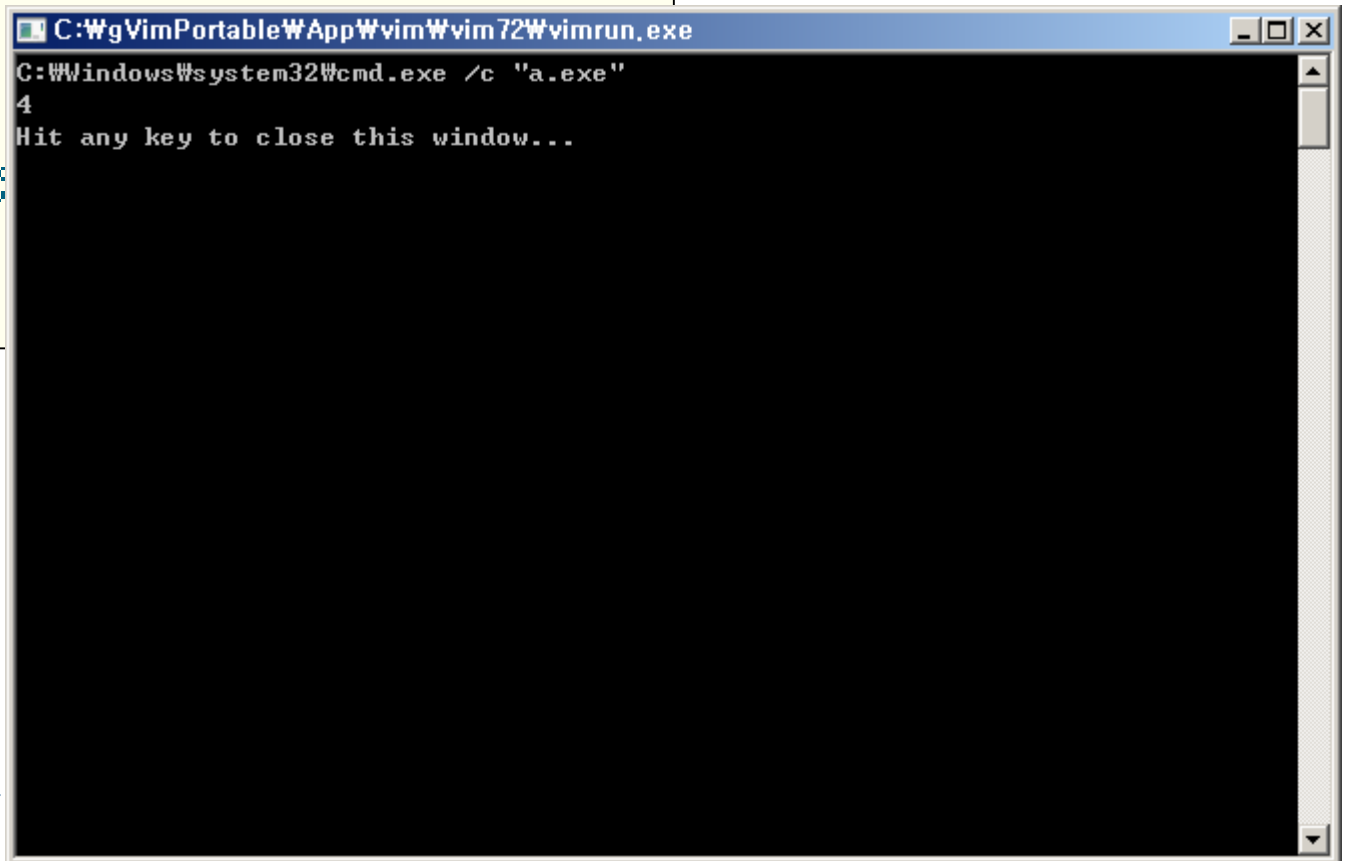


Pointer

```
▶ #include <stdio.h>
#include <stdlib.h>
int main(){
    int* p;
    p = (int*) malloc(4);
    *p = 4;
    printf("%d\n", *p);
    return 0;
}
```

▶ gcc

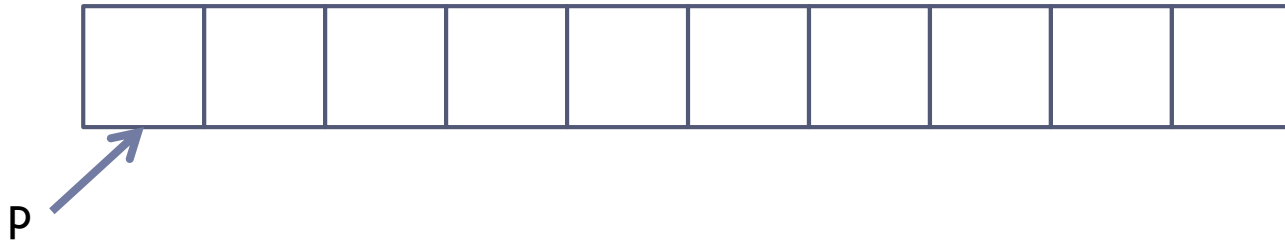
- ▶ 할당했던
- ▶ free(p);



```
C:\WgVimPortable\WApp\vim\vim72\vimrun.exe
C:\Windows\system32\cmd.exe /c "a.exe"
4
Hit any key to close this window...
```

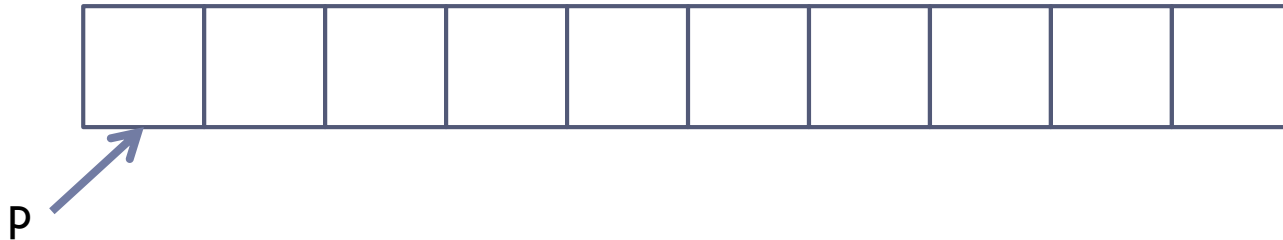

Pointer

- ▶ malloc을 이용하여 공간을 더 많이 잡으면?
 - ▶ `int* p = (int *)malloc(sizeof(int) * 10);`
 - ▶ 40byte가 p에 할당됨



Pointer

- ▶ malloc을 이용하여 공간을 더 많이 잡으면?
 - ▶ `int* p = (int *)malloc(sizeof(int) * 10);`
 - ▶ 40byte가 p에 할당됨



`int* q = p + 1;`



Pointer

- ▶ malloc을 이용하여 공간을 더 많이 잡으면?
 - ▶ `int* p = (int *)malloc(sizeof(int) * 10);`
 - ▶ 40byte가 p에 할당됨

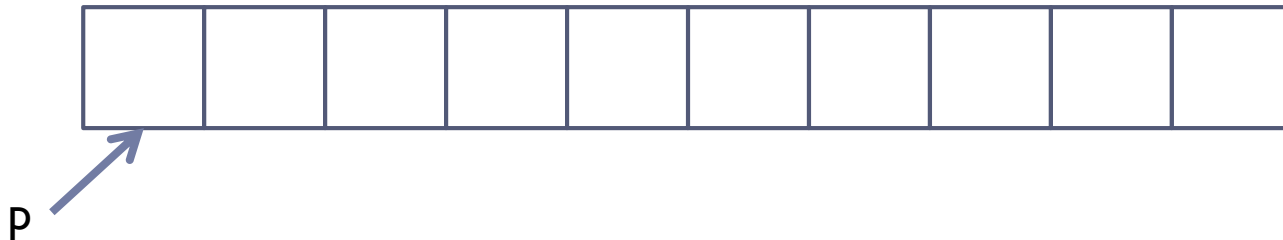


`int* q = p + 1;`



Pointer

- ▶ malloc을 이용하여 공간을 더 많이 잡으면?
 - ▶ `int* p = (int *)malloc(sizeof(int) * 10);`
 - ▶ 40byte가 p에 할당됨



`*(p+3) = 4;`



Pointer

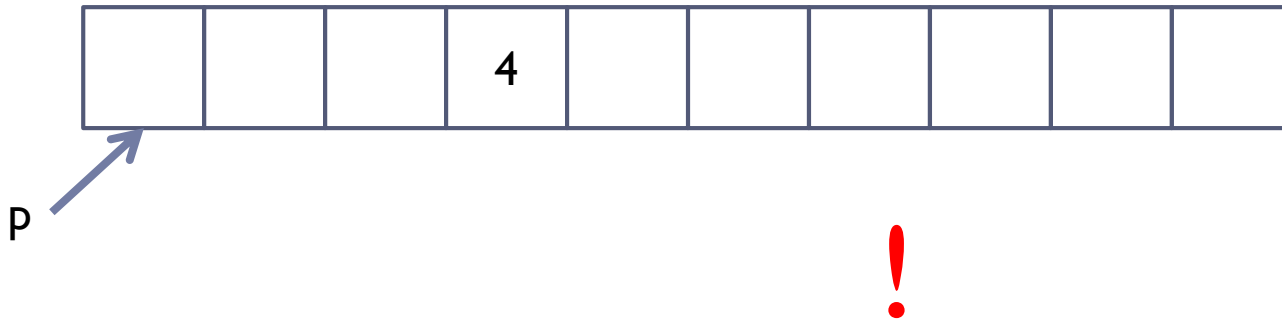
- ▶ malloc을 이용하여 공간을 더 많이 잡으면?
 - ▶ `int* p = (int *)malloc(sizeof(int) * 10);`
 - ▶ 40byte가 p에 할당됨



`*(p+3) = 4;`

Pointer

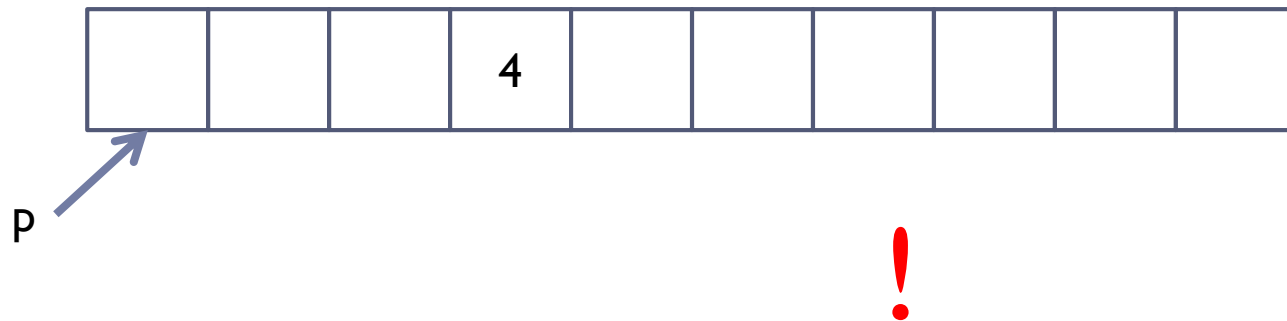
- ▶ malloc을 이용하여 공간을 더 많이 잡으면?
 - ▶ `int* p = (int *)malloc(sizeof(int) * 10);`
 - ▶ 40byte가 p에 할당됨



`*(p+3) = 4;`

Pointer

- ▶ malloc을 이용하여 공간을 더 많이 잡으면?
 - ▶ `int* p = (int *)malloc(sizeof(int) * 10);`
 - ▶ 40byte가 p에 할당됨



`*(p+3) = 4;`

→ Array의 매커니즘

Pointer & Array

- ▶ `int Array[30];`
 - ▶ `int* Array = (int *)sizeof(malloc(int) * 30);` 와 같음
- ▶ `Array[2] = 3;`
 - ▶ `*(Array+2) = 3;`



Pointer & Array

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int Array[10];
    *Array = 119;
    printf("%d\n", *Array);
    return 0;
}
```

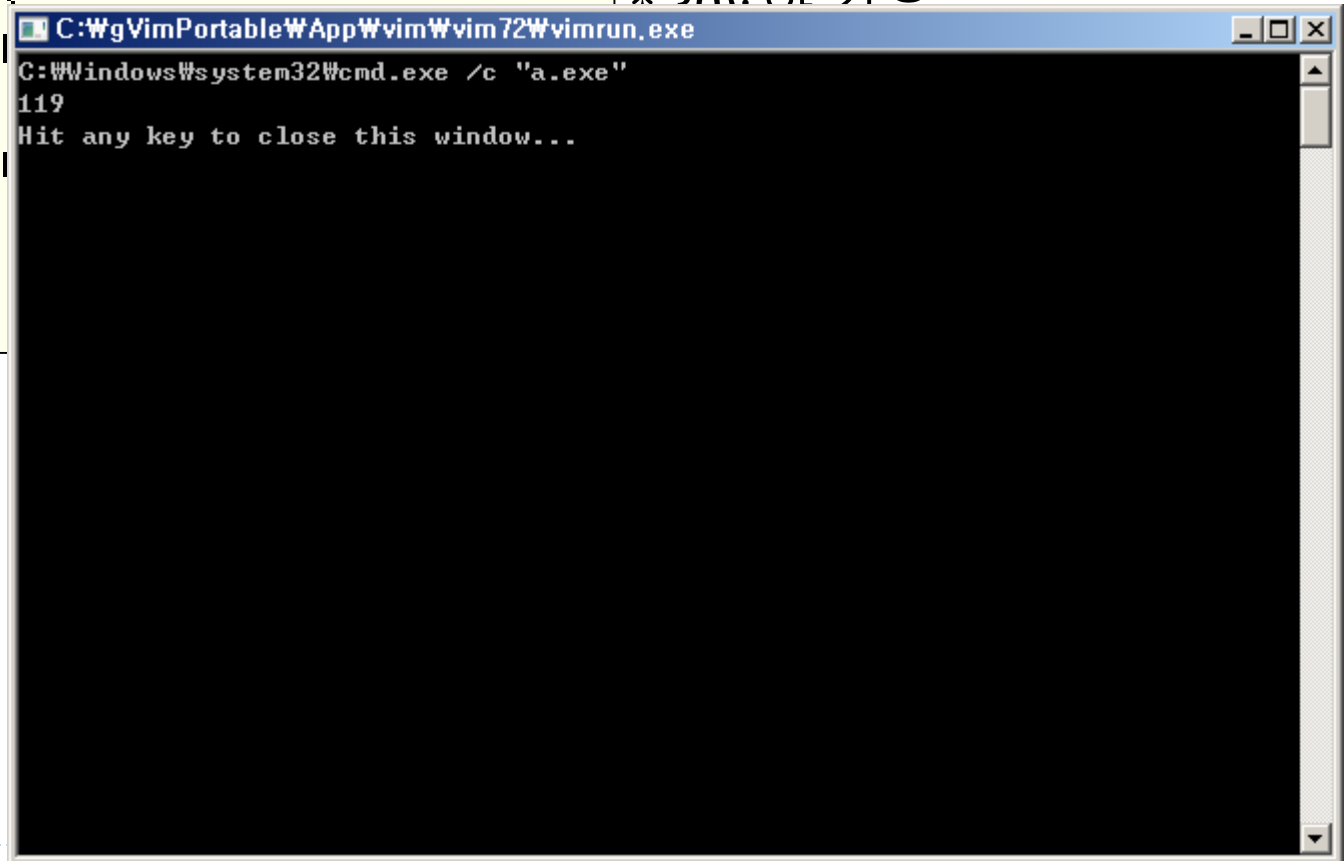
* 30); 와 같음



Pointer & Array

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int Array[10];
    *Array = 119;
    printf("%d", *Array);
    return 0;
}
```

* 201. 01. 21. 0



```
C:\WgVimPortable\App\vim\vim72\vimrun.exe
C:\Windows\system32\cmd.exe /c "a.exe"
119
Hit any key to close this window...
```

Pointer & Array

- ▶ 2차원 Array는?
 - ▶ 2가지 방법으로 이해하기
 - 방법 1. 1차원 배열을 만들어서 적당히 자른다



Pointer & Array

- ▶ 2차원 Array는?
 - ▶ 2가지 방법으로 이해하기
 - 방법 1. 1차원 배열을 만들어서 적당히 자른다

```
int Array[3][3];
```



Pointer & Array

- ▶ 2차원 Array는?

- ▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다

```
int Array[3][3];
```

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---



Pointer & Array

- ▶ 2차원 Array는?

- ▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다

```
int Array[3][3];
```

0	1	2
3	4	5
6	7	8



Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다

```
int Array[3][3];
```

0	1	2
3	4	5
6	7	8

Access Array[2][1]



Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다

```
int Array[3][3];
```

0	1	2
3	4	5
6	7	8

Access Array[2][1]

⇔ 1차원 배열의 6번째



Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다

```
int Array[3][3];
```

0	1	2
3	4	5
6	7	8

Access Array[2][1]

⇔ 1차원 배열의 6번째



Pointer & Array

▶ Problem

int Array[r][c] 를 선언하여 Array[y][x]에 접근했을 때, 실제로 ID array의 어느 부분에 접근해야 하는지 생각.

ex. If $r = 3, c = 3, y = 2, x = 1$, then our result is 6

0	1	2
3	4	5
6	7	8



Pointer & Array

- ▶ 2차원 Array는?

- ▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
 - 방법 2. 포인터를 가리키는 포인터를 만든다 (??)



Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다 (??)

```
int Array[3][3];
```



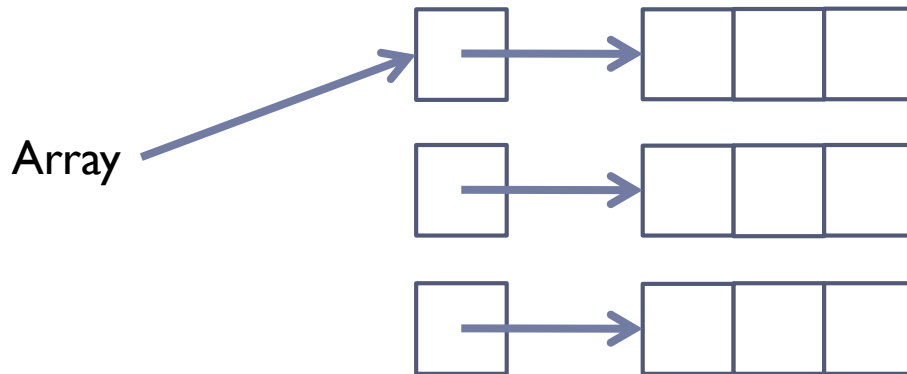
Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다 (??)

```
int Array[3][3];
```



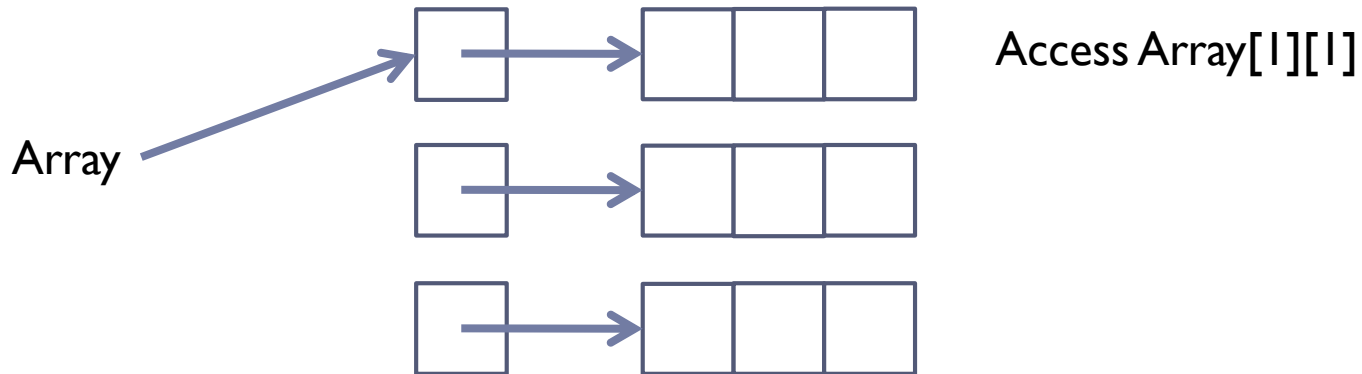
Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다 (??)

```
int Array[3][3];
```



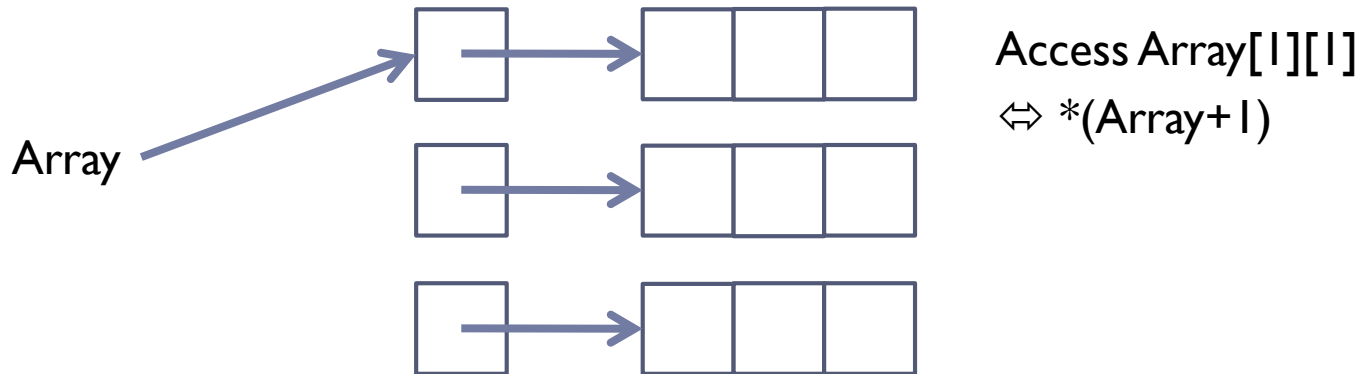
Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다 (??)

```
int Array[3][3];
```



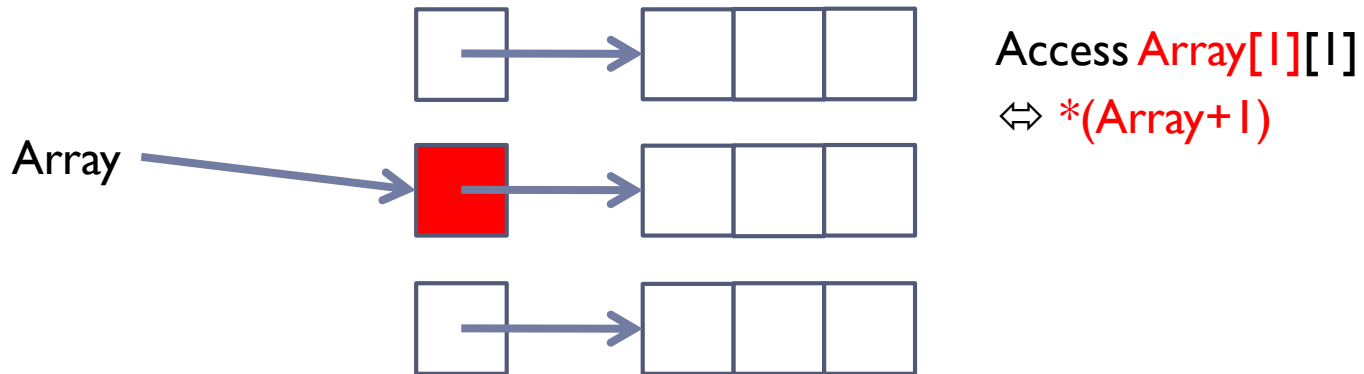
Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다 (??)

```
int Array[3][3];
```



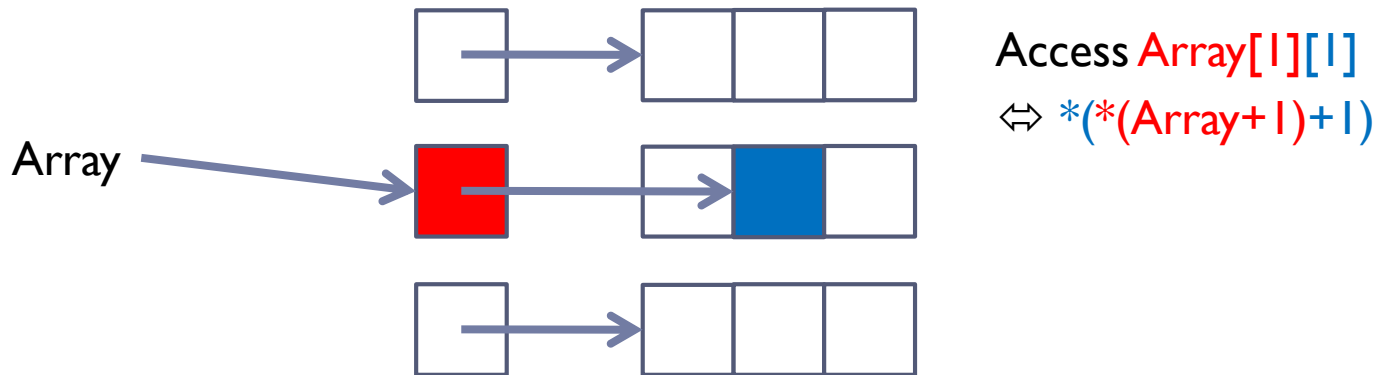
Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다 (??)

```
int Array[3][3];
```



Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다
- 실제로는 두 개의 방법이 섞임



Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다
- 실제로는 두 개의 방법이 섞임

```
int Array[3][3];
```



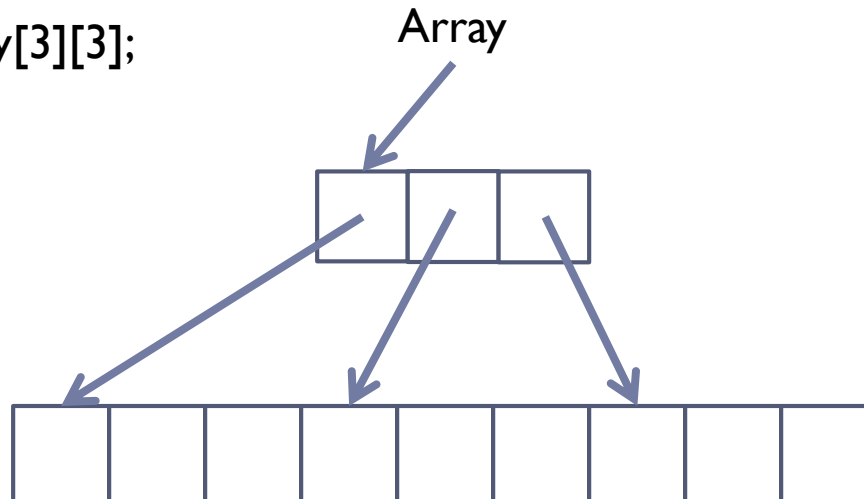
Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다
- 실제로는 두 개의 방법이 섞임

```
int Array[3][3];
```

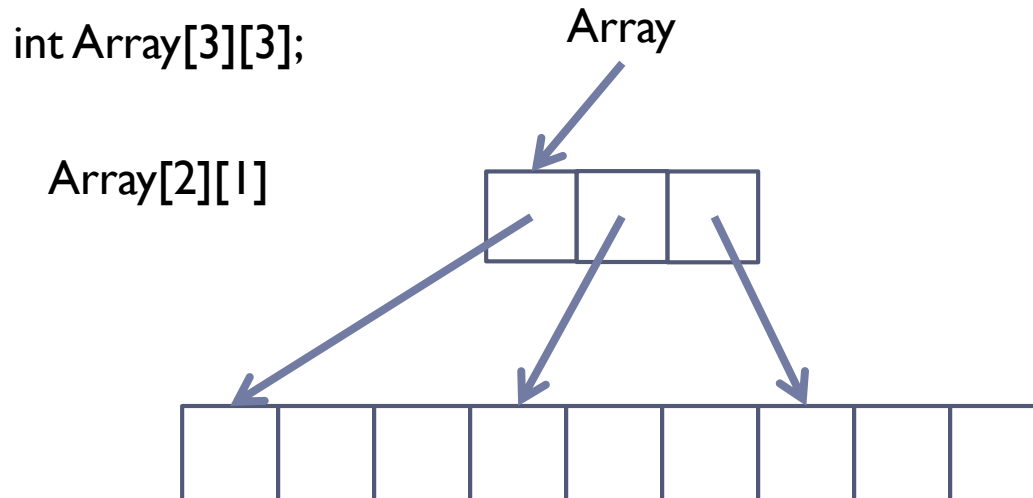


Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다
- 실제로는 두 개의 방법이 섞임



Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다
- 실제로는 두 개의 방법이 섞임

```
int Array[5][3];
```



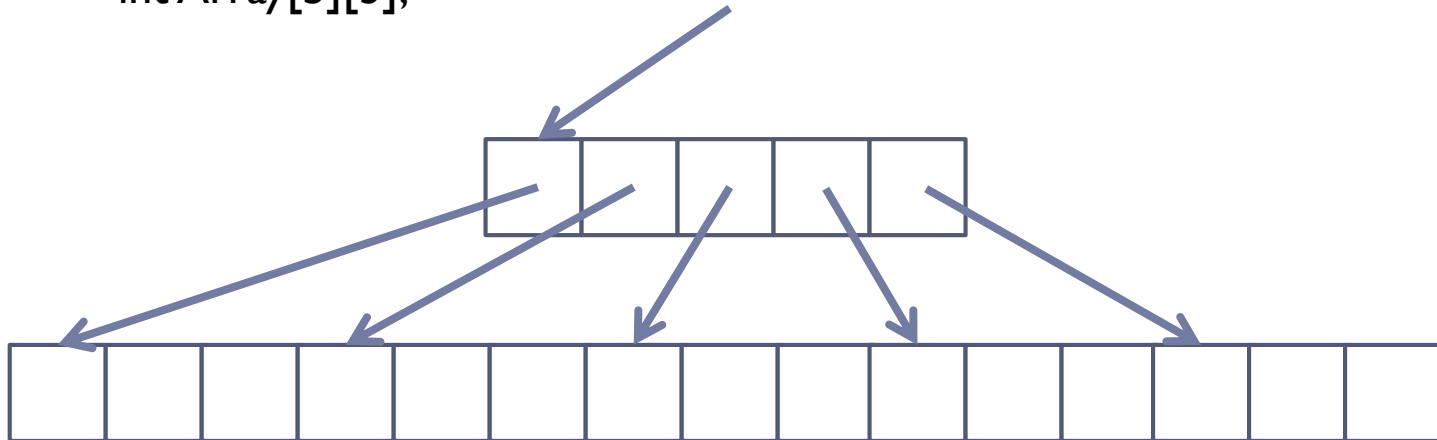
Pointer & Array

▶ 2차원 Array는?

▶ 2가지 방법으로 이해하기

- 방법 1. 1차원 배열을 만들어서 적당히 자른다
- 방법 2. 포인터를 가리키는 포인터를 만든다
- 실제로는 두 개의 방법이 섞임

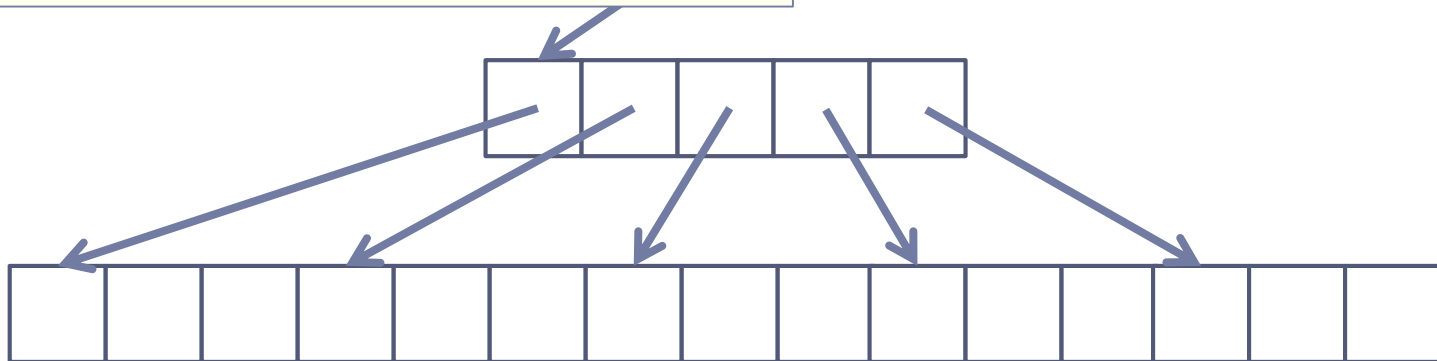
`int Array[5][3];` Array



Pointer & Array

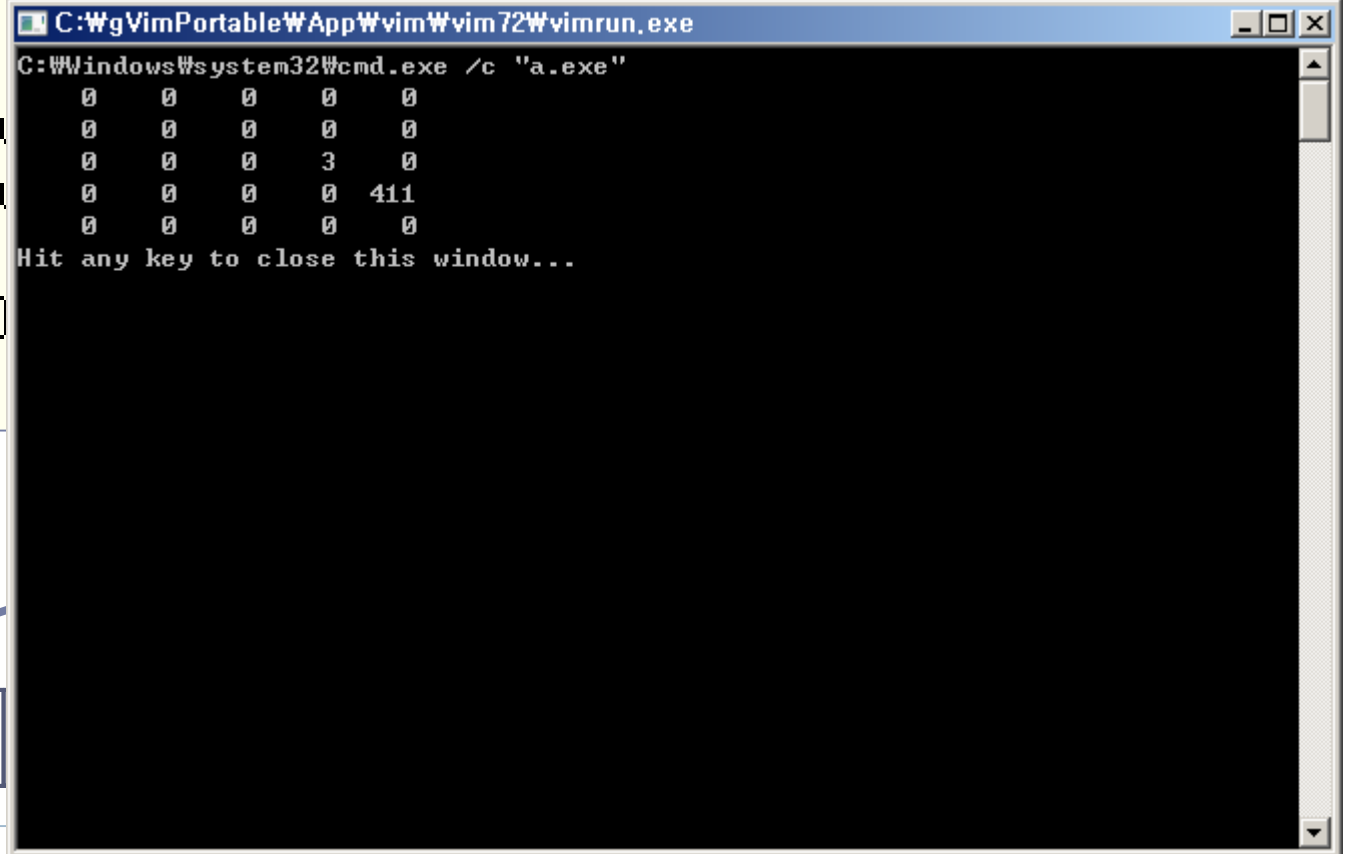
```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int Array[5][5];
    *((Array+2)+3) = 3;
    *(Array[3]+4) = 411;
    return 0;
}
```

적당히 자른다
터블 만든다



Pointer & Array

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int Array[5];
    *(Array+4) = 411;
    return 0;
}
```



```
C:\Windows\system32\cmd.exe /c "a.exe"
0 0 0 0 0
0 0 0 0 0
0 0 0 3 0
0 0 0 0 411
0 0 0 0 0
Hit any key to close this window...
```



Pointer & Array

▶ 3차원 Array는?

- ▶ 이런 쓸데없는 거 생각할 시간에 맛있는 걸 먹으러 가자
- ▶ `int*** Array;`
 - 멘붕이 오기 시작함

▶ Array 쓸 때 Array가 pointer인 것을 고려해야 함

- ▶ `*(*(Array+1)+2) = 3;` 이런 짓을 하라는 것이 아님
 - 머리에 총 맞지 않았으면 `A[1][2] = 3;` 이라 하는게 맞음
- ▶ 후에 이에 대해서 다시 이야기하도록 합니당

