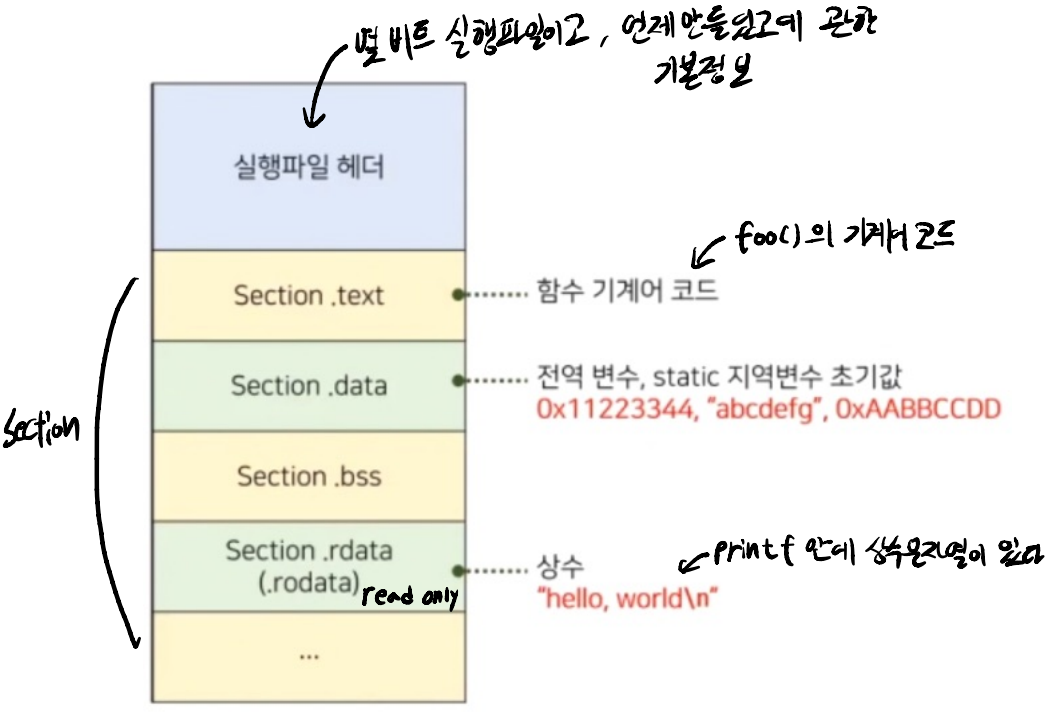
1-13

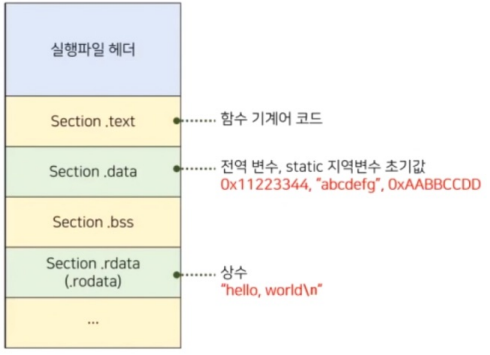


Window에서는 PE

Portable Excutable file format

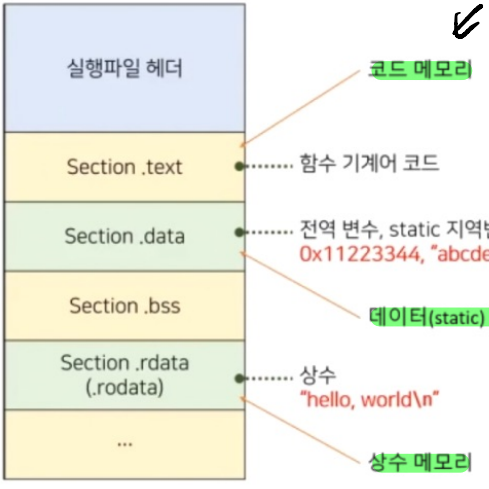
linux에서는 ELF

Excutable Linking format



↓

이 실행파일이 메모리로
올라간다

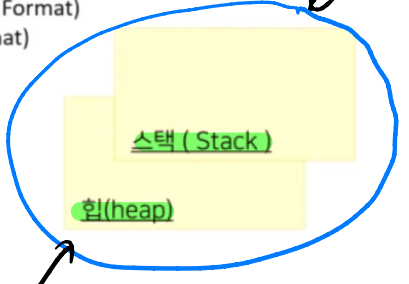


PE (Portable Executable File Format)
ELF(Executable Linking Format)

① .text Section에이
메모리로 올라가면
그걸 코드메모리라 부른다

② ← 파괴되지 않는다.
실행중에 크기가변경되지 않는다

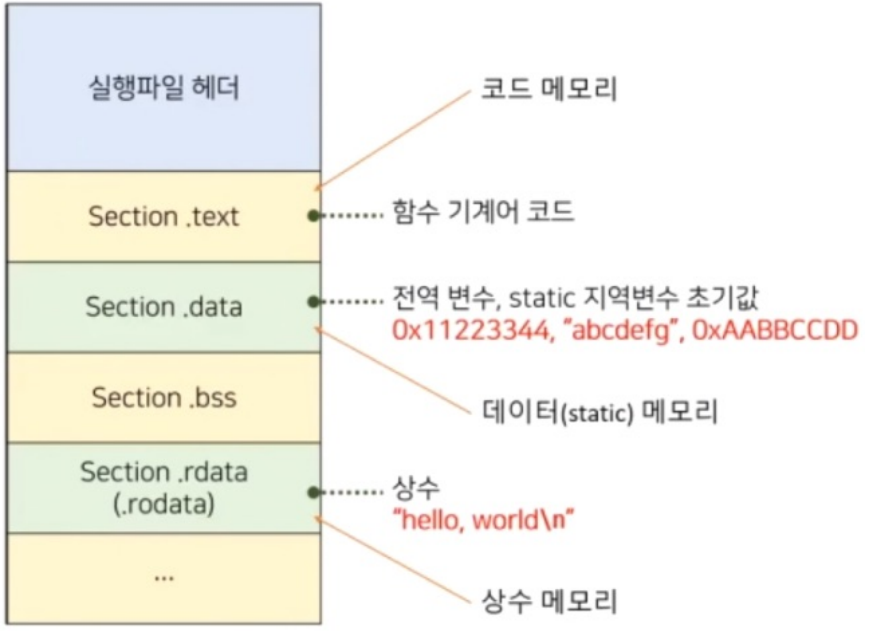
③ ← 시스템에 의해 할당된다.



④ stack : 지역변수
foo() 안에 있는
함수 호출 끝나면 파괴된다.
stack은 실행할때 크기가
계속 바뀐다.

⑤ malloc에 의한 할당

⑥ 코드메모리, 데이터메모리, 상수메모리는
실행파일이다.



PE (Portable Executable File Format)
ELF(Executable Linking Format)

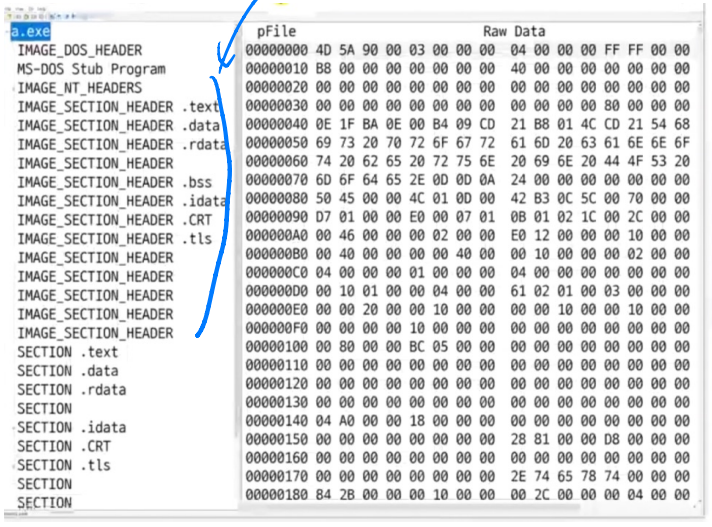
스택 (Stack)

힙(heap)

⦿ PEView 로 실행파일 포맷 조사

- http://wjradburn.com/software/
- PEView.exe 실행 후 "File -> Open" 메뉴 선택 후 first.exe 선택
- Linux 환경에서는 readelf 와 objdump 유틸리티로 확인 가능

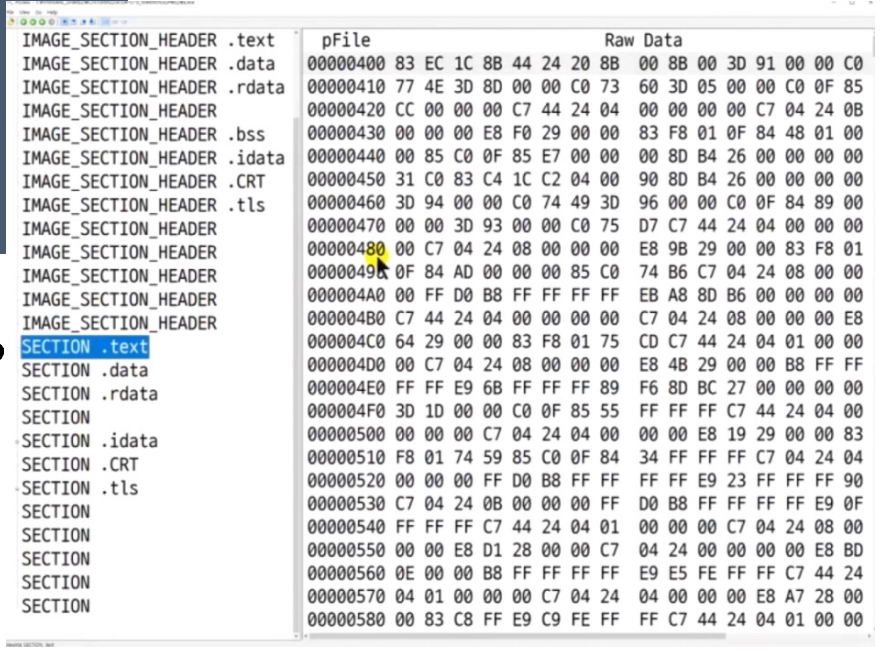
헤더



실행파일과 메모

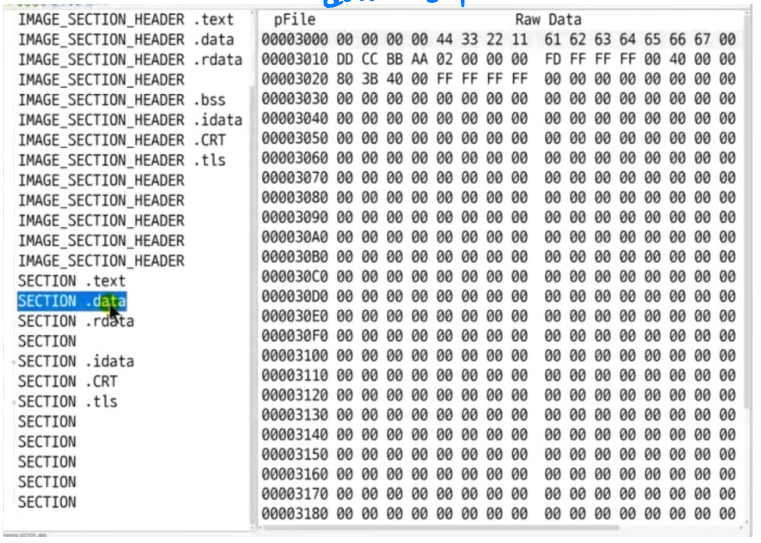
- PEView 로 실
- http://wjradb
- PEView.exe

```
> peview a.exe
>
```



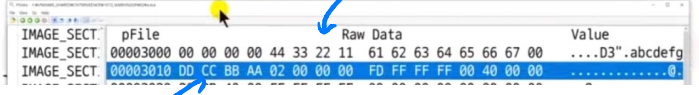
text 영역

data 영역

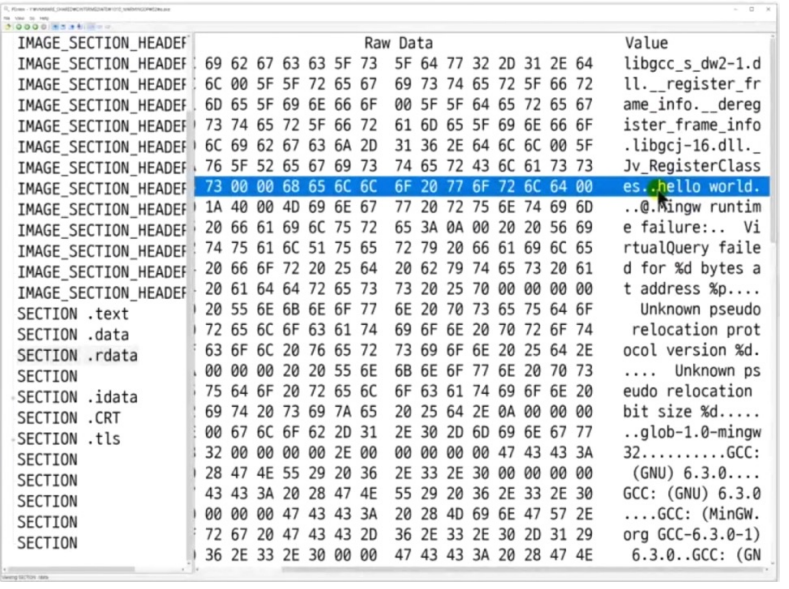



```
int  g1   = 0x11223344;
char g2[] = "abcdefg";

void foo(void)
{
    static int sx = 0xAABBCCDD;
```



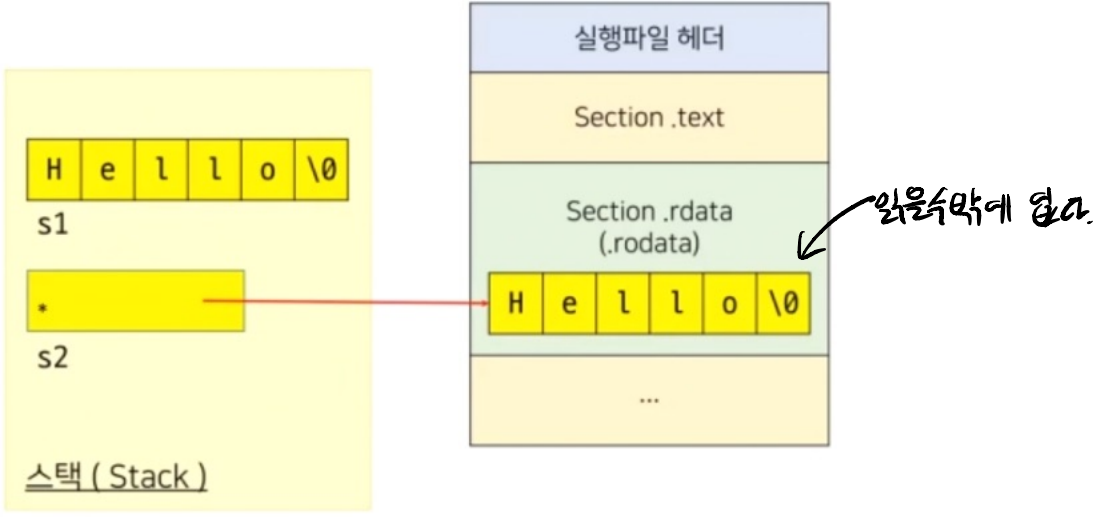
.rdata 영역



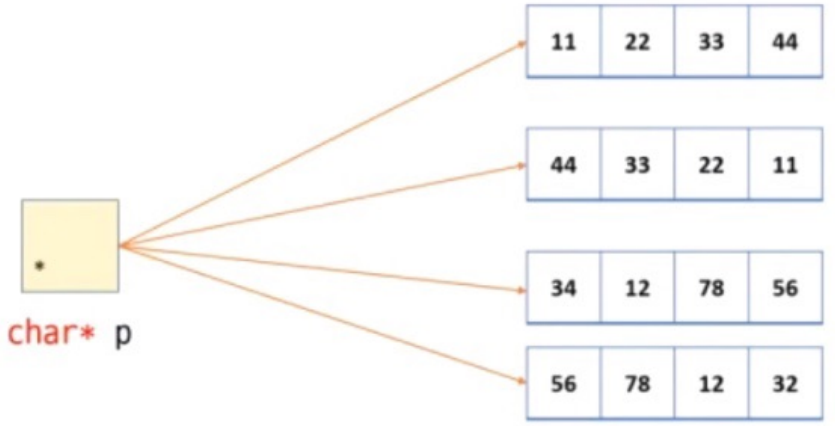
구조가

컴파일러마다 약간다를 수 있다.

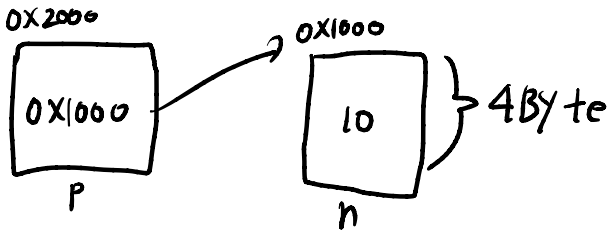
1-14

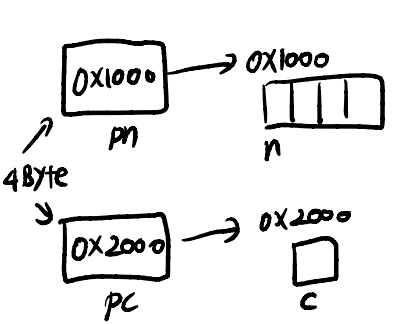

실행파일 헤더
Section .text
Section .rdata
(.rodata)
H e l l o \0
...
H e l l o \0
s1
s2
스택 (Stack)
읽음속박에 걸다.

1-16

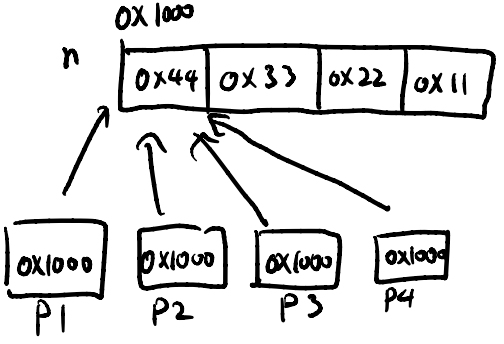


2-1 ①

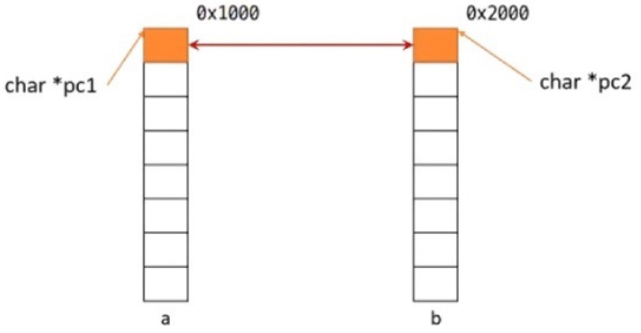

0X2000
0X1000
P
0X1000
10
n
4Byte


0X1000
pn
0X1000
n
4Byte
0X2000
pc
0X2000
c

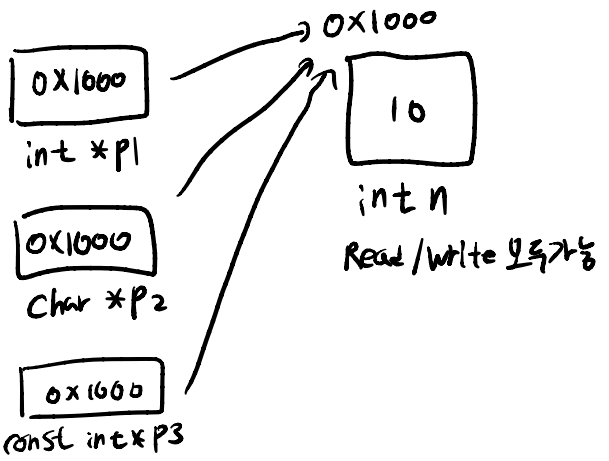
2-2

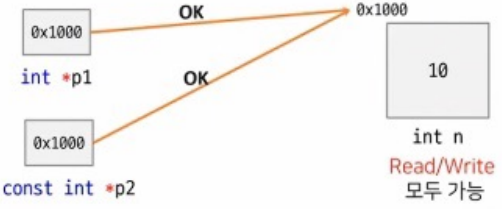

0X1000
n
0X44
0X33
0X22
0X11
0X1000
P1
0X1000
P2
0X1000
P3
0X1000
P4

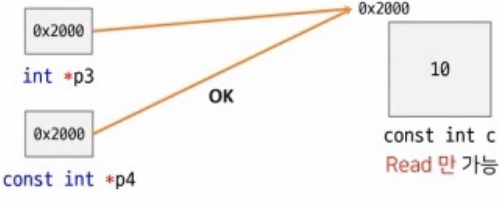
2-4


0x1000
0x2000
char *pc1
char *pc2
a
b

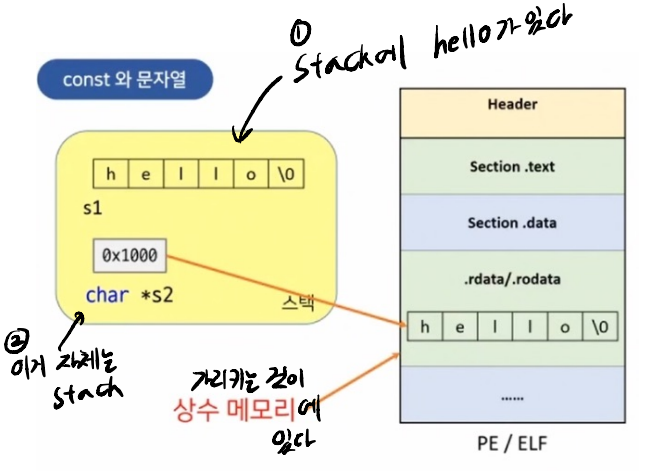
2-6


0x1000
0x1000
int *p1
10
int n
Read/Write 모두가능
0x1000
Char *p2
0x1000
const int *p3


0x1000
OK
0x1000
int *p1
OK
10
0x1000
int n
const int *p2
Read/Write
모두 가능


0x2000
0x2000
int *p3
OK
10
0x2000
const int c
const int *p4
Read 만 가능

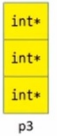
2-10


const 와 문자열
① Stack에 hello가 있다
h e l l o \0
s1
0x1000
char *s2
스택
② 이거 자체는 stack
가리키는 것이 상수 메모리에 있다
Header
Section .text
Section .data
.rdata/.rodata
h e l l o \0
.....
PE / ELF

3-1

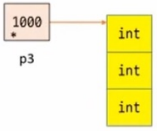
```
int *p3[3];
```

[] 연산자를 먼저 해석 한다.
p3는 크기가 3인 배열, 각 요소는 int*
32비트 환경에서 sizeof(p3) => 12



포인터 배열 (Array of pointers)

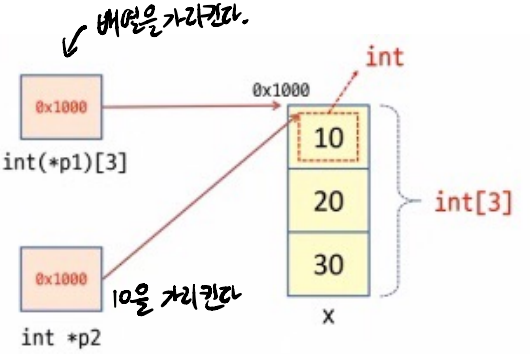
```
int (*p3)[3];
```



* 연산자를 먼저 해석한다.
p3는 포인터 변수, 배열(int[3]) 을 가리킨다.
32비트 환경에서 sizeof(p3) => 4

배열을 가리키는 포인터 (Pointer to array)

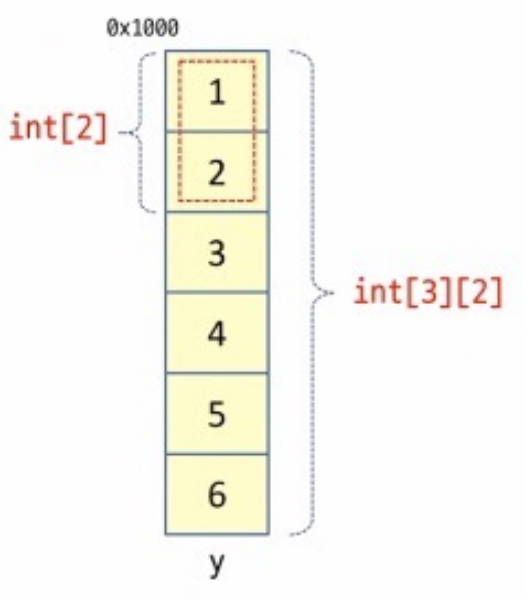
3-2

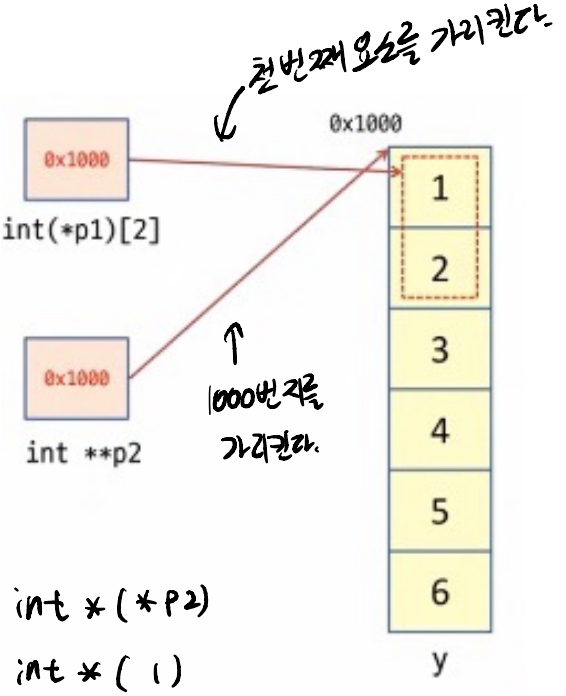


- "Array to pointer conversion" 문법

배열은 배열의 "첫 번째 요소의 주소"로 암시적 형 변환 된다.

3-3





int *(*p2)

int *(i)

결국 i번지를 가리킴