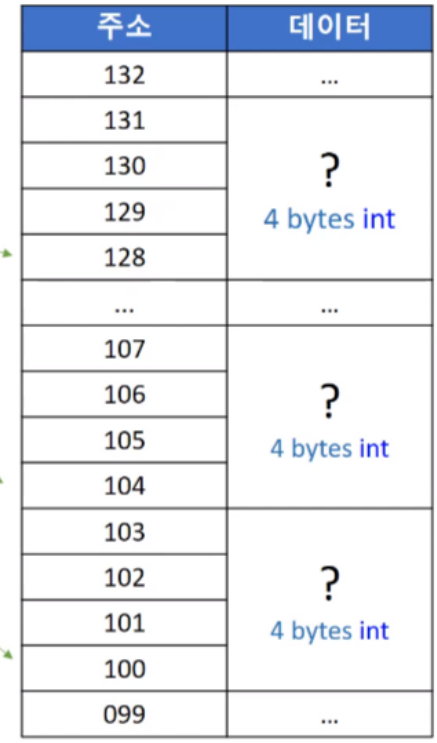
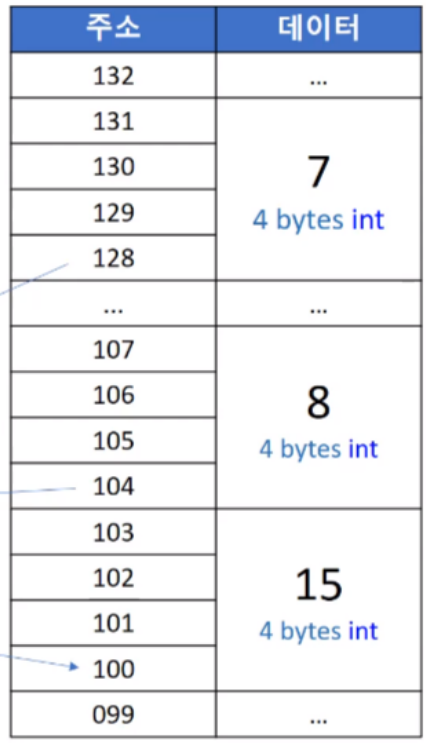
```
int a;
int b;
int c;
```



- 위의 메모리 주소들은 이해를 돕기 위한 쉬운 예시이며 실제와는 차이가 있습니다.
- 디버거를 이용해서 메모리를 직접 들여다보는 방법은 이후 강의에서 자세히 다룹니다.

```
int a;
int b;
int c;

a = 7;
b = 8;
c = a + b;
```



- 위의 메모리 주소들은 이해를 돕기 위한 쉬운 예시이며 실제와는 차이가 있습니다.
- 디버거를 이용해서 메모리를 직접 들여다보는 방법은 이후 강의에서 자세히 다룹니다.

```
int a = 7;
```

주소	데이터
132	...
131	7 4 bytes int
130	
129	
128	
...	...
107	128 4 bytes int*
106	
105	
104	

int형 변수의 주소를 저장하는 변수

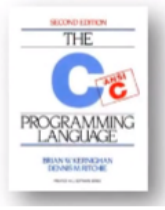
주소 연산자
Address-of operator

```
int *a_ptr = &a;
```

*: asterisk

x86에서는 4바이트
x64에서는 8바이트

포인터 변수에 저장되어 있는 값은 다른 변수의 주소이다.
따라서 직접적으로 데이터를 저장하기 보다는
다른 메모리 공간이나 그 공간에 저장되어 있는 데이터(또는 객체Object)를
(간접적으로) 가리키는 역할을 한다.



- 위의 메모리 주소들은 이해를 돕기 위한 쉬운 예시이며 실제와는 차이가 있습니다.
- 디버거를 이용해서 메모리를 직접 들여다보는 방법은 이후 강의에서 자세히 다룹니다.

```
int a = 7;
```

주소	데이터
132	...
131	8 4 bytes int
130	
129	
128	
...	...
107	128 4 bytes int*
106	
105	
104	

*a_ptr가 int 자료형이라는 의미

```
int (*a_ptr) = &a;
```

x86에서는 4바이트
x64에서는 8바이트

```
*a_ptr = 8;
```

간접 접근indirection
역참조dereferencing
방향 재지정redirection

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

- 위의 메모리 주소들은 이해를 돕기 위한 쉬운 예시이며 실제와는 차이가 있습니다.
- 디버거를 이용해서 메모리를 직접 들여다보는 방법은 이후 강의에서 자세히 다룹니다.

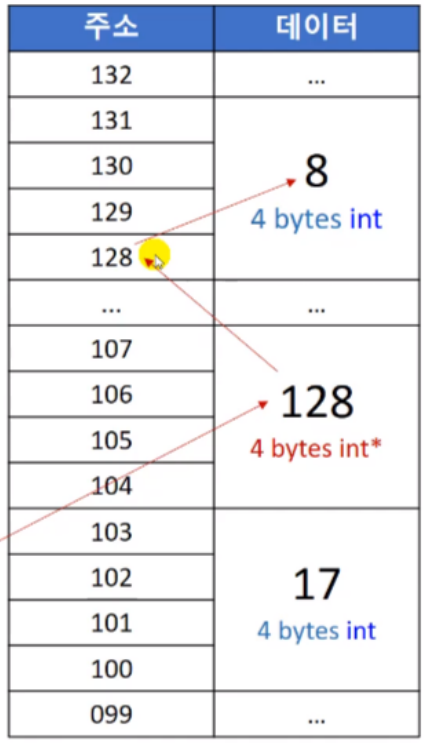
```
int a = 7;

int *a_ptr = &a;

*a_ptr = 8;

int c = 9;

c += *a_ptr;
```



x86에서는 4바이트
x64에서는 8바이트

- 위의 메모리 주소들은 이해를 돕기 위한 쉬운 예시이며 실제와는 차이가 있습니다.
- 디버거를 이용해서 메모리를 직접 들여다보는 방법은 이후 강의에서 자세히 다룹니다.