Pointers

summary

요약

순서

- Binary to Byte
- 주소
- 변수의 주소 확인
- Scanf에서 & 연산자의 활용
- 포인터의 2개의 연산자
- 포인터 변수와 사용 예
- 활동: 인간 포인터 연습
- void 타입과 변수 크기와의 관계
- 포인터에서의 캐스팅
- 포인터와 배열

Binary to Byte

- 1203의 2진수 표현

$$1203 = 1024 + 128 + 32 + 16 + 2 + 1$$

$$= 2^{10} + 2^{7} + 2^{5} + 2^{4} + 2^{1} + 2^{0}$$

2진수 => 0100 1011 0011

16진수 => 4 B 3

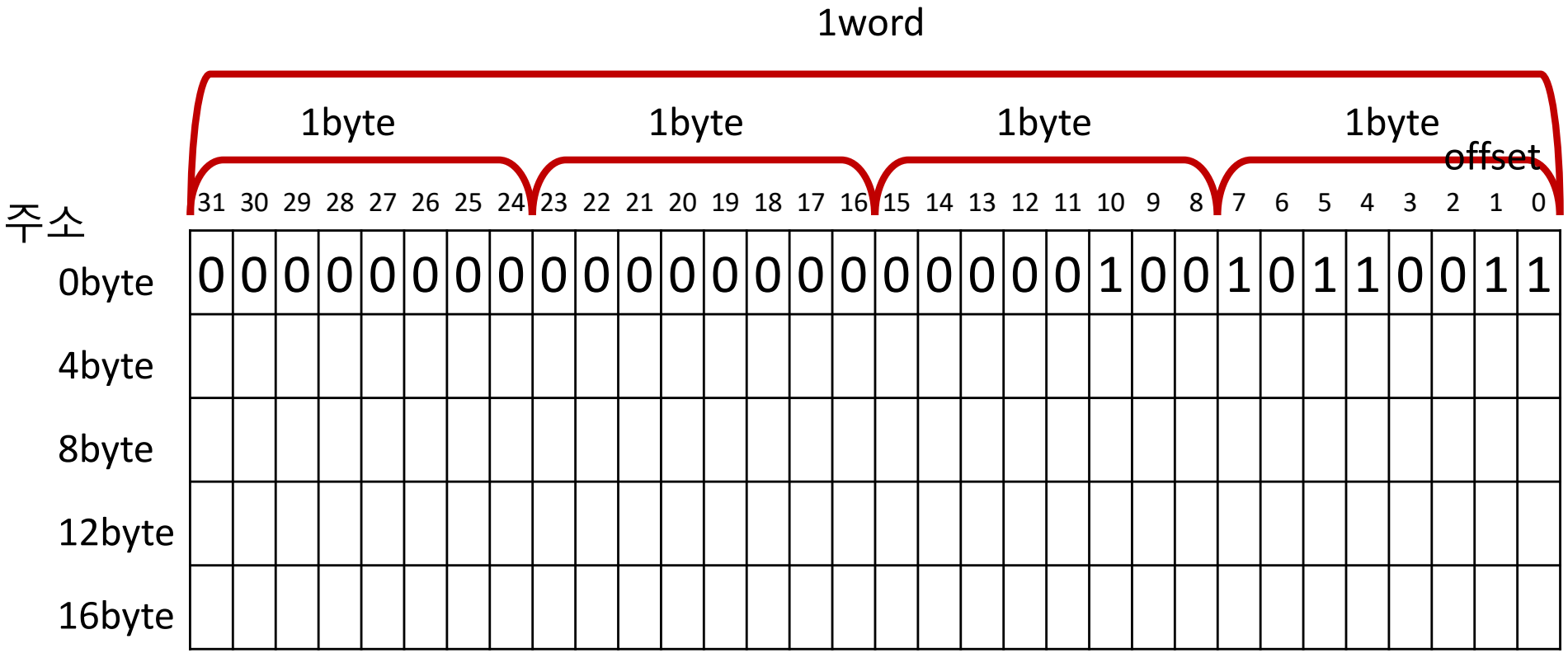
Binary(2진수)는 각 자리 또는 bit에 0과 1만 표현 가능
1203을 2진수로 표현하기 위해 11bit가 필요함

8 bit = 1 byte

위치	10	9	8	7	6	5	4	3	2	1	0
값	1	0	0	1	0	1	1	0	0	1	1

주소

- 32bit 주소 체계를 따르는 컴퓨터의 경우 정보의 표현
  - 64bit인 경우 한 줄에 64개 2진수 표현 가능

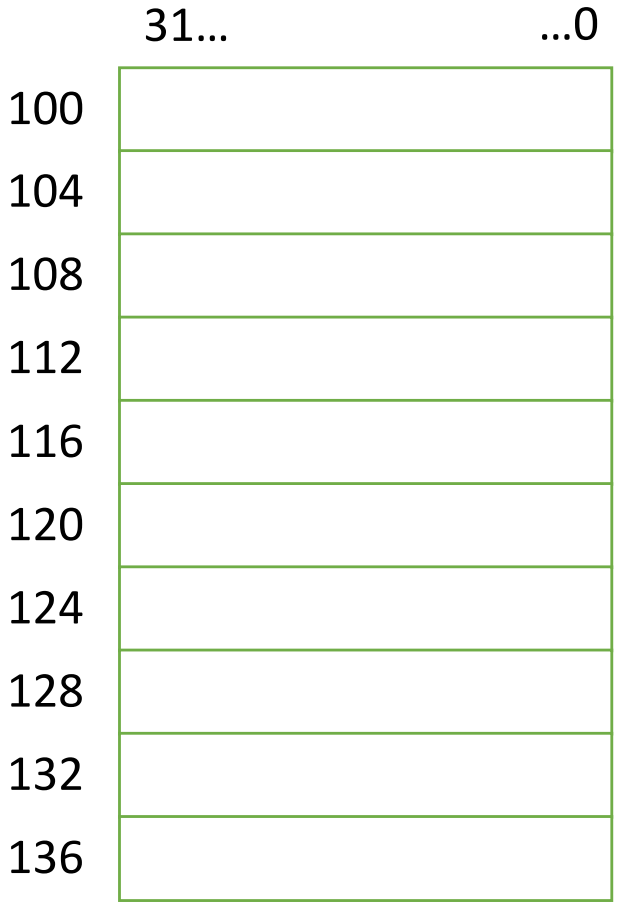


주소

- 모든 객체 (변수, 함수 등)은 고유의 주소를 갖음

예시

```
int a = 100;
int b = 200;
int c = 300;
```

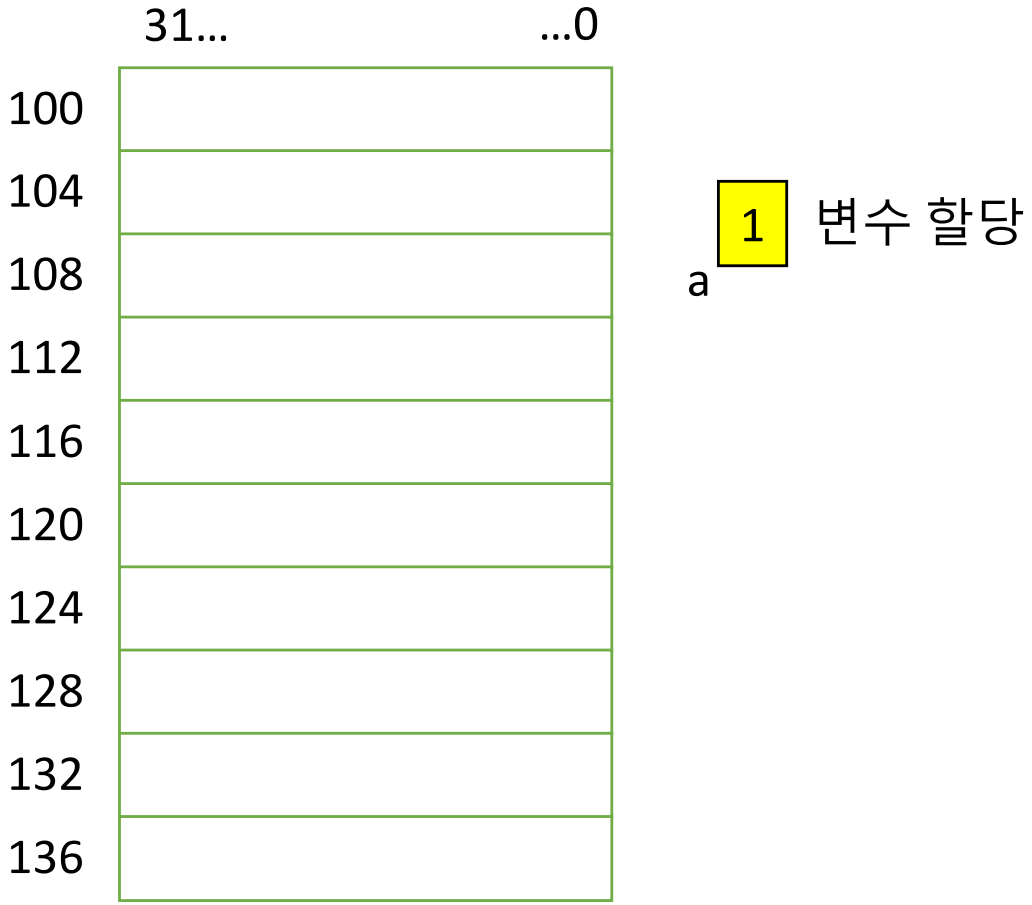


주소

- 모든 객체 (변수, 함수 등)은 고유의 주소를 갖음

예시

```
int a = 100;
int b = 200;
int c = 300;
```

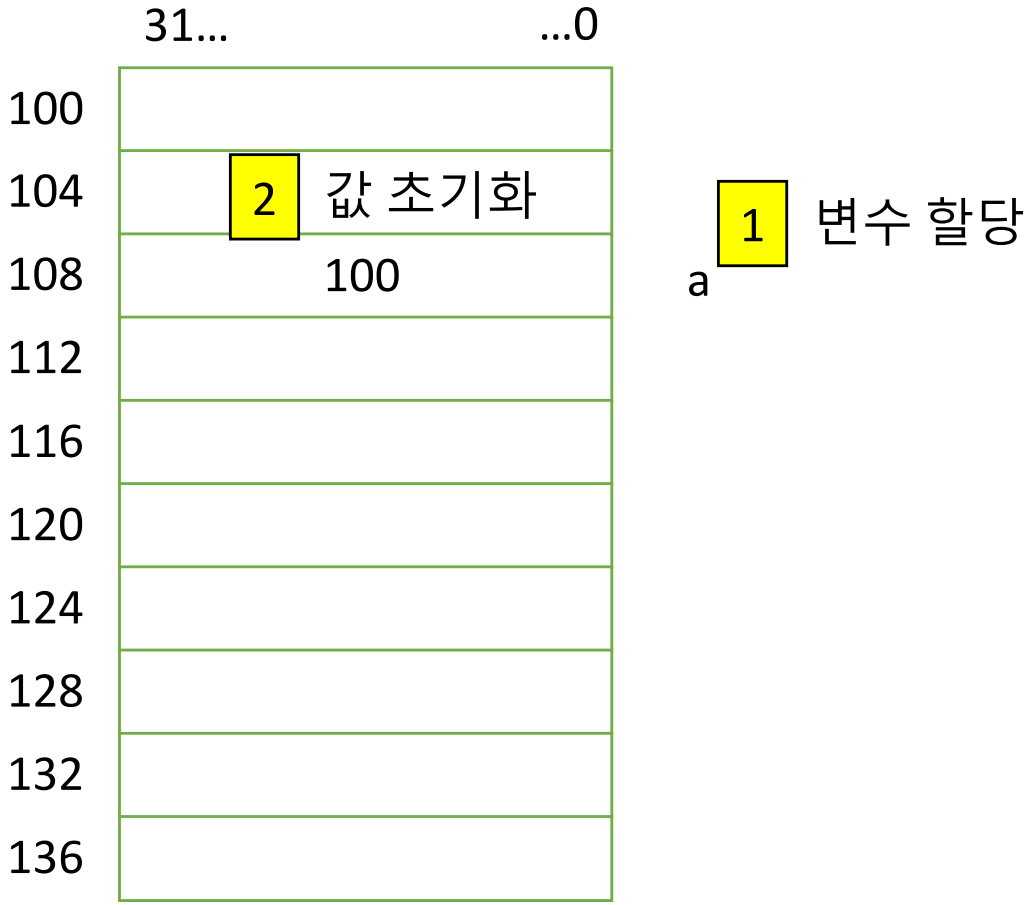


주소

- 모든 객체 (변수, 함수 등)은 고유의 주소를 갖음

예시

```
int a = 100;
int b = 200;
int c = 300;
```

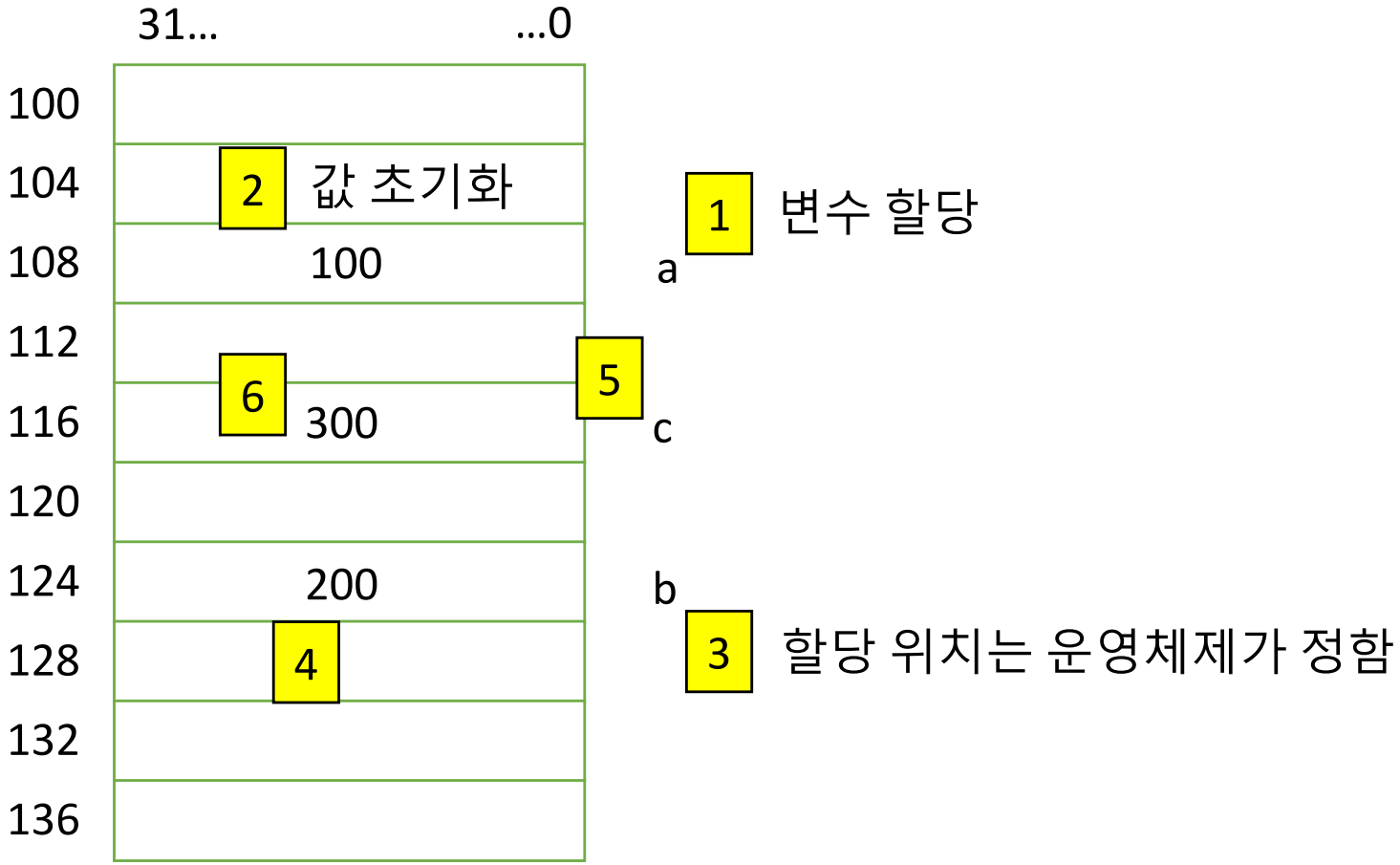


주소

- 모든 객체 (변수, 함수 등)은 고유의 주소를 갖음

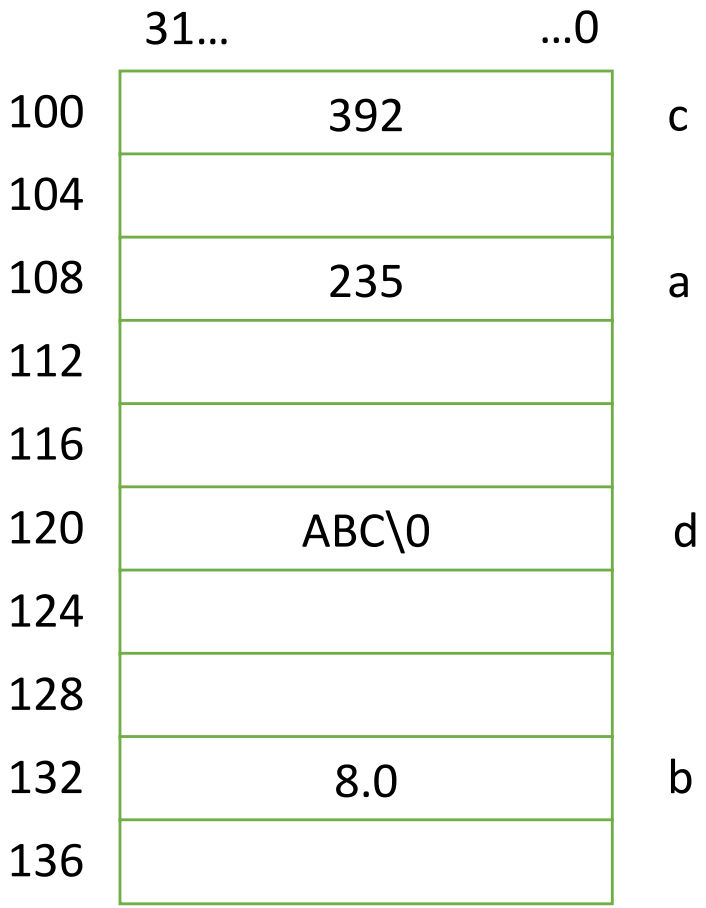
예시

```
int a = 100;
int b = 200;
int c = 300;
```



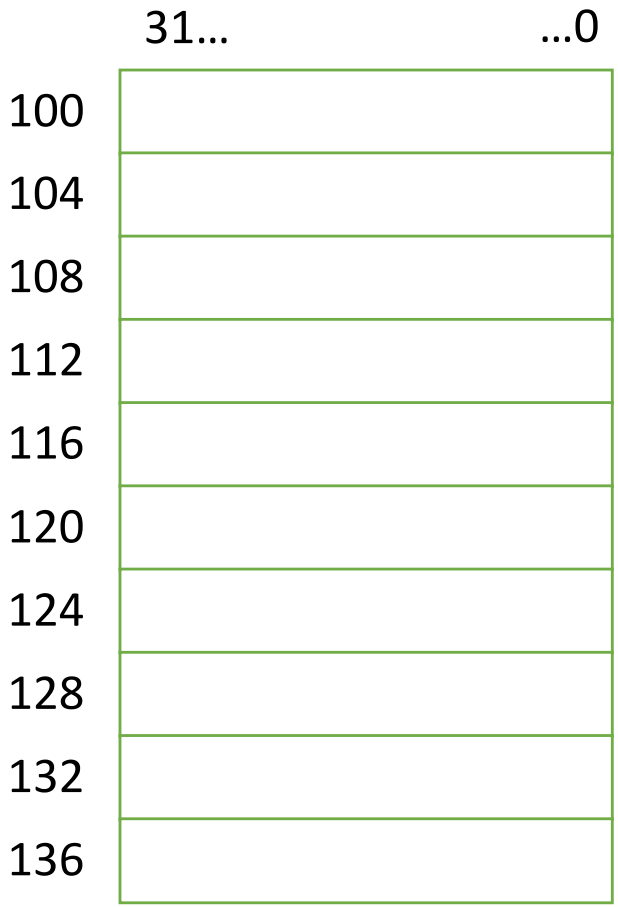
문제

- 다음의 메모리 상태를 참고로, 변수 초기화 문장을 작성



문제

- 다음의 메모리 초기화 문장을 토대로 메모리 배치도 작성




```
int a = 1;
float b = 39.2;
double c = 400201;
char d[5] = { 1, 2, 3, 4, 5};
```

변수의 주소 확인

```
#include <stdio.h>
int main(){
    int c[5] = { 3, 2, [3] = 8, [4] = 9};
    for (int k = 0; k < 5; k++)
        printf("value of c[%d] = %d; address is %p\n", \
                k, c[k], &c[k]);
    return 0;
}
```

16진수 주소 형식

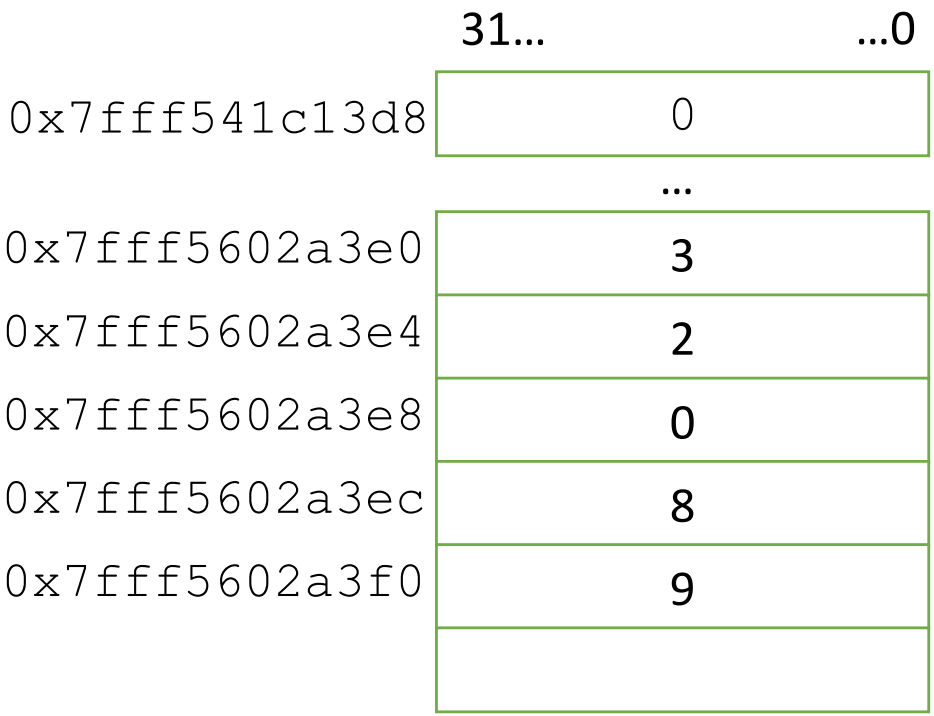
주소 연산자: 변수의 저장된 주소

1. 변수 c 할당
2. 변수 c 초기화
3. 변수 k 할당
4. 변수 k 초기화
5. c[k]의 값과 주소 출력
6. 변수 k의 값 증가

변수의 주소 확인

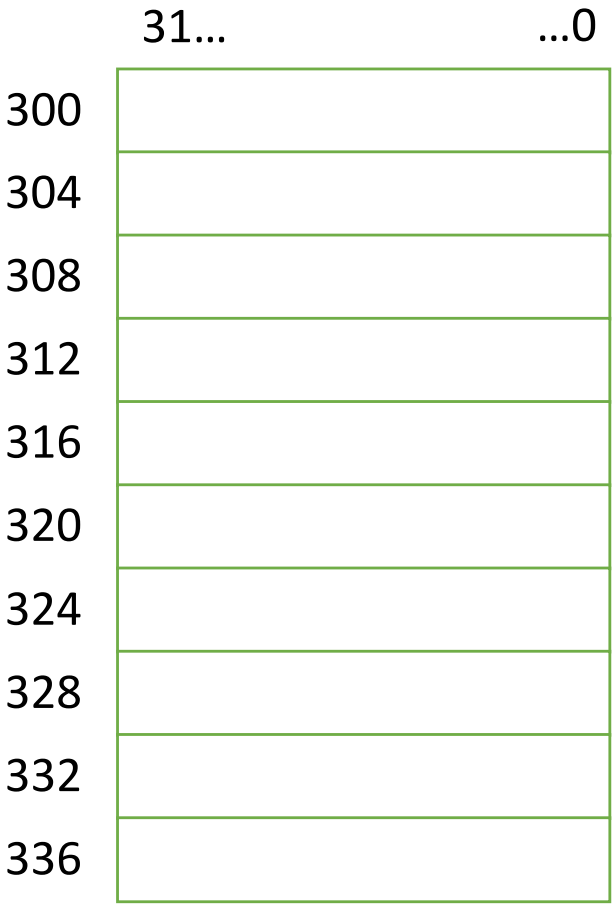
실행 결과

```
value of c[0] = 3; address is 0x7fff5602a3e0
value of c[1] = 2; address is 0x7fff5602a3e4
value of c[2] = 0; address is 0x7fff5602a3e8
value of c[3] = 8; address is 0x7fff5602a3ec
value of c[4] = 9; address is 0x7fff5602a3f0
```



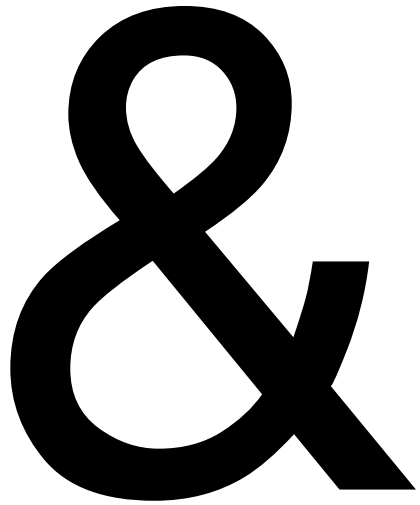
Scanf에서 &연산자의 활용

```
1   float a; // 4byte
2   scanf("%f", &a);
```



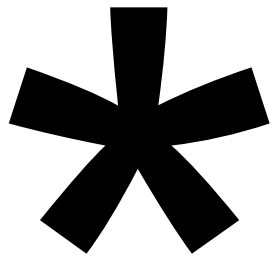
포인터의 2개의 연산자

포인터 변수
저장하는 것은 주소



참조 연산자(reference operator)

“~의 주소”로 이해



역참조 연산자(dereference operator)

“~가 가리키는 값”

으로 이해

*Dereference는 역참조 또는 간접참조라 부름

포인터 변수

- 선언 – 역참조 연산자를 사용하여 주소를 저장하는 변수임을 표시

타입 / 역참조 연산자 / 포인터 변수명

```
double *pointer_variable;
```

- 할당 – 참조 연산자를 사용하여 주소를 포인터 변수에 저장

포인터 변수 / 참조 연산자 / 변수명

```
pointer_variable = &variable;
```

- 읽기 – 역참조 연산자를 사용하여 저장된 주소가 가리키는 곳의 값을 가져옴

역참조 연산자 / 포인터 변수명

```
*pointer_variable;
```

포인터 변수의 사용 예

- 다른 변수의 주소를 포인터 변수에 저장하는 방법

- Type I – 포인터 변수 선언시 주소 할당

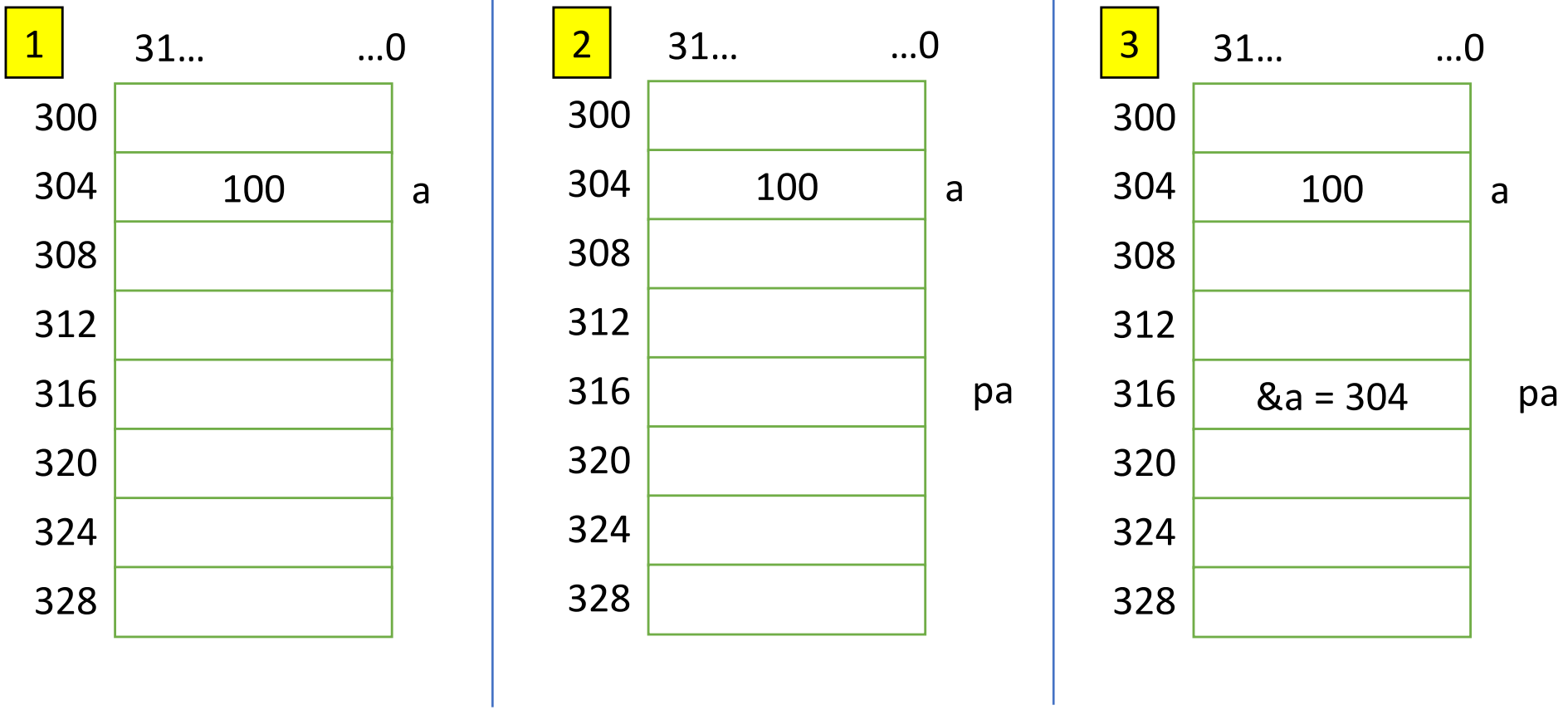
```
1   int a = 100;    // a 의 주소는 256이라 가정
2   int *pa = &a;   // 포인터 변수 선언 및 주소 할당
```

- Type II – 일반적인 주소 할당 방법

```
1   int a = 100;    // a 의 주소는 256이라 가정
2   int *pa;        // 포인터 변수 선언
3   …               // 다른 문장들 실행
4   pa = &a;        // 포인터 변수에 a의 주소 저장
```

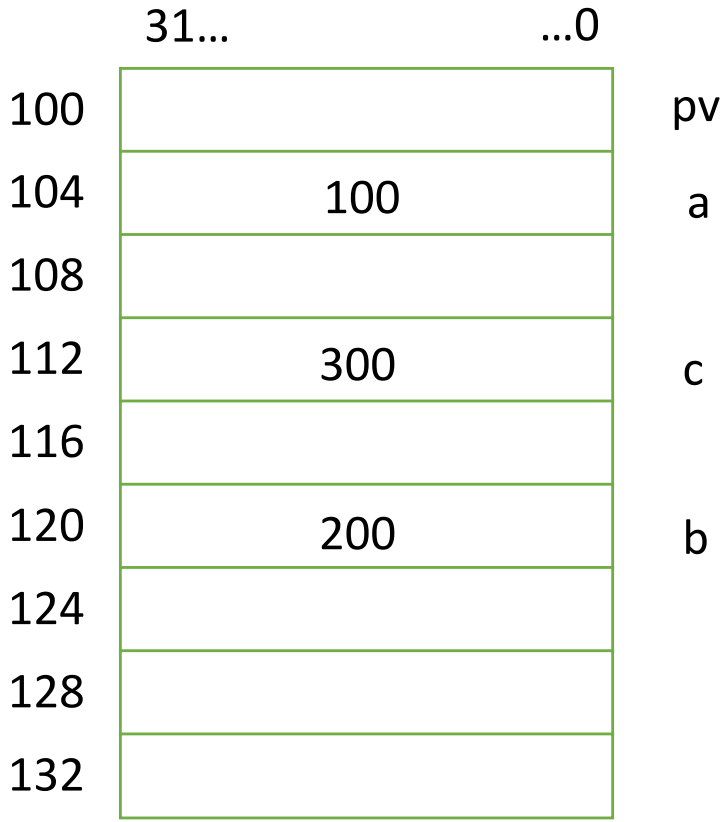
포인터 변수의 사용 예

```
1   int a = 100;      // a 의 주소는 256이라 가정
2   int *pa;          // 포인터 변수 선언
3   …                 // 다른 문장들 실행
4   pa = &a;          // 포인터 변수에 a의 주소 저장
```



예시

```
1   int a = 100, b = 200, c = 300;
2   int *pv;
3   printf("%p\n", pv = &a);
4   printf("%p\n", pv = &b);
5   printf("%p\n", pv = &c);
```



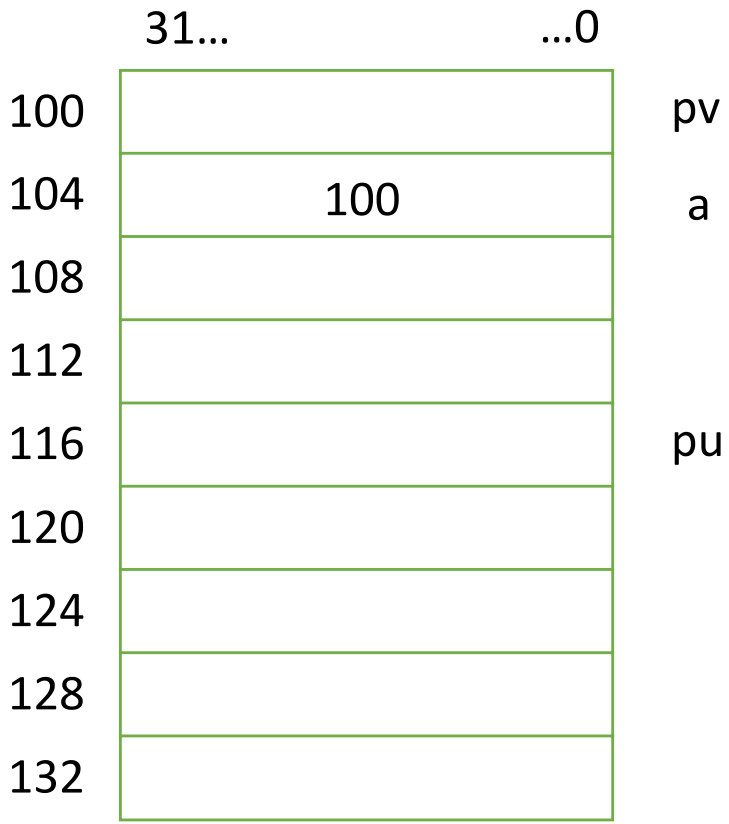
1 pv 값은 a의 주소 104

2 pv 값은 b의 주소 120

3 pv 값은 c의 주소 112

예시

```
1   int a;
2   int *pv = &a, *pu = &a);  // pv와 pv는 a의 주소 104를 저장
3   printf("%d\n", *pu = 400);
4   printf("%d, %d\n", a, *pv);
```



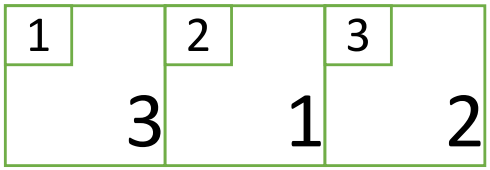
1 pu가 가리키는 a에 400 저장

2 pv가 가리키는 a의 값 400 읽음

3 pv 값은 c의 주소 112

활동: 인간 포인터 연습

- 전체 인원을 반으로 나누어 2 팀으로 나눔
- 제비뽑기를 통해 모든 사람은 두 개의 숫자를 갖음
  - 그 중 한 숫자는 주소 다른 한 숫자는 값임
  - 주소는 1에서 부터 40까지, 값도 1에서 부터 40까지임
  - 주소와 값이 동일한 수이면 왼쪽 사람의 값과 교환
- 1번 주소부터 40번 주소까지 모두 순회하는 데 걸리는 시간을 비교 함
- 1번 주소의 사람은 자신의 값을 활용하여 역참조(*연산)를 하여 값을 찾아 부름
- 역참조된 값을 갖고 있는 주소로 반복
  - 예시:

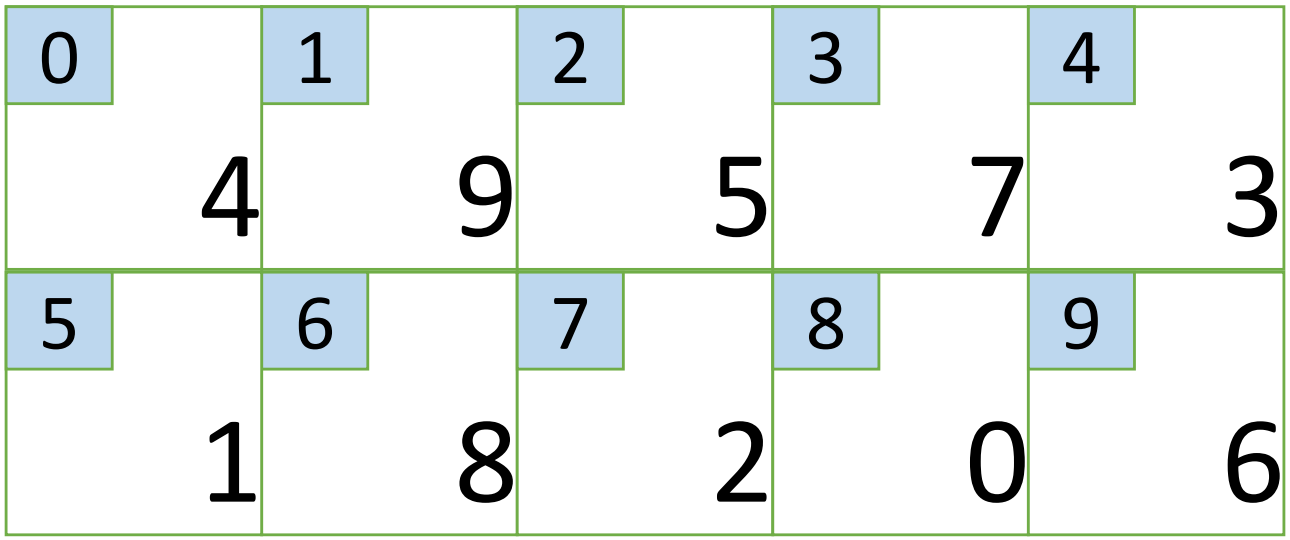


포인터 연습 문제

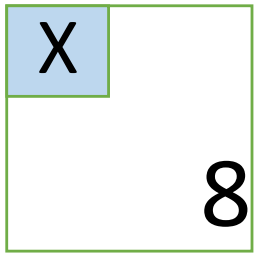
```
int a; // a의 값은 0, 주소는 8
int *p = &a; // p의 주소는 8
printf("%p ", *p); // 0를 출력함
printf("%p ", *******p); // 1을 출력함
printf("%p ", ****p); // 4를 출력함
```

문제:

다음과 같이 각 주소와 값이 설정되어 있다고 할 때,
포인터만을 써서 자신의 번호의 뒷자리를 출력하도록 해보자.



포인터 변수 p



포인터 연습 문제

```
int a; // a의 주소는 1
int *p = &a; // p의 주소는 20
*p = 12;
```

주소	값

문제:

1	2	3	4	5
12	3	4	11	14
6	7	8	9	10
20	17	15	10	19
11	12	13	14	15
18	9	8	16	13
16	17	18	19	20
7	2	6	5	1

포인터 변수 p

X
20

void 타입과 변수 크기와의 관계

void는 타입 미지정

필요에 따라 void 타입 변수를
다른 타입으로 캐스팅(casting)하여 사용

- 캐스팅 방법 (새로운 타입) 변수명

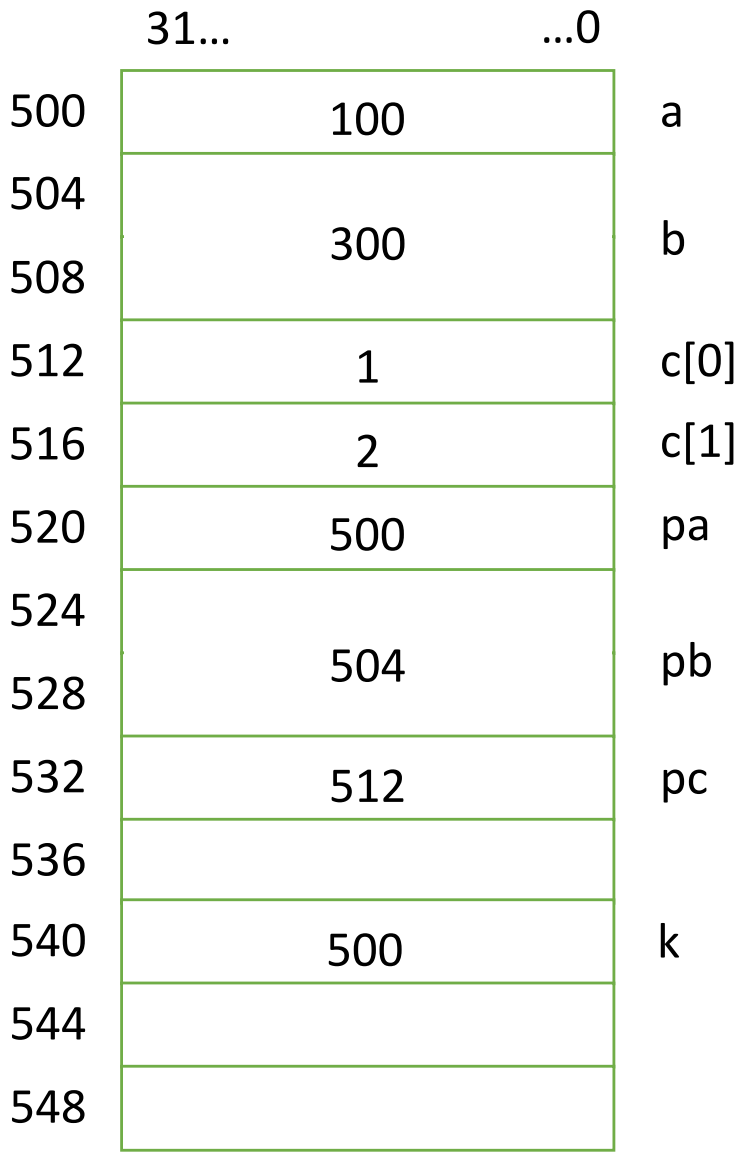
```
float a = 3.9, b = 7.2
int sum;
sum = (int)b % (int)a;
```

float형 변수 a와 b를 int형으로 캐스팅
int형 변수 sum에 결과를 저장

포인터에서의 캐스팅

```
int a = 100;
double b = 300;
int c[2] = {1, 2};

int *pa = &a;
double *pb = &b;
int *pc = &c[0];

void *k;

k = &a;
```

1 k 값은 a의 주소 500

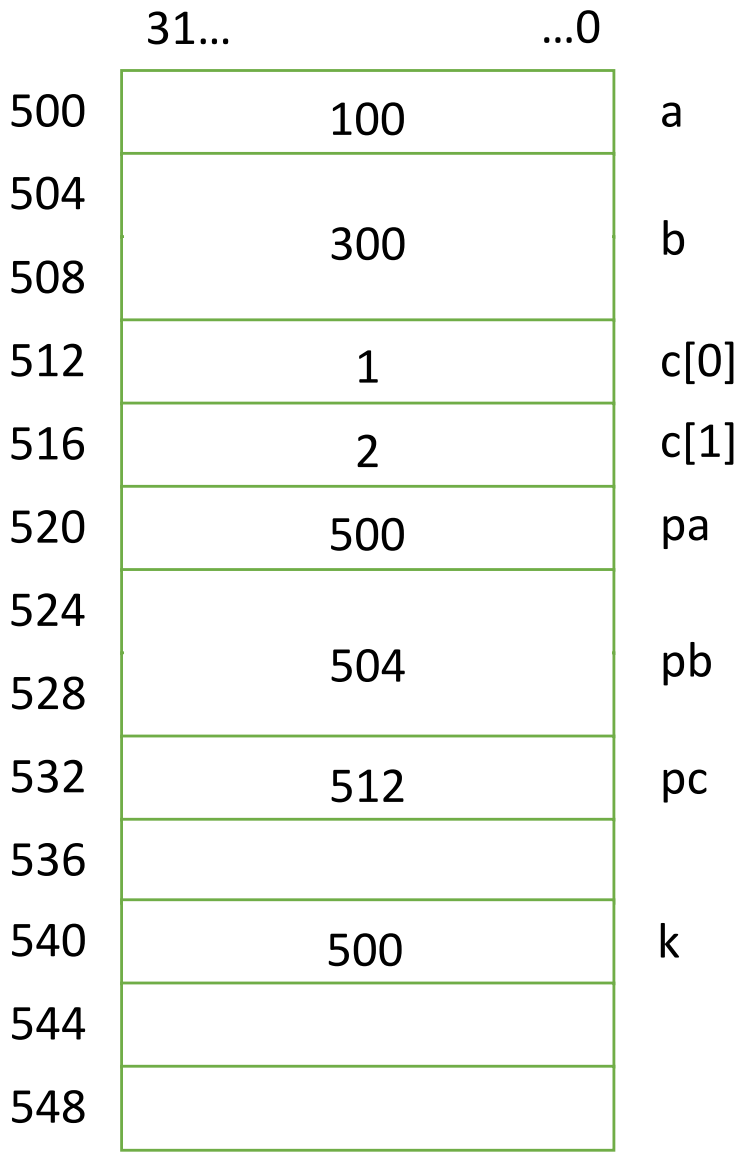


포인터에서의 캐스팅

```
int a = 100;
double b = 300;
int c[2] = {1, 2};

int *pa = &a;
double *pb = &b;
int *pc = &c[0];

void *k;

k = &a;
printf("%d\n", *(int*)k);
```

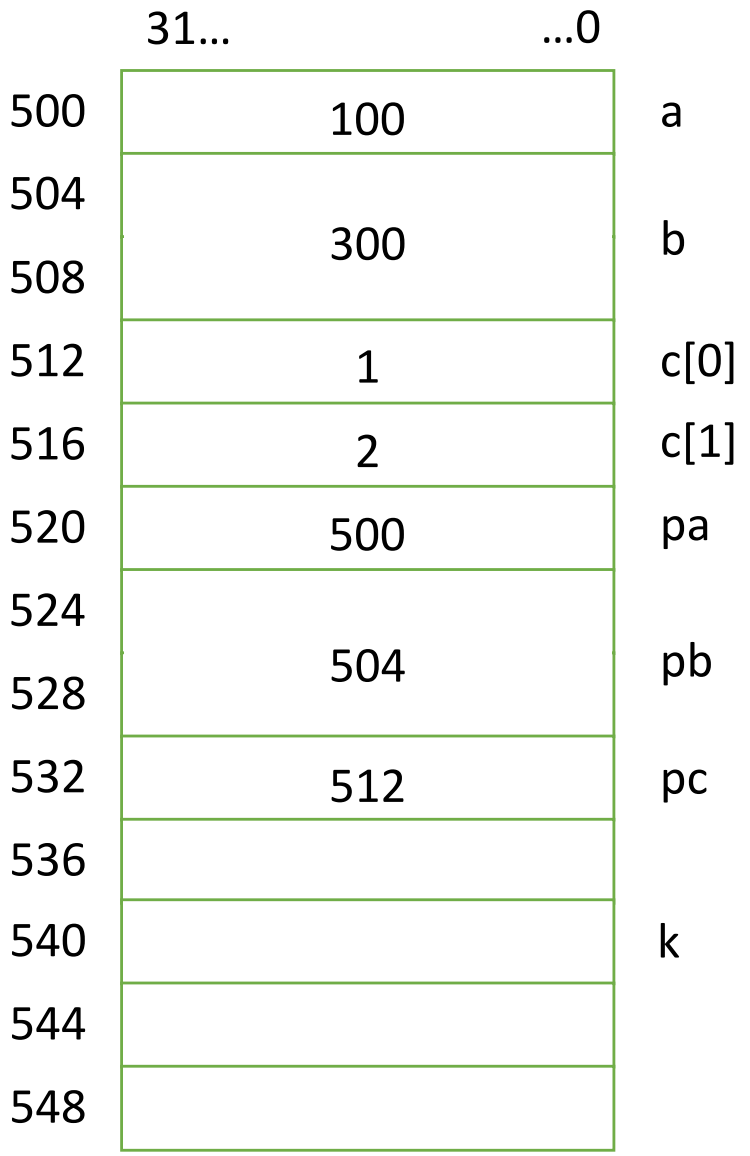
1 k 값은 a의 주소 500

2 읽을 바이트의 길이는 int 타입의 길이 4바이트



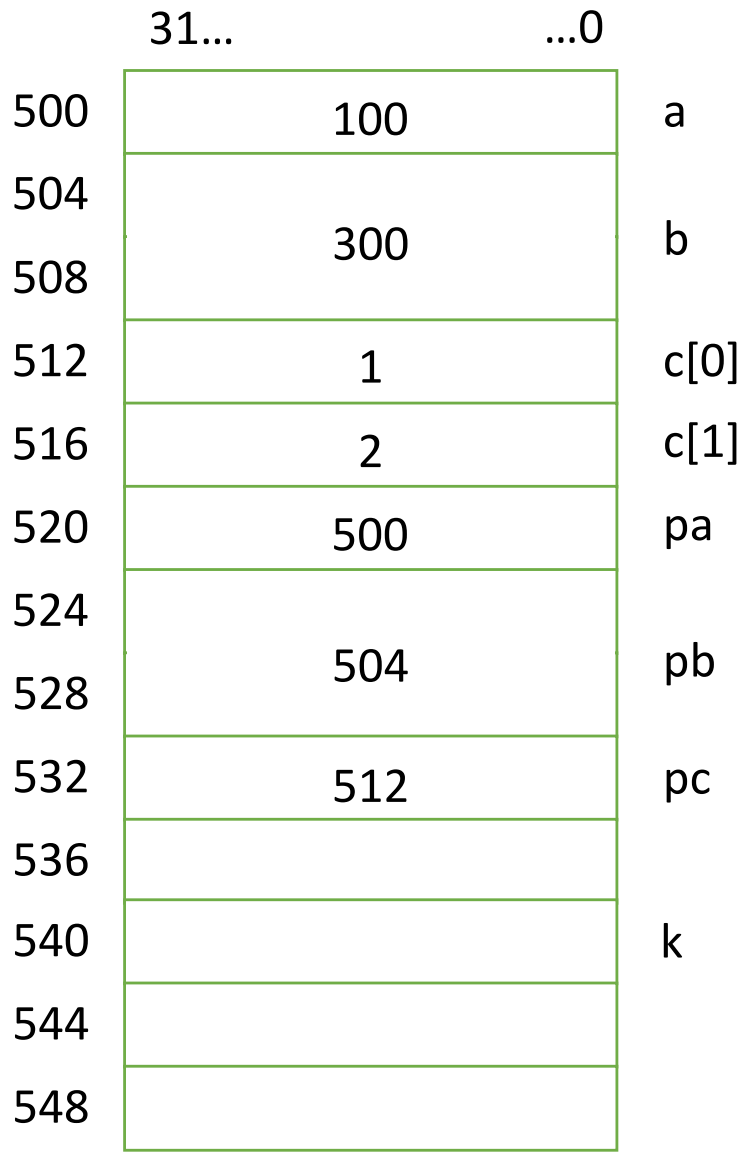
포인터에서의 캐스팅

```
int a = 100;
double b = 300;
int c[2] = {1, 2};

int *pa = &a;
double *pb = &b;
int *pc = &c[0];

void *k;

k = &a;
printf("%d\n", *(int*)k);

k = &b;
printf("%d\n", *(double*)k);
```



포인터에서의 캐스팅

```
int a = 100;
double b = 300;
int c[2] = {1, 2};

int *pa = &a;
double *pb = &b;
int *pc = &c[0];

void *k;

k = &a;
printf("%d\n", *(int*)k);

k = &b;
printf("%d\n", *(double*)k);

k = &c[0];
printf("%d\n", *(int*)k);
```



포인터와 배열

```
int c[4] = {1, 2, 3, 4};
int *pc;
```

포인터를 이용한 배열 접근

```
pc = &c[0]; // c[0]의 주소
pc = &c[1]; // c[1]의 주소
pc = &c[2]; // c[2]의 주소
pc = &c[3]; // c[3]의 주소
```

포인터 연산을 이용한 배열 접근

```
pc; // c[0]의 주소
pc+1; // c[1]의 주소
pc+2; // c[2]의 주소
pc+3; // c[3]의 주소
```

시작 주소+(변수의 타입)*배열인덱스
c[2]의 주소 = 512 + 4*2 = 520