

Arrays and Pointers and Structures

Data Structures and Algorithms

Arrays

Array, 배열

- 하나 이상의 _____ 하나의 변수에 저장하는 데이터 타입
 - 선언
 - 값 초기화와 할당 (다양한 방법이 있음, 자료 참고)
 - 인덱스
 - 활용
 - 배열의 차원(dimension)

Array, 배열: 선언

- 구성 요소:
 - 타입: int, float, char, double, long, 등
 - 변수명: 변수이름 짓는 제약 조건을 따름
 - 배열 크기: 임의의 양수, 저장하고자 하는 값의 개수 만큼

Array, 배열: 초기화와 할당 (상세내용은 자료참고)

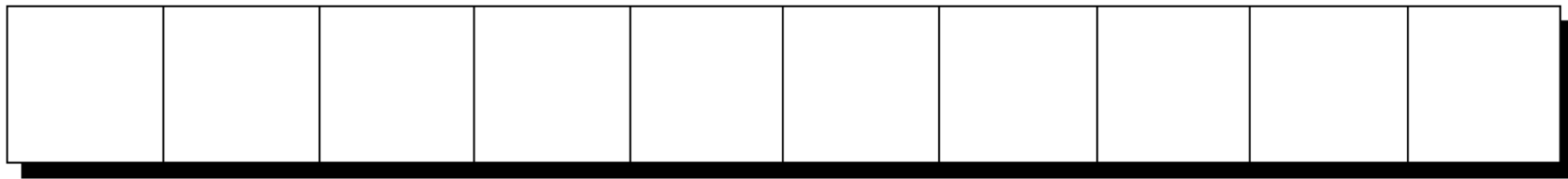
```
int scores[5];
```

Array, 배열: 인덱스

- 배열의 물리적 표현

`int a[10];` 10개의 값을 저장할 수 있는
a라는 이름의 배열

배열의 물리적 표현




중요: _____에서 인덱스 시작
값이 저장된 위치의 주소

Array, 배열: 활용

- 바코드 계산 프로그램



```
int i1, i2, i3, i4, i5;  
scanf("%1d%1d%1d%1d%1d", &i1, &i2, &i3, &i4, &i5);  
first_sum = i1 + i2 + i3 + i4 + i5;
```



Array, 배열: 활용



- 바코드 계산 프로그램

```
int i1, i2, i3, i4, i5;  
scanf("%1d%1d%1d%1d%1d", &i1, &i2, &i3, &i4, &i5);  
first_sum = d + i2 + i4 + j1 + j3 + j5;  
  
int i[5];  
scanf("%1d%1d%1d%1d%1d", &i[0], &i[1], &i[2], &i[3], &i[4]);  
first_sum = d + i[1] + i[2];  
  
int i[5];  
for(cnt = 0; cnt < 5; cnt++)  
    scanf("%1d", &i[cnt]);  
first_sum = d + i[1] + i[2];
```

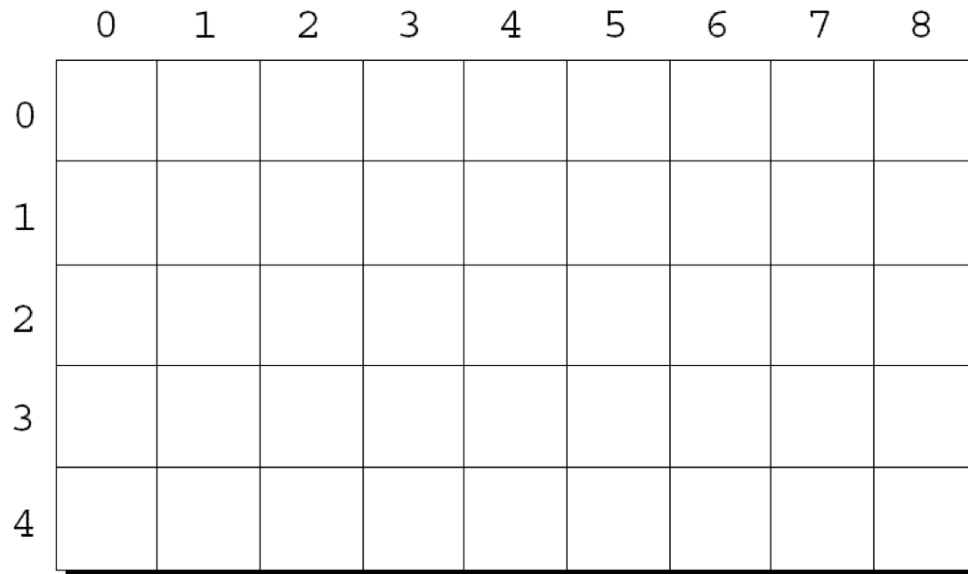

Array, 배열: 차원

- 1차원 `int a[10];`



a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]

- 2차원 `int a[5][9];`



3차원 이상도 표현가능
단, 시각화는 못함

문제

- `float element[100]`으로 선언된 배열의 시작 주소를 1000번지라고 할 때 10번째 요소의 주소는 몇 번지 인가?
- `int element[10] = { [4] = 1, 2, 3, [9] = 5};` 라고 선언 되었을 때 배열의 메모리 상의 모습을 그리시오
- `int element[3] = {X, Y};` 으로 선언되었다. 이 때 `x, y`는 임의의 상수이다. 두 요소를 바꾸는 문장을 작성 하시오

배열과 문자열

- 문자열 선언 방식 – 선언할 때만 문자열을 지정할 수 있음

```
char greeting = "hello world";
```

- 그 외의 모든 경우

```
char greeting[] = "hello world";
```

- 한 글자를 바꿀 때

```
greeting[0] = 'B';
```

- 모두 바꿀 때

```
strcpy(&greeting[0], "Incredible ");
```

Pointers

꼭 알아야 할 것

- 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 64개 2진수 표현 가능

• 1203의 2진수 표현

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

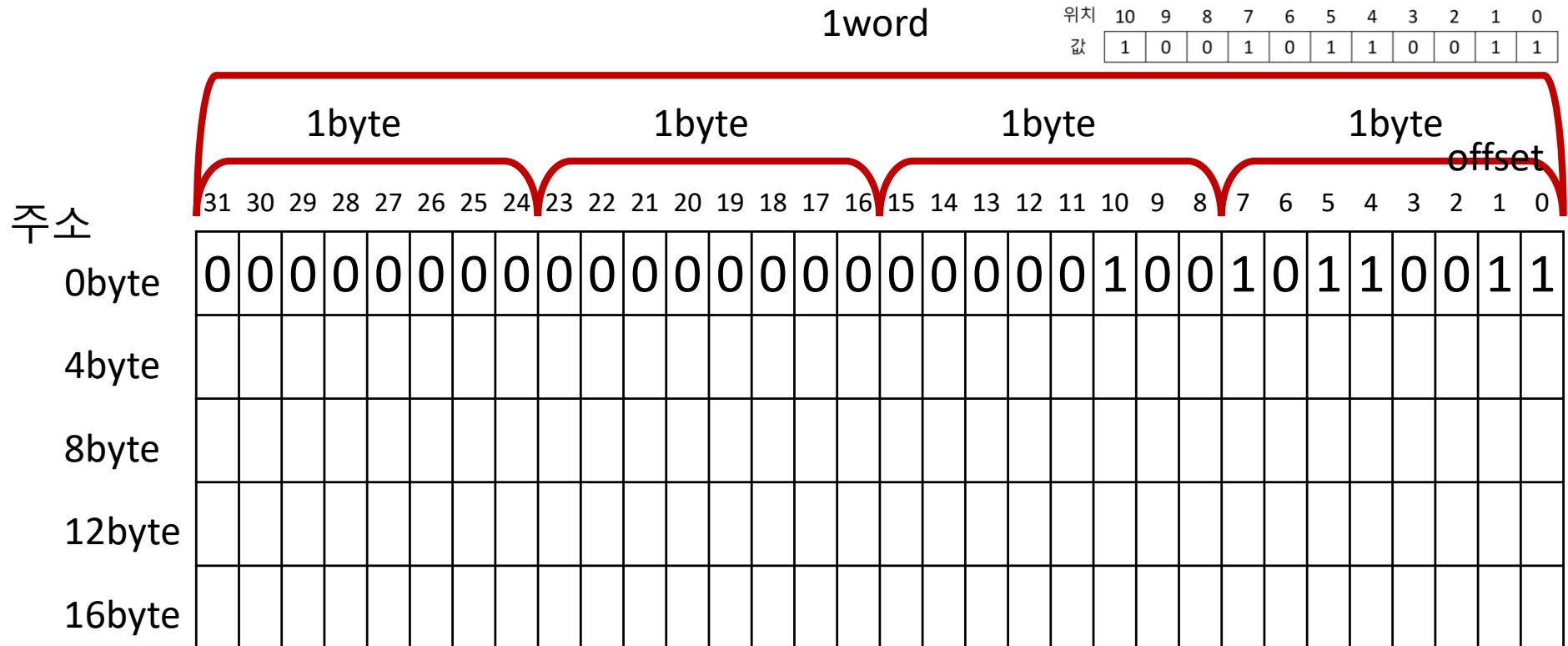
$$= 2^{10} + 2^7 + 2^5 + 2^4 + 2^1 + 2^0$$

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



주소

- 모든 객체 (변수, 함수 등)은 고유의 주소를 갖음

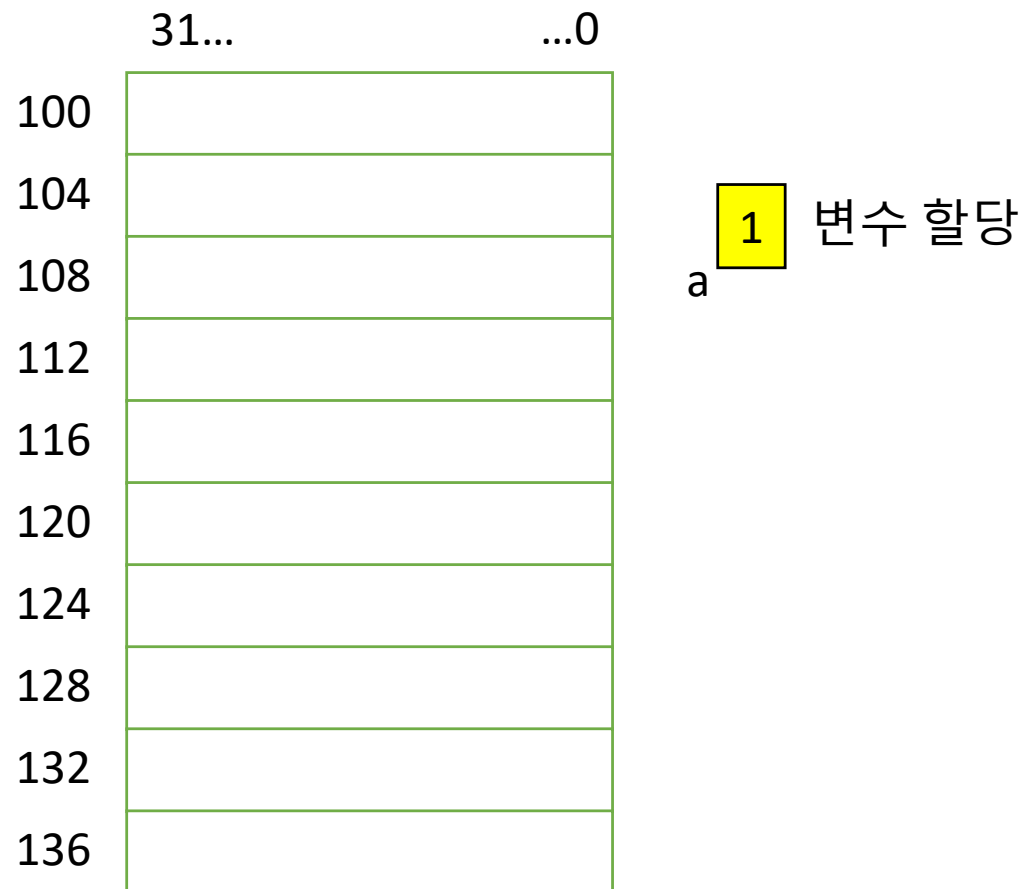
예시 `int a = 100;`
`int b = 200;`
`int c = 300;`

	31...	...0
100		
104		
108		
112		
116		
120		
124		
128		
132		
136		

주소

- 모든 객체 (변수, 함수 등)은 고유의 주소를 갖음

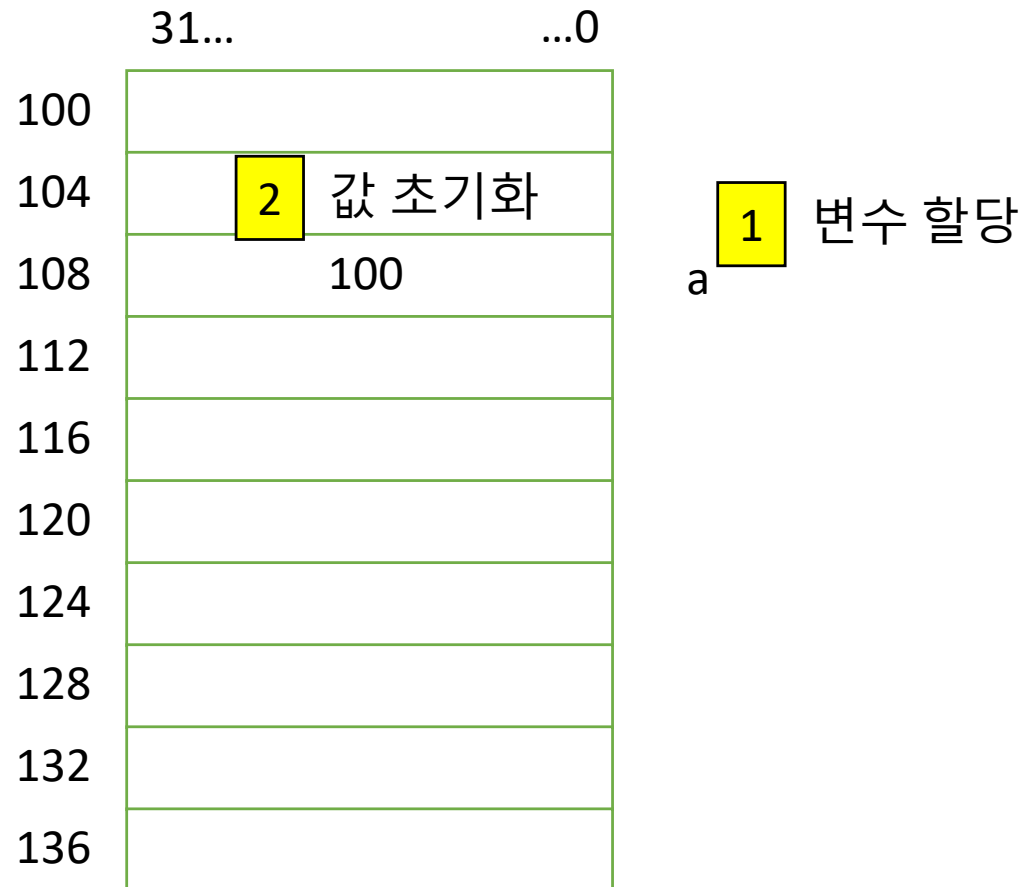
예시 `int a = 100;`
`int b = 200;`
`int c = 300;`



주소

- 모든 객체 (변수, 함수 등)은 고유의 주소를 갖음

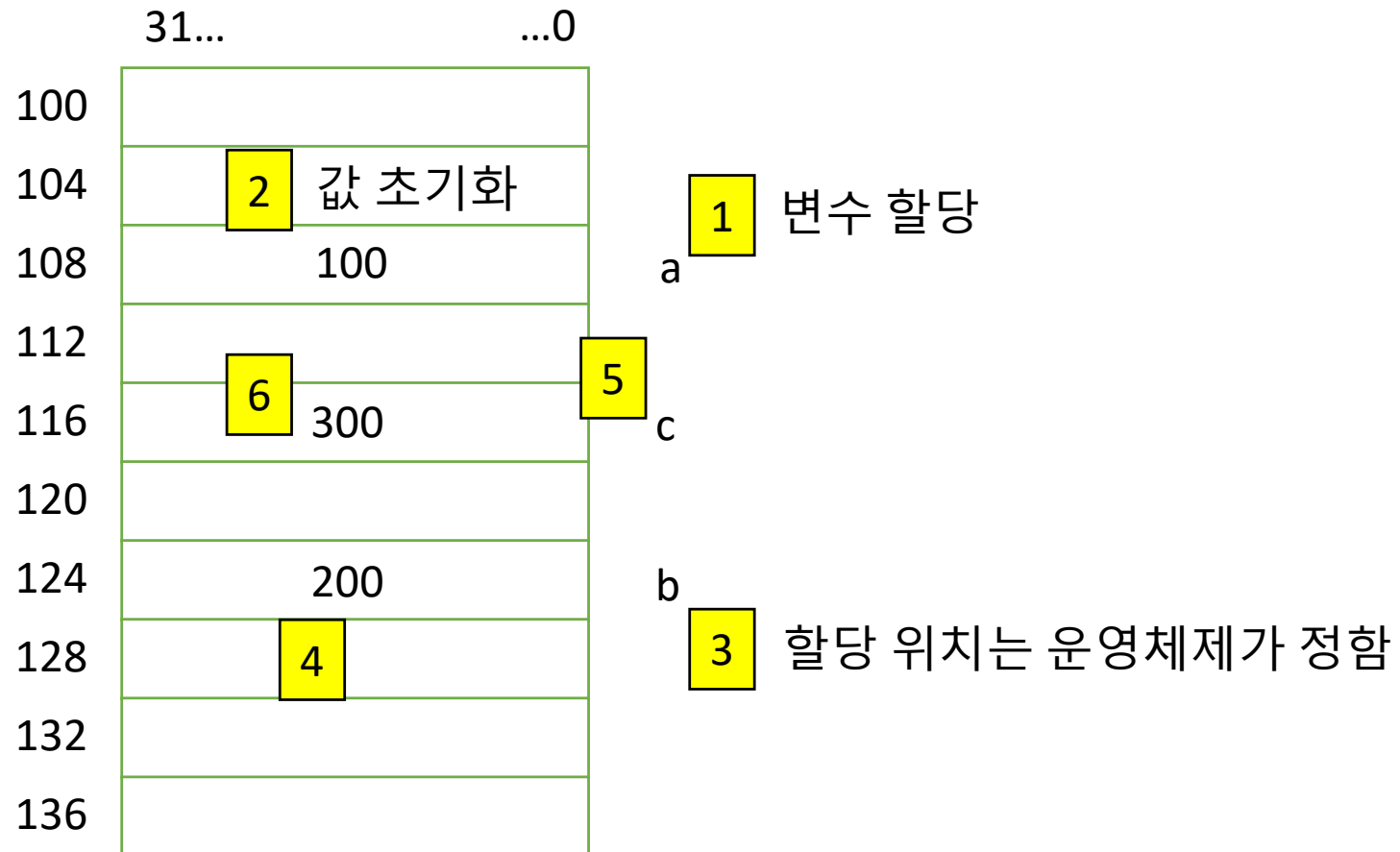
예시 `int a = 100;`
`int b = 200;`
`int c = 300;`



주소

- 모든 객체 (변수, 함수 등)은 고유의 주소를 가짐

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



문제

- 다음의 메모리 상태를 참고로, 변수 초기화 문장을 작성

	31...	...0	
100	392		c
104			
108	235		a
112			
116			d
120	ABC\0		
124			
128			b
132	8.0		
136			

문제

- 다음의 메모리 초기화 문장을 토대로 메모리 배치도 작성

	31...	...0
100		
104		
108		
112		
116		
120		
124		
128		
132		
136		

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

변수의 주소 확인

```
1 #include <stdio.h>
2 int main() {
3     int c[5] = { 3, 2, [3] = 8, [4] = 9};
4     for (int k = 0; k < 5; k++)
5         printf("value of c[%d] = %d; address is %p\n", \
6             k, c[k], &c[k]);
7
8     return 0;
9 }
10 }
```

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

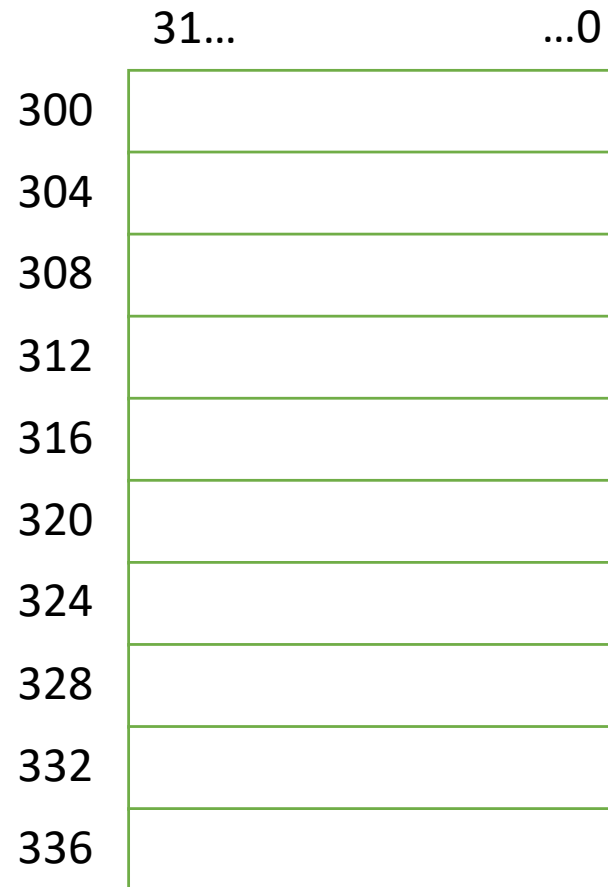
실행 결과

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

	31...	...0
0x7fff541c13d8	0	
	...	
0x7fff5602a3e0	3	
0x7fff5602a3e4	2	
0x7fff5602a3e8	0	
0x7fff5602a3ec	8	
0x7fff5602a3f0	9	

Scanf에서 &연산자의 활용

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



포인터의 2개의 연산자

&

참조 연산자(reference operator)

“~의 주소”로 이해

포인터 변수
저장하는 것은 주소

*

역참조 연산자(dereference operator)

“~가 가리키는 값”
으로 이해

*Dereference는 역참조 또는 간접참조라 부름

포인터 변수

1. 선언 – 역참조 연산자를 사용하여 주소를 저장하는 변수임을 표시
2. 할당 – 참조 연산자를 사용하여 주소를 포인터 변수에 저장
3. 읽기 – 역참조 연산자를 사용하여 저장된 주소가 가리키는 곳의 값을 가져옴

포인터 변수의 사용 예

- 다른 변수의 주소를 포인터 변수에 저장하는 방법

- Type I – 포인터 변수 선언시 주소 할당

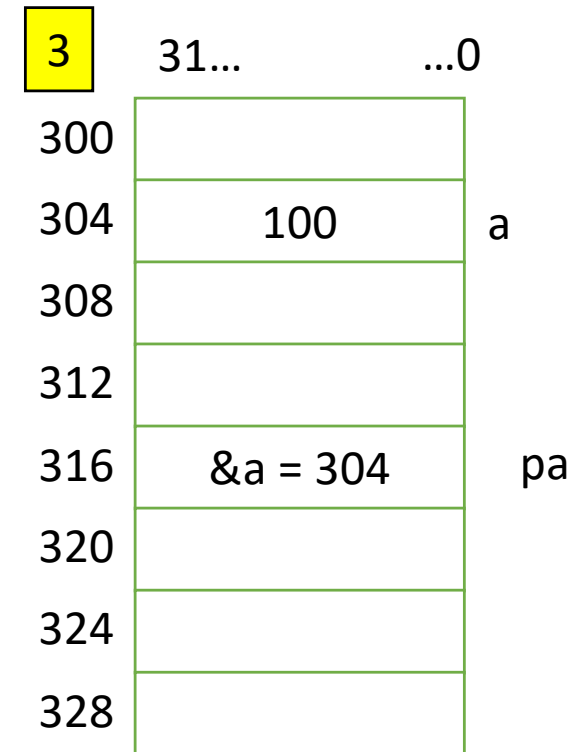
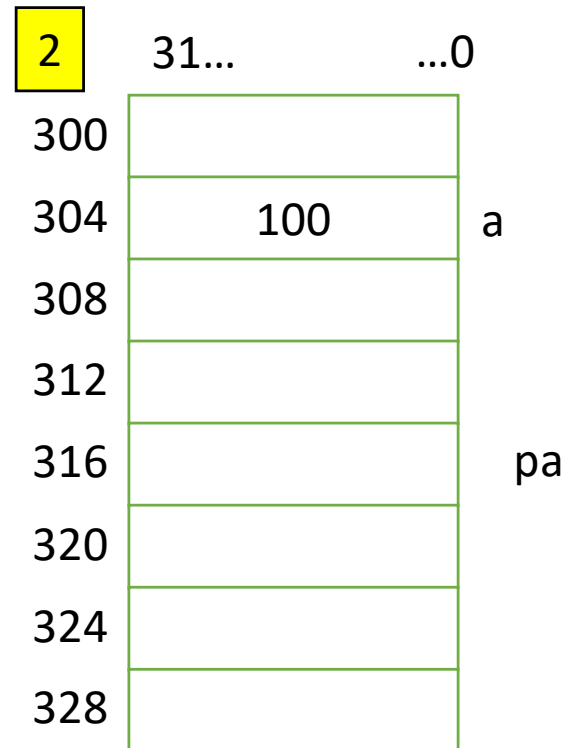
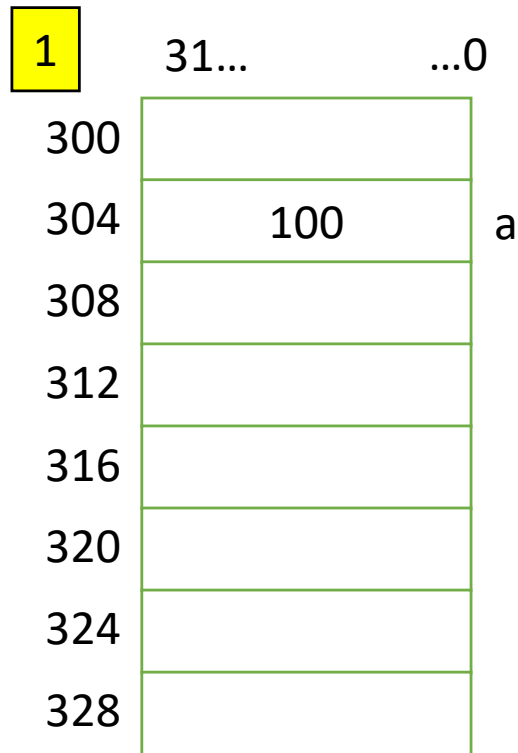
```
1  int a = 100;    // a 의 주소는 256이라 가정
2  int *pa = &a;   // 포인터 변수 선언 및 주소 할당
```

- 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);
```

	31...	...0	
100			pv
104	100		a
108			
112	300		c
116			
120	200		b
124			
128			
132			

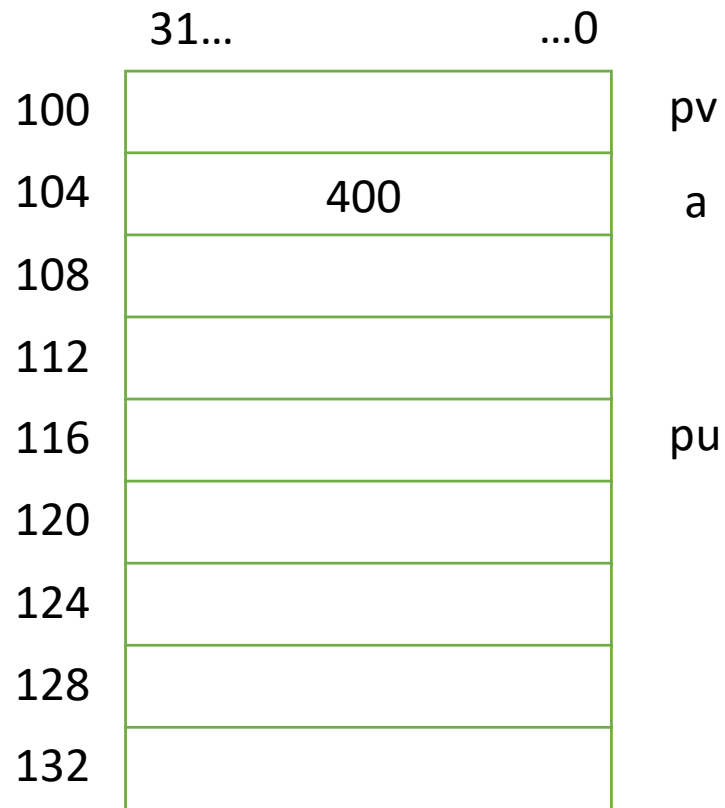
1 pv 값은 a의 주소 104

2 pv 값은 b의 주소 120

3 pv 값은 c의 주소 112

예시

```
1  int a;  
2  int *pv = &a, *pu = &a;  // pv와 pu는 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 a의 주소 104

포인터 연습 문제

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

14:16 -> 5
주소 1
주소 9

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

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

	31...	...0	
500	100		a
504			
508	300		b
512	1		c[0]
516	2		c[1]
520	500		pa
524			
528	504		pb
532	512		pc
536			
540	500		k
544			
548			

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

	31...	...0	
500	100		a
504			
508	300		b
512	1		c[0]
516	2		c[1]
520	500		pa
524			
528	504		pb
532	512		pc
536			
540	500		k
544			
548			

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("%f\n", *(double*)k);
```

	31...	...0	
500	100		a
504			
508	300		b
512	1		c[0]
516	2		c[1]
520	500		pa
524			
528	504		pb
532	512		pc
536			
540			k
544			
548			

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("%f\n", *(double*)k);
```

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

	31...	...0	
500	100		a
504			
508	300		b
512	1		c[0]
516	2		c[1]
520	500		pa
524			
528	504		pb
532	512		pc
536			
540			k
544			
548			

문제

- double형 변수 large를 int형 변수 small에 임시로 저장하고자 한다. 어떻게 해야 할까?

`small = (int)large;`

- void 형 포인터 변수 element가 가리키는 값을 float형으로 출력하고 싶다. 다음의 빈 칸을 채우시오

`printf("%f\n", (float)*element);`

- 포인터 변수의 크기는 항상 4 (주소체계의 크기) 바이트이다.
포인터 변수가 가리키는 값의 크기는 자료형의 크기에 의존한다

Arrays and Pointers

꼭 알아야 할 것

- 포인터와 배열의 관계
- 포인터 변수의 덧셈이 갖는 의미
- 포인터 변수의 배열 요소 지정 공식

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 = $512 + 4 * 2 = 520$

	31...	...0	
500			
504			
508			
512	1		c[0]
516	2		c[1]
520	3		c[2]
524	4		c[3]
528			
532			
536			
540	512		pc
544			
548			

배열의 주소

```
#include <stdio.h>
int main() {
    int c[] = {1, 2, 3, 4};
    printf("c\t%p\n", c);
    printf("&c\t%p\n", &c);
    printf("&c[0]\t%p\n", &c[0]);
    return 0;
}
```

```
$ gcc -o a.out addr.c
$ ./a.out
c      0x7fff574dbfd0
&c     0x7fff574dbfd0
&c[0] 0x7fff574dbfd0
```

31...	...	0	
0x7fff574dbfd0	1		c[0]
0x7fff574dbfd4	2		c[1]
0x7fff574dbfd8	3		c[2]
0x7fff574dbfdc	4		c[3]

배열 시작 주소 = 배열 인덱스 0의 주소

배열의 주소를 포인터로 확인하는 방법

```
#include <stdio.h>
int main() {
    int c[] = {1, 2, 3, 4};
    int *p = c;
    printf("c\t%p\n", c);
    printf("&c\t%p\n", &c);
    printf("&c[0]\t%p\n", &c[0]);
    printf("&*p\t%p\n", &*p);
    return 0;
}
```

```
$ gcc -o a.out addr.c
```

```
$ ./a.out
```

```
c      0x7fff574dbfd0
```

```
&c     0x7fff574dbfd0
```

```
&c[0]  0x7fff574dbfd0
```

```
&*p    0x7fff58497fd0
```

0x7fff574dbfd0

0x7fff574dbfd4

0x7fff574dbfd8

0x7fff574dbfdc

31...	...0	
0x7fff511e5fd0		p
	1	c[0]
	2	c[1]
	3	c[2]
	4	c[3]

포인터로 배열 참조 가능

포인터 변수의 덧셈

```
#include <stdio.h>
int main() {
    int c[] = {1, 2, 3, 4};
    int *p = c;
    for (int i = 0; i < 4; i++)
        printf("p+%d, addr: %p", i, p+i);
    return 0;
}
```

```
$ gcc -o a.out addr.c
$ ./a.out
p+0      0x7fff574dbfd0
p+1      0x7fff574dbfd4
p+2      0x7fff574dbfd8
p+3      0x7fff58497fdc
```

```
0x7fff574dbfd0
0x7fff574dbfd4
0x7fff574dbfd8
0x7fff574dbfdc
```

31...	...0	
0x7fff511e5fd0		p
1		c[0]
2		c[1]
3		c[2]
4		c[3]

포인터 변수를 증감하면 형의 크기만큼 증감

포인터 변수의 덧셈 II

```
#include <stdio.h>
int main() {
    int c[] = {1, 2, 3, 4};
    int *p = c;
    for (int i = 0; i < 4; i++)
        printf("p++, addr: %p", p++);
    return 0;
}
```

```
$ gcc -o a.out addr.c
```

```
$ ./a.out
```

```
p++      0x7fff574dbfd0
```

```
p++      0x7fff574dbfd4
```

```
p++      0x7fff574dbfd8
```

```
p++      0x7fff58497fdc
```

0x7fff574dbfd0

0x7fff574dbfd4

0x7fff574dbfd8

0x7fff574dbfdc

31...	...0	
0x7fff511e5fd0		p
1		c[0]
2		c[1]
3		c[2]
4		c[3]

++, -- 증감 연산자도 형의 크기만큼 증감

Putting it together

1. 배열 시작 주소 = 배열 인덱스 0의 주소

2. 포인터로 배열 참조 가능

+ 3. 포인터 변수의 증감 \rightarrow 형의 크기만큼 증감

포인터로 배열 인덱스 참조 가능

포인터로 배열 인덱스 지정하기 공식

시작 주소+(변수의 타입)*인덱스

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

	31...	...0	
500	512		pc
504			
508			
512	1		c[0]
516	2		c[1]
520	3		c[2]
524	4		c[3]

c[2]의 주소 = $512 + 4 * 2 = 520$

포인터로 배열 인덱스 지정하기

- 포인터 변수는 형의 크기 만큼 증감함

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

연습 문제

*p

c 주소의 값 1을 읽음

	31...	...0	
	0x7fff511e5fd0		p
0x7fff511e5fd0	1		c[0]
0x7fff511e5fd4	2		c[1]
0x7fff511e5fd8	3		c[2]
0x7fff511e5fdc	4		c[3]

포인터로 배열 인덱스 지정하기

- 포인터 변수는 형의 크기 만큼 증감함

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

연습 문제

$*p$ c[0]을 읽음

$*p+2$ c[2]를 읽음

	31...	...0	
	0x7fff511e5fd0		p
0x7fff511e5fd0	1		c[0]
0x7fff511e5fd4	2		c[1]
0x7fff511e5fd8	3		c[2]
0x7fff511e5fdc	4		c[3]

구조체 (structures)

꼭 알아야 할 것

- 구조체란
- 메모리에서의 표현
- 구조체 선언
- 구조체 초기화
- 구조체 값의 접근
- 구조체 활용
- 구조체로 형 선언
- **포인터 + 구조체 + malloc**

구조체란

- 복합 저장 공간

- 하나 이상의 서로 다른 형의 변수를 저장 가능
- 구조체 안에 구조체 포함 가능
- 관련있는 데이터의 모음

모이면 풍성해지는
구조체

- 예:

- 좌표계의 한 지점의 위치 값 (x, y, z)
- 성적표의 과목마다 받은 성적 (국, 영, 수, ...)
- 연락처에 들어가는 정보 (이름, 전화1, 전화2, 생일, 주소, ...)
- 도서관 서지 정보 (책 제목, 페이지수, 출판사, 출판일, ...)

구조체 선언: 기본 구조

```
struct {  
    변수형 변수이름; ← 멤버 변수  
    변수형 변수이름;  
    ...  
} 구조체명;
```

좌표계의 한 지점의 위치 값 (x, y, z)

지점의 좌표

```
struct {  
    int x;  
    int y;  
    int z;  
} PointA;
```

성적표의 과목마다 받은 성적
(국, 영, 수, ...) 성적표

```
struct {  
    float Kor;  
    float Eng;  
    float Math;  
} Score;
```

구조체 선언: 기본 구조

```
struct {  
    변수형 변수이름; ← 멤버 변수  
    변수형 변수이름;  
    ...  
} 구조체명;
```

연락처에 들어가는 정보 (이름, 전화1,
전화2, 생일, 주소,...)
전화번호

```
struct {  
    char Name[20];  
    char Phone[13];  
    char Birth[10];  
    char Addr[100];  
} PersonInfo;
```

도서관 서지 정보 (책 제목, 페이지수,
출판사, 출판일,...)
서지정보

```
struct {  
    int Title;  
    int Page;  
    int Pub;  
    int Year;  
} Book;
```

구조체 초기화: 선언시 초기화

```
struct {  
    변수형 변수이름;  
    변수형 변수이름;  
    ...  
}
```

배열처럼
초기화

```
    구조체명 = { 값1, 값2, ... };
```

지점의 좌표

```
struct {  
    int x;  
    int y;  
    int z;  
} PointA = { 30, 40, 50 }, PointB = {50, 30, 20 };
```

구조체 값의 접근

지점의 좌표

```
struct {  
    int x;  
    int y;  
    int z;  
} PointA = { 30, 40, 50 }, PointB = {50, 30, 20 };
```

새로운 연산자 . 점

구조체의 멤버 변수 접근

```
PointA . x; PointA . y; PointA . z;
```

구조체 초기화: 선언 후 초기화

지점의 좌표

```
struct {  
    int x;  
    int y;  
    int z;  
} PointA;  
  
PointA.x = 30;  
PointA.y = 40;  
PointA.z = 50;
```


메모리에서의 표현

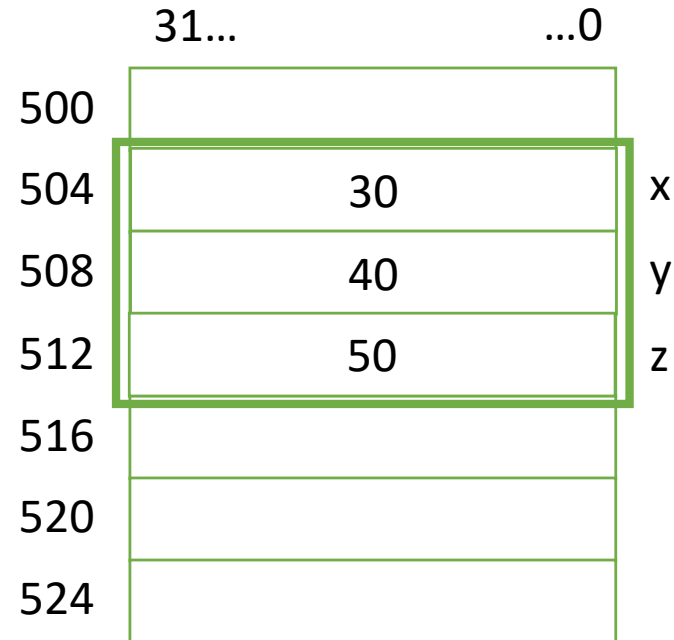
지점의 좌표

```
struct {  
    int x;  
    int y;  
    int z;  
} PointA;
```

```
PointA.x = 30;
```

```
PointA.y = 40;
```

```
PointA.z = 50;
```



구조체 활용: 함수의 인자, 매개 변수

```
struct point {  
    int x;  
    int y;  
    int z;  
};
```

구조체의 선언

리턴 타입: struct point

```
struct point savePoint(int x, int y, int z)  
{
```

```
    struct point p;  
    p.x = x;  
    p.y = y;  
    p.z = z;  
    return p;  
}
```

구조체 변수 리턴

```
int main()  
{
```

```
    struct point pointA;  
    pointA = savePoint(30, 40, 50);
```

리턴 받은 정보를
PointA에 저장

구조체 형 선언

- 태그로 형 선언

지점의 좌표

```
struct Point{  
    int x;  
    int y;  
    int z;  
} ;
```

- typedef로 형 선언

구조체에 반복 사용할 수 있는
이름을 부여함

```
struct Point A = {30, 40, 50};  
struct Point B = {40, 30, 20};
```

구조체 태그가 있기 때문에 구조체 재선언 없이
같은 타입의 구조체를 재활용하여 선언 할 수 있음
태그가 없으면 구조체 선언부를 매번 다시 써야 함

구조체 형 선언

- 태그로 선언하여 쓰는 방법의 단점
 - “struct 구조체 태그명”을 반복하여 써야 함
- typedef으로 선언하는 방법
 - 선언한 구조체가 새로운 형이 됨

지점의 좌표

```
typedef struct {  
    int x;  
    int y;  
    int z;  
} Point;
```

typedef 새로운 형을 만들 때 쓰는 키워드

새로운 형의 이름

```
Point A = {30, 40, 50};  
Point B = {40, 30, 20};
```

다른 형들처럼 변수 명 앞에 씀

구조체의 할당

지점의 좌표

```
typedef struct {  
    int x;  
    int y;  
    int z;  
} Point;
```

```
Point A = {30, 40, 50};  
Point B;  
B = A;
```

구조체는 = 연산자로 복사 됨
A.x에 30, A.y에 40, A.z에 50을 저장
B = A; 문장으로 통해
B.x에 30, B.y에 40, B.z에 50을 저장

단, 모든 구조체가 동일하게 동작하는 것은 아님
상세 자료 참고

구조체 포인터 변수

새로운 연산자 ->

```
typedef struct person {
    int age;
    float weight;
};

int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;
    scanf("%d", &(*personPtr).age);
    printf("%d", (*personPtr).age); // 다른 표현 personPtr->age;
}
```

Ex: malloc() and pointer to struct

```
#include <stdio.h>
#include <stdlib.h>

struct person {
    int age;
    float weight;
    char name[30];
};

int main() {
    struct person *ptr;
    int i, num;
    printf("Enter number of persons: ");
    scanf("%d", &num);
    ptr = (struct person*) malloc(num * sizeof(struct person));

    for(i = 0; i < num; ++i) {
        printf("Enter name, age and weight of the person respectively:\n");
        scanf("%s%d%f", &(ptr+i)->name, &(ptr+i)->age, &(ptr+i)->weight);
    }

    for(i = 0; i < num; ++i) // Displaying Info
        printf("%s\t%d\t%.2f\n", (ptr+i)->name, (ptr+i)->age, (ptr+i)->weight);
    return 0;
}
```

Appendix: Function

Function, 함수

- 수행하고자 하는 일련의 동작들에 붙여진 이름
- 프로그램을 이해하고 수정하는데 도움이 됨
 - Definition, 정의
 - Calling, 호출
 - Arguments, 인자
 - return, 리턴
 - recursion, 재귀

Function, 함수: Definition, 정의

- 호출하려는 함수보다 먼저 함수의 정의가 작성되어야함
 - 함수 선언:
 - 함수의 내용은 없이 앞으로 사용할 함수의 이름과 인자값을 프로그램에 등록함.
 - 작성 스타일에 따라 항상 필요하진 않음.
 - 함수 정의:
 - 함수의 실제 내용이 기록됨

Function, 함수: Declaration, 선언

```
#include <stdio.h>
```

```
int sum(int a, int b) ; 함수 선언
```

```
int main() {
```

```
    ...
```

```
}
```

```
int sum(int a, int b) 함수 정의
```

```
{
```

함수 내용

```
}
```

Function, 함수: Definition, 정의

리턴 데이터타입	함수 이름	함수 내에서 사용될 인자들
<i>return-type</i>	<i>function-name</i>	(<i>parameters</i>)
{		
<i>declarations</i>	함수 내에서 사용될 변수 선언	
<i>statements</i>	함수 내에서 실행할 문장들	
}		

int sum(int a, int b)

{

 int result=0;

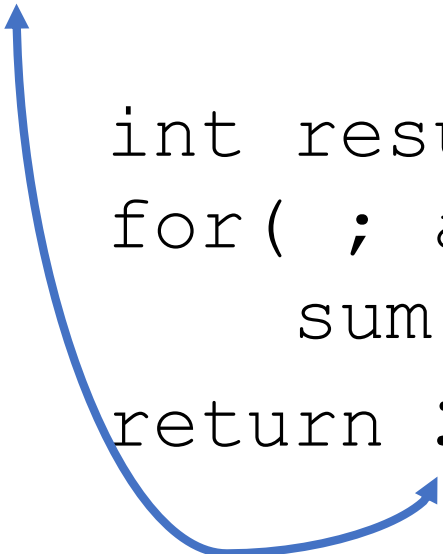
 for(; a <= b; a++)

 sum += a;

 return **result**;

}

타입 일치



Function, 함수: Calling, 호출

- 코드 내에 함수 이름을 작성하여 호출

```
int main() {
```

```
...
```

```
answer = sum(val1, val2);
```

```
...
```

```
}
```

기존 코드의 흐름은 위에서 아래로 실행

함수 호출 시 함수 정의 위치로 실행의 흐름이 이동함

```
int sum(int a, int b)
```

```
{
```

```
...
```

```
return result;
```

```
}
```

Function, 함수: Arguments, 인자

- 함수 밖에서 정의된 값으로 함수 내에서 활용할 변수 이름

```
int main() {
```

```
...
```

```
answer = sum(val1, val2);
```

```
...
```

```
}
```

메인 코드에서 선언된 val1과 val2를
sum이라는 함수에 활용할 수 있도록 인자로 넣었음

함수에서 활용할 변수 이름은 메인코드의 변수 이름과 달라도 됨

```
int sum(int a, int b)
```

```
{
```

```
...
```

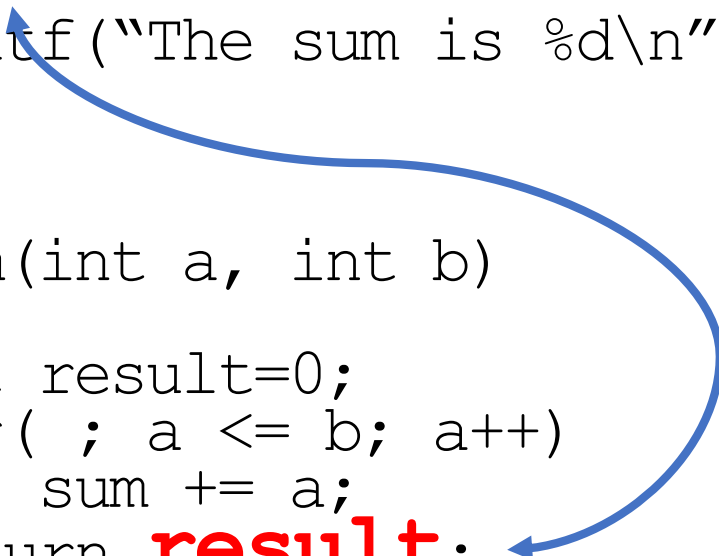
```
return a + b;
```

```
}
```

Function, 함수: 종합 예제

```
#include <stdio.h>
int sum(int a, int b);
int main()
{
    int val1, val2, answer;
    printf("Input two numbers in increasing order: ");
    scanf("%d %d", &val1, &val2);
    answer = sum(val1, val2);호출
    printf("The sum is %d\n", answer);
}

int sum(int a, int b)
{
    int result=0;
    for( ; a <= b; a++)
        sum += a;
    return result;
}
```

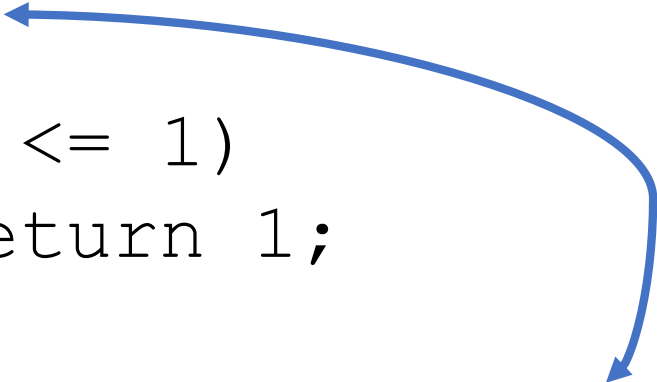


결과 값 반환

Function, 함수: Recursion, 재귀

- 함수가 호출되면 해당 코드를 실행하고 결과를 리턴함
- 함수가 자기 스스로를 다시 호출하는 것이 가능함
 - 팩토리얼을 기억해보기 바람

```
int fact(int n)
{
    if (n <= 1)
        return 1;
    else
        return n * fact(n - 1);
}
```



다시 호출