2008 Spring

Computer Engineering Programming 1

Lesson 12
- 제13장 구조체

Lecturer: JUNBEOM YOO
jbyoo@konkuk.ac.kr

이번 장에서 학습할 내용

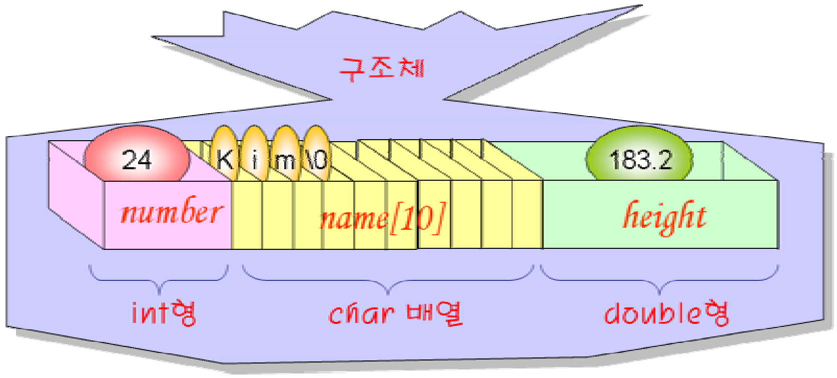
- 구조체란 무엇인가?
- 구조체의 선언, 초기화, 사용
- 구조체의 배열
- 구조체와 포인터
- 구조체와 함수
- 공용체
- 열거형
- typedef

구조체는 복잡한 내용을 일목요연하여 정리하는데 유용하게 사용됩니다.

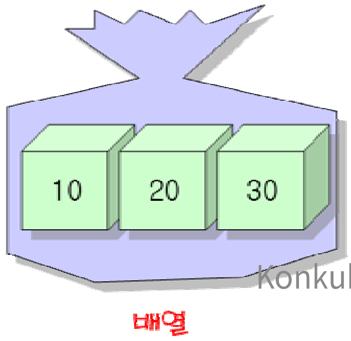
구조체란?

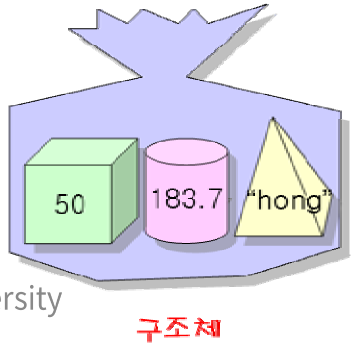
자료형
- 기본자료형: char, int, float, double 등
- 파생자료형: 배열, 열거형, 구조체, 공용체

- 구조체: 서로 다른 자료형을 하나로 묶는 구조



- 구조체 vs 배열





구조체의 선언

- 구조체 선언 형식

```
struct 구조체_태그_이름 {
      자료형  멤버_이름;
      자료형  멤버_이름;
      ...
};
```

- (예)학생에 대한 데이터

태그(tag)

```
struct student {
        int number;        // 학번
        char name[10];  // 이름
        double height;  // 키
};
```

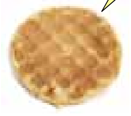
멤버(member)

- 구조체 선언은 변수 선언은 아님

구조체를 정의하는 것은 와플이나 붕어빵을 만드는 틀을 정의하는 것과 같다.

와플이나 붕어빵을 실제로 만들릭 위해서는 구조체 변수를 선언하여야 한다.

구조체


구조체 변수

구조체 선언의 예

```
// x값과 y값으로 이루어지는 화면의 좌표
struct point {
        int x;              // x 좌표
        int y;              // y 좌표
};
```

```
// 복소수
struct complex {
        double real;        // 실수부
        double imag;        // 허수부
};
```

```
// 날짜
struct date {
        int month;
        int day;
        int year;
};
```

```
// 사각형
struct rect {
        int x;
        int y;
        int width;
        int height;
};
```

```
// 직원
struct employee {
        char name[20];      // 이름
        int age;            // 나이
        int gender;         // 성별
        int salary;         // 월급
};
```

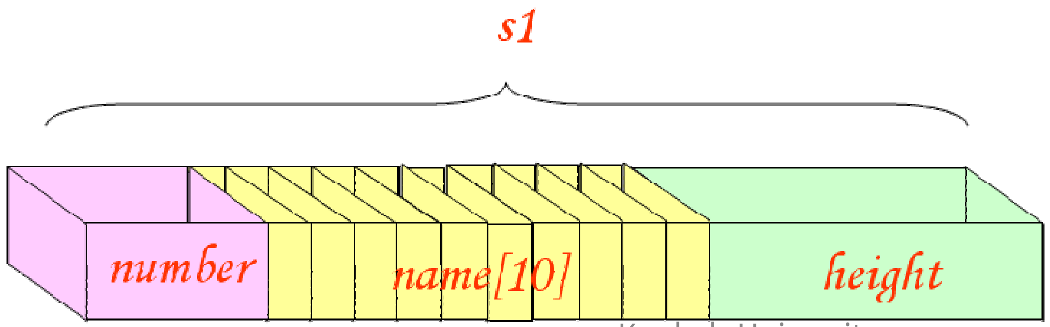
구조체 변수 선언

- 구조체 정의와 구조체 변수 선언은 다르다.

```
struct student {
        int number;
        char name[20];
        double height;
};

int main(void){
        struct student s1;
        ...
}
```

구조체 정의

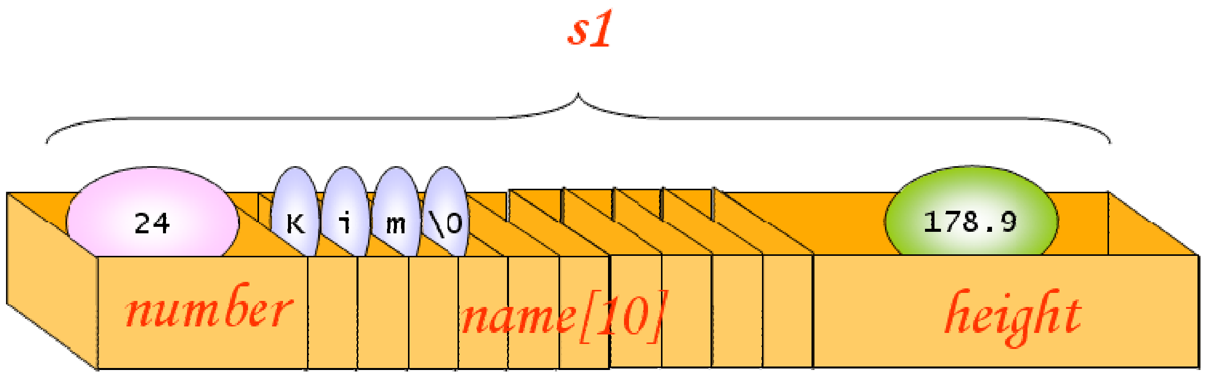
구조체 변수 선언



구조체의 초기화

- 중괄호를 이용하여 초기값을 나열한다.

```
struct student {
            int number;
            char name[10];
            double height;
};
struct student s1 = { 24, "Kim", 178.9 };
```



구조체 멤버 참조

- 구조체 멤버를 참조하려면 다음과 같이 . 연산자를 사용한다.

```
s1.number = 26;                    // 정수 멤버
strcpy(s1.name, "Kim");            // 문자열 멤버
s1.height = 183.2;                 // 실수 멤버
```

구조체를 멤버로 가지는 구조체

```
struct date {                      // 구조체 선언
      int year;
      int month;
      int day;
};
```

```
struct student {                   // 구조체 선언
            int number;
            char name[10];
            struct date dob;    // 구조체 안에 구조체 포함
            double height;
};
struct       student    s1;        // 구조체 변수 선언
```

```
s1.dob.year = 1983;                // 멤버 참조
s1.dob.month = 03;
s1.dob.day = 29;
```

예제 #1

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

struct student {
        int number;
        char name[10];
        double height;
};

int main(void)
{
        struct student s;

        s.number = 20070001;
        strcpy(s.name,"홍길동");
        s.height = 180.2;

        printf("학번: %d\n", s.number);
        printf("이름: %s\n", s.name);
        printf("신장: %f\n", s.height);

        return 0;
}
```

구조체 선언

구조체 멤버 참조

```
학번: 20070001
이름: 홍길동
신장: 180.200000
```

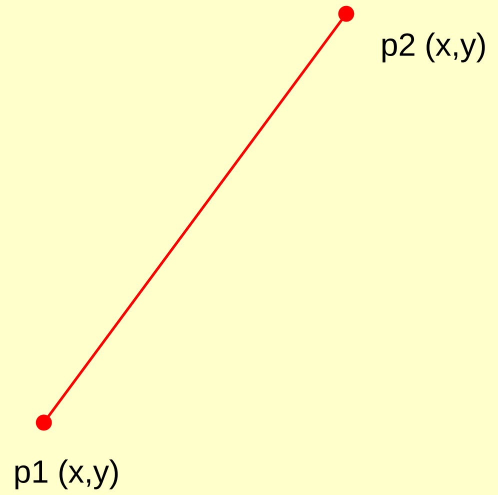
예제 #2

```c
struct student {
	int number;
	char name[10];
	double height;
};

int main(void)
{
	struct student s;

	printf("학번을 입력하시오: ");
	scanf("%d", &s.number);

	printf("이름을 입력하시오: ");
	scanf("%s", s.name);

	printf("신장을 입력하시오(실수): ");
	scanf("%lf", &s.height);

	printf("학번: %d\n", s.number);
	printf("이름: %s\n", s.name);
	printf("신장: %f\n", s.height);
	return 0;
}
```

```
학번을 입력하시오: 20070001
이름을 입력하시오: 홍길동
신장을 입력하시오(실수): 180.2
학번: 20070001
이름: 홍길동
신장: 180.200000
```

예제 #3

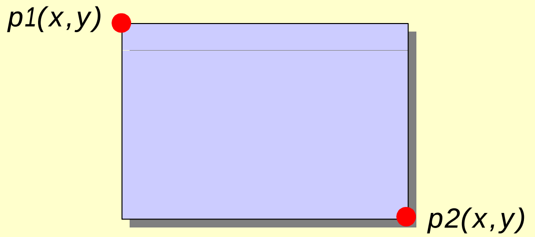
```
#include <math.h>

struct point {
	int x;
	int y;
};

int main(void)
{
	struct point p1, p2;
	int xdiff, ydiff;
	double dist;

	printf("점의 좌표를 입력하시오(x  y): ");
	scanf("%d %d", &p1.x, &p1.y);

	printf("점의 좌표를 입력하시오(x  y): ");
	scanf("%d %d", &p2.x, &p2.y);

	xdiff = p1.x - p2.x;
	ydiff = p1.y - p2.y;

	dist = sqrt(xdiff * xdiff + ydiff * ydiff);

	printf("두 점사이의 거리는 %f입니다.\n", dist);

	return 0;
}
```



```
점의 좌표를 입력하시오(x y): 10 10
점의 좌표를 입력하시오(x y): 20 20
두 점사이의 거리는 14.142136 입니다.
```

예제 #4

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

struct rect {
	struct point p1;
	struct point p2;
};

int main(void)
{
	struct rect r;
	int w, h, area, peri;

	printf("왼쪽 상단의 좌표를 입력하시오: ");
	scanf("%d %d", &r.p1.x, &r.p1.y);

	printf("오른쪽 상단의 좌표를 입력하시오: ");
	scanf("%d %d", &r.p2.x, &r.p2.y);

	w = r.p2.x - r.p1.x;
	h = r.p2.x - r.p1.x;

	area = w * h;
	peri = 2 * w + 2 * h;
	printf("면적은 %d이고 둘레는 %d입니다.\n", area, peri);

	return 0;
}
```



```
왼쪽 상단의 좌표를 입력하시오: 1 1
오른쪽 상단의 좌표를 입력하시오: 6 6
면적은 25이고 둘레는 20입니다.
```

구조체 변수의 대입과 비교

- 같은 구조체 변수끼리 대입은 가능하지만 비교는 불가능하다.

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

int main(void)
{
	struct point p1 = {10, 20};
	struct point p2 = {30, 40};

	p2 = p1;                               // 대입 가능

	if( p1 == p2 )                         // 비교 -> 컴파일 오류!!
		printf("p1와 p2이 같습니다.")

	if( (p1.x == p2.x) && (p1.y == p2.y) ) // 올바른 비교
		printf("p1와 p2이 같습니다.")
}
```

구조체 배열

- 구조체 배열의 선언

```
struct student {
      int number;
      char name[20];
      double height;
};

int main(void)
{
      struct student list[100];       // 구조체의 배열 선언


      list[2].number = 27;
      strcpy(list[2].name, "홍길동");
      list[2].height = 178.0;
}
```

- 구조체 배열의 초기화

```
struct student list[3] = {
      { 1, "Park", 172.8 },
      { 2, "Kim", 179.2 },
      { 3, "Lee", 180.3 }
};
```

구조체 배열 예제

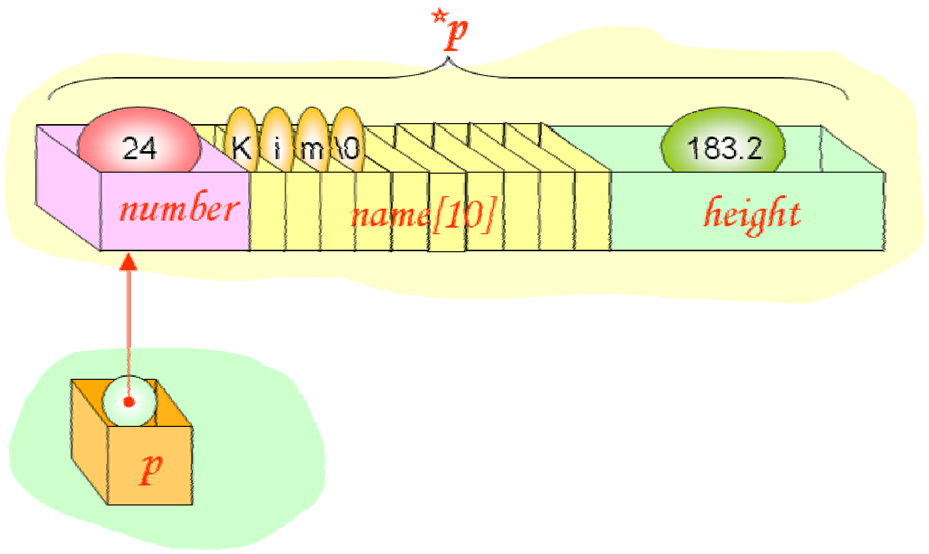
```c
#define SIZE 3

struct student {
	int number;
	char name[20];
	double height;
};
int main(void)
{
	struct student list[SIZE];
	int i;

	for(i = 0; i < SIZE; i++)
	{
		printf("학번을 입력하시오: ");
		scanf("%d", &list[i].number);
		printf("이름을 입력하시오: ");
		scanf("%s", list[i].name);
		printf("신장을 입력하시오(실수): ");
		scanf("%lf", &list[i].height);
	}

	for(i = 0; i< SIZE; i++)
		printf("학번: %d, 이름: %s,  신장: %f\n", list[i].number, list[i].name, list[i].height);

	return 0;
}
```

```
학번을 입력하시오: 20070001
이름을 입력하시오: 홍길동
신장을 입력하시오(실수): 180.2
학번을 입력하시오: 20070002
이름을 입력하시오: 김유신
신장을 입력하시오(실수): 178.3
학번을 입력하시오: 20070003
이름을 입력하시오: 이성계
신장을 입력하시오(실수): 176.3
학번: 20070001, 이름: 홍길동, 신장: 180.200000
학번: 20070002, 이름: 김유신, 신장: 178.300000
학번: 20070003, 이름: 이성계, 신장: 176.300000
```

구조체와 포인터

- 구조체를 가리키는 포인터

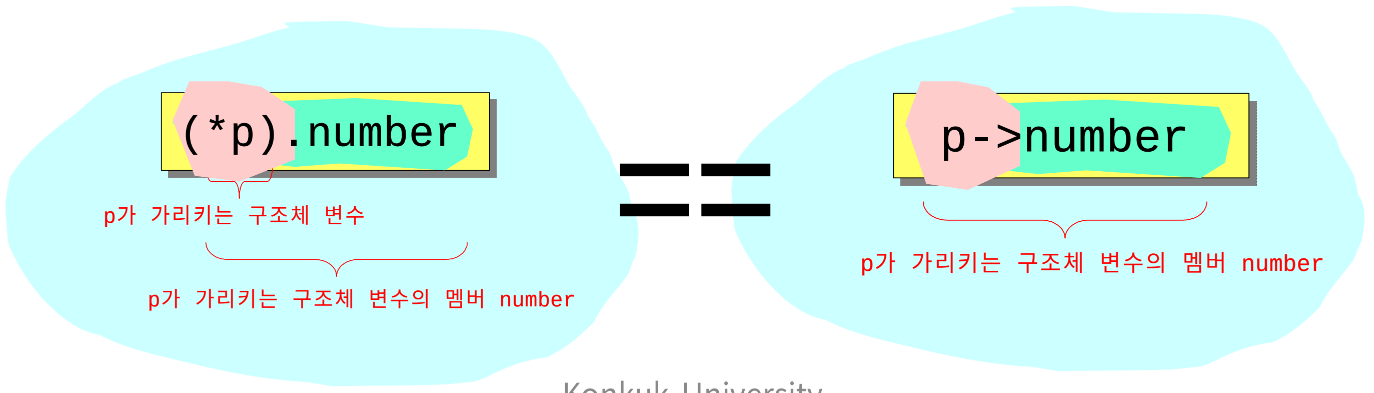
```
struct student *p;

struct student s = { 20070001, "홍길동", 180.2 };
struct student *p;

p = &s;

printf("학번=%d 이름=%s 키=%f \n", s.number, s.name, s.height);
printf("학번=%d 이름=%s 키=%f \n", (*p).number,(*p).name,(*p).height);
```



-> 연산자

- -> 연산자는 구조체 포인터로 구조체 멤버를 참조할 때 사용

```
struct student *p;

struct student s = { 20070001, "홍길동", 180.2 };
struct student *p;

p = &s;

printf("학번=%d 이름=%s 키=%f \n", s.number, s.name, s.height);
printf("학번=%d 이름=%s 키=%f \n", (*p).number,(*p).name,(*p).height);
printf("학번=%d 이름=%s 키=%f \n", p->number, p->name, p->height);
```



(*p).number
p가 가리키는 구조체 변수
p가 가리키는 구조체 변수의 멤버 number
==
p->number
p가 가리키는 구조체 변수의 멤버 number

예제

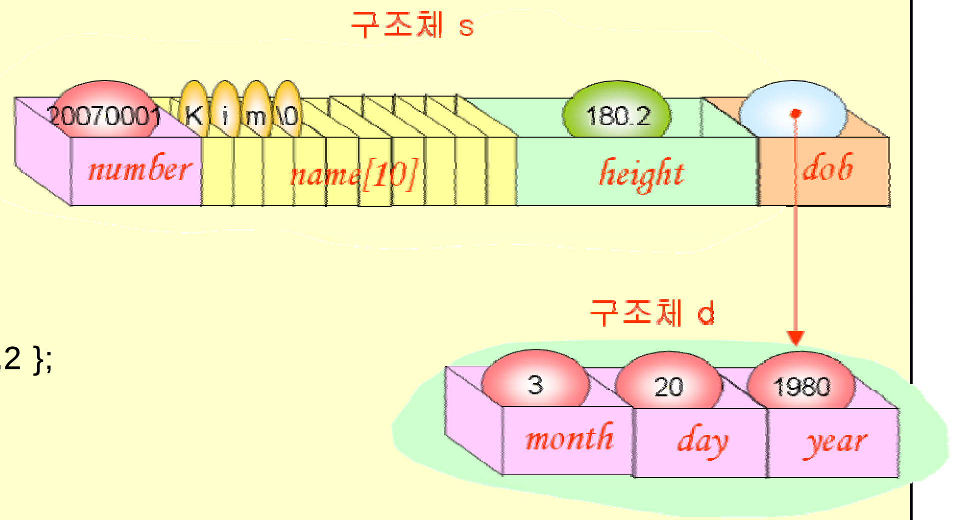
```c
// 포인터를 통한 구조체 참조
#include <stdio.h>

struct student {
	int number;
	char name[20];
	double height;
};

int main(void)
{
	struct student s = { 20070001, "홍길동", 180.2 };
	struct student *p;

	p = &s;

	printf("학번=%d 이름=%s 키=%f \n", s.number, s.name, s.height);
	printf("학번=%d 이름=%s 키=%f \n", (*p).number,(*p).name,(*p).height);
	printf("학번=%d 이름=%s 키=%f \n", p->number, p->name, p->height);

	return 0;
}
```

```
학번=20070001 이름=홍길동 키=180.200000
학번=20070001 이름=홍길동 키=180.200000
학번=20070001 이름=홍길동 키=180.200000
```

포인터를 멤버로 가지는 구조체

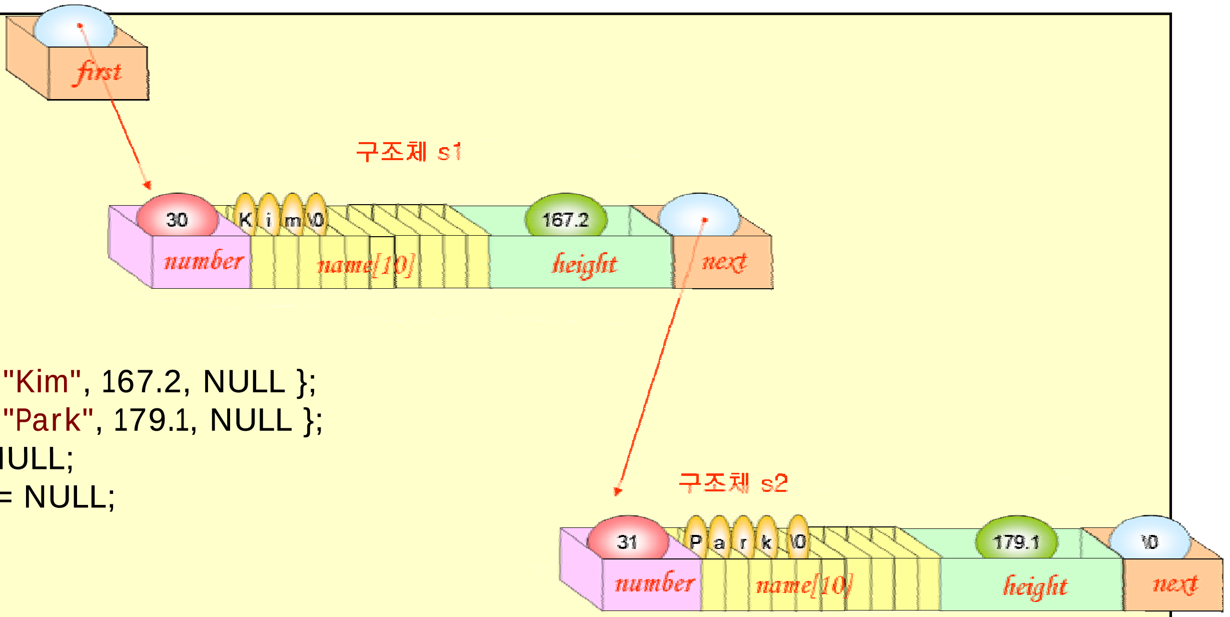
```c
struct date {
        int month;
        int day;
        int year;
};
struct student {
        int number;
        char name[20];
        double height;
        struct date *dob;
};
int main(void)
{
        struct date d = { 3, 20, 1980 };
        struct student s = { 20070001, "Kim", 180.2 };

        s.dob = &d;

        printf("학번: %d\n", s.number);
        printf("이름: %s\n", s.name);
        printf("신장: %f\n", s.height);
        printf("생년월일: %d년 %d월 %d일\n", s.dob->year, s.dob->month, s.dob->day);
        return 0;
}
```

```
학번: 20070001
이름: Kim
신장: 180.200000
생년월일: 1980년 3월 20일
```



자체 참조 구조체

```c
struct student {
        int number;
        char name[10];
        double height;
        struct student *next;
};

int main(void)
{
        struct student s1 = { 30, "Kim", 167.2, NULL };
        struct student s2 = { 31, "Park", 179.1, NULL };
        struct student *first = NULL;
        struct student *current = NULL;

        first = &s1;
        s1.next = &s2;
        s2.next = NULL;

        current = first;
        while( current != NULL )
        {
                printf("학생의 번호=%d 이름=%s, 키=%f\n", current->number,
                                current->name, current->height);
                current = current->next;
        }
}
```



```
학생의 번호=30 이름=Kim, 키=167.200000
학생의 번호=31 이름=Park, 키=179.100000
```

구조체와 함수

- *구조체*를 함수의 인수로 전달하는 경우
  – 구조체의 복사본이 함수로 전달되게 된다.
  – 만약 구조체의 크기가 크면 그만큼 시간과 메모리가 소요된다.

```
int equal(struct student s1, struct student s2)
{
        if( strcmp(s1.name, s2.name) == 0 )
                return 1;
        else
                return 0;
}
```

- *구조체의 포인터*를 함수의 인수로 전달하는 경우
  – 시간과 공간을 절약할 수 있다.

```
int equal(struct student const *p1, struct student const *p2)
{
        if( strcmp(p1->name, p2->name) == 0 )
                return 1;
        else
                return 0;
}
```

구조체를 반환하는 경우

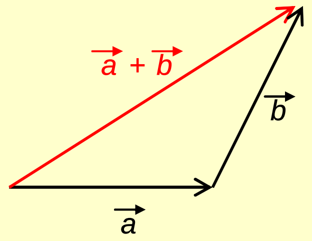
- 복사본이 반환된다.

```
struct student make_student(void)
{
      struct student s;

      printf("나이:“);
      scanf("%d", &s.age);
      printf("이름:“);
      scanf("%s", s.name);
      printf("키:“);
      scanf("%f", &s.height);

      return s;
}
```

구조체 s의 복사본이 반환된다.

예제

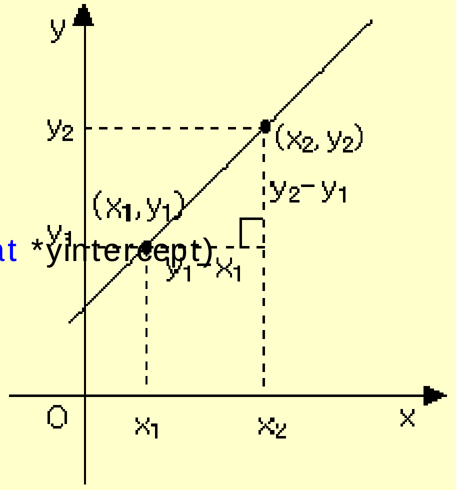
```c
#include <stdio.h>

struct vector {
	float x;
	float y;
};
struct vector get_vector_sum(struct vector a, struct vector b);

int main(void)
{
	struct vector a = { 2.0, 3.0 };
	struct vector b = { 5.0, 6.0 };
	struct vector sum;

	sum = get_vector_sum(a, b);
	printf("벡터의 합은 (%f, %f)입니다.\n", sum.x, sum.y);

	return 0;
}

struct vector get_vector_sum(struct vector a, struct vector b)
{
	struct vector result;

	result.x = a.x + b.x;
	result.y = a.y + b.y;

	return result;
}
```



```
벡터의 합은 (7.000000, 9.000000)입니다.
```

예제

```c
#include <stdio.h>
struct point {
	int x;
	int y;
};
// 기울기와 y절편을 계산
int get_line_parameter(struct point p1, struct point p2, float *slope, float *yintercept)
{
	if( p1.x == p2.x )
		return (-1);
	else
	{
	  *slope = (float)(p2.y - p1.y)/(float)(p2.x - p1.x);
	  *yintercept = p1.y - (*slope) * p1.x;
	  return (0);
	}
}
int main(void)
{
	struct point pt1 = {3, 3}, pt2 = {6, 6};
	float s,y;

	if( get_line_parameter(pt1, pt2, &s, &y) == -1 )
		printf("오류: 두점의 x좌표값이 동일합니다.\n");
	else
		printf("기울기는 %f, y절편은 %f\n", s, y);
	return 0;
}
```

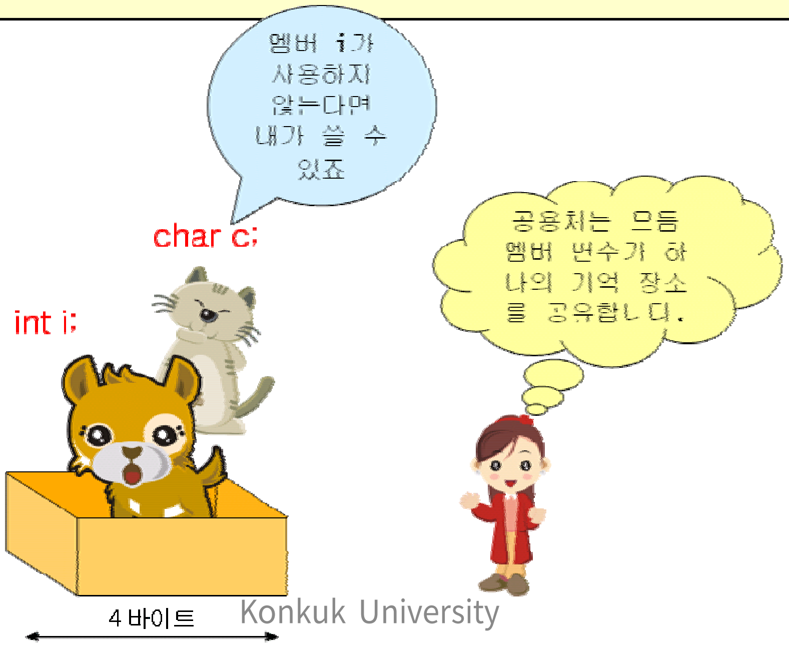


```
기울기는 1.000000, y절편은 0.000000
```

공용체

- 공용체(union)
  - 같은 메모리 영역을 여러 개의 변수가 공유
  - 공용체를 선언하고 사용하는 방법은 구조체와 아주 비슷

```
union example {
        char c;                    // 같은 기억 공간 공유
        int i;                     // 같은 기억 공간 공유
};
```


멤버 i가 사용하지 않는다면 내가 쓸 수 있죠
char c;
int i;
공용체는 모든 멤버 변수가 하나의 기억 장소를 공유합니다.
4 바이트

예제

```
#include <stdio.h>

union example {
      int i;
      char c;
};

int main(void)
{
      union example v;

      v.c = 'A';
      printf("v.c:%c   v.i:%i\n", v.c, v.i );

      v.i = 10000;
      printf("v.c:%c   v.i:%i\n", v.c, v.i);

}
```

공용체 선언

공용체 변수 선언.

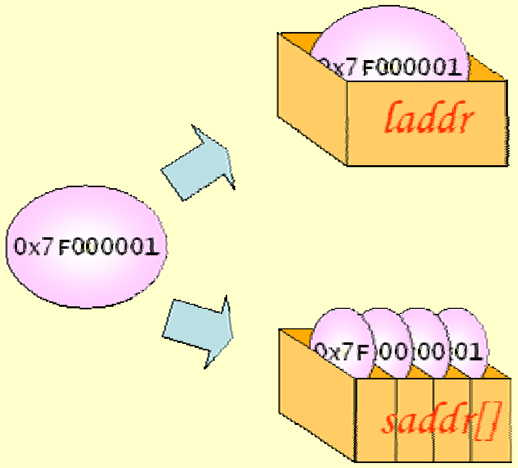
char 형으로 참조.

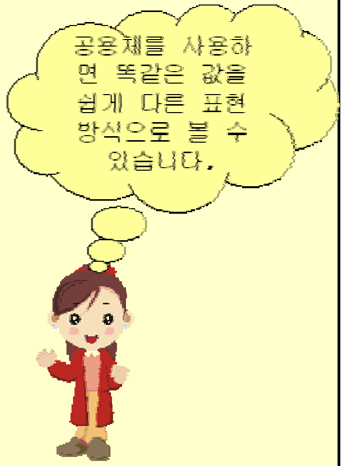
int 형으로 참조.

```
v.c:A v.i:65
v.c:□ v.i:10000
```

ip 주소 예제

```c
#include <stdio.h>

union ip_address {
      unsigned long laddr;
      unsigned char saddr[4];
};

int main(void)
{
      union ip_address addr;

      addr.saddr[0] = 1;
      addr.saddr[1] = 0;
      addr.saddr[2] = 0;
      addr.saddr[3] = 127;

      printf("%x\n", addr.laddr);

      return 0;
}
```




공용체를 사용하면 똑같은 값을 쉽게 다른 표현 방식으로 볼 수 있습니다.

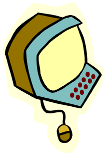
```
7f000001
```

타입 필드를 같이 사용하는 예

```
#include <stdio.h>

#define STU_NUMBER 1
#define REG_NUMBER 2

struct  student {
        int type;
        union {
                int stu_number;             // 학번
                char reg_number[15];        // 주민등록번호
        } id;
        char name[20];
};
void print(struct student s)
{
        switch(s.type)
        {
                case STU_NUMBER:
                        printf("학번: %d\n", s.id.stu_number);
                        printf("이름: %s\n", s.name);
                        break
                case REG_NUMBER:
                        printf("주민등록번호: %d\n", s.id.reg_number);
                        printf("이름: %s\n", s.name);
                        break
                default:
                        printf("타입오류\n");
                        break
        }
}
```

타입 필드를 같이 사용하는 예

```
int main(void)
{
	struct student s1, s2;

	s1.type = STU_NUMBER;
	s1.id.stu_number = 20070001;
	strcpy(s1.name, "홍길동");

	s2.type = REG_NUMBER;
	strcpy(s2.id.reg_number, "860101-1058031");
	strcpy(s2.name, "김철수");

	print(s1);
	print(s2);

	return 0;
}
```



```
학번: 20070001
이름: 홍길동
주민등록번호: 1244868
이름: 김철수
```

열거형

- *열거형(enumeration)*이란 변수가 가질 수 있는 값들을 미리 열거해 놓은 자료형
- (예) 요일을 저장하고 있는 변수는 { 일요일, 월요일, 화요일, 수요일, 목요일, 금요일, 토요일 } 중의 하나의 값만 가질 수 있다.
- 열거형은 *enum*이라는 키워드를 사용하여 만들어진다.

```
enum 태그_이름 { 값1, 값2, ... };
```

```
enum days1 { MON, TUE, WED, THU, FRI, SAT, SUN };
enum days2 { MON=1, TUE, WED, THU, FRI, SAT, SUN };
enum days3 { MON=1, TUE=2, WED=3, THU=4, FRI=5, SAT=6, SUN=7 };
enum days4 { MON, TUE=2, WED=3, THU, FRI, SAT, SUN };

enum days1 d;
d = WED;
```

열거형의 예

```
enum days  { SUN, MON, TUE, WED, THU, FRI, SAT };

enum colors { white, red, blue, green, black };

enum boolean { 0, 1 };

enum months { JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };

enum major { COMMUNICATION, COMPUTER, ELECTRIC, ELECTRONICS };

enum component { MAIN_BOARD, CPU, GRAPHIC_CARD, DISK, MEMORY };

enum levels { low = 1, medium, high };

enum CarOptions
{
    SunRoof = 0x01,
    Spoiler = 0x02,
    FogLights = 0x04,
    TintedWindows = 0x08,
}
```

열거형과 다른 방법과의 비교

정수 사용

```
switch(code) {
  case 1:
    printf("LCD TV\n");
    break;
  case 2:
    printf("PDP TV\n");
    break;
}
```

컴퓨터는 알기 쉬우나 사람은 기억하기 어렵다.

기호 상수

```
#define LCD 1
#define PDP 2

switch(code) {
  case LCD:
    printf("LCD TV\n");
    break;
  case PDP:
    printf("PDP TV\n");
    break;
}
```

기호 상수를 작성할 때 오류를 저지를 수 있다.

열거형

```
enum tvtype { LCD, PDP };
enum tvtype code;

switch(code) {
  case LCD:
    printf("LCD TV\n");
    break;
  case PDP:
    printf("PDP TV\n");
    break;
}
```

컴파일러가 중복이 일어나지 않도록 체크한다.

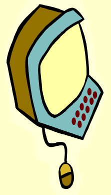
예제

```
// 열거형
#include <stdio.h>

enum days { MON, TUE, WED, THU, FRI, SAT, SUN };

char *days_name[] = {
"monday", "tuesday", "wednesday", "thursday", "friday",
      "saturday", "sunday" };

int main(void)
{
      enum days d;

      for(d=MON; d<=SUN; d++)
      {
               printf("%d번째 요일의 이름은 %s입니다\n", d, days_name[d]);
      }
}
```

```
0번째 요일의 이름은 monday입니다
1번째 요일의 이름은 tuesday입니다
2번째 요일의 이름은 wednesday입니다
3번째 요일의 이름은 thursday입니다
4번째 요일의 이름은 friday입니다
5번째 요일의 이름은 saturday입니다
6번째 요일의 이름은 sunday입니다
```

예제

```
#include <stdio.h>
enum tvtype { tube, lcd, plasma, projection };

int main(void)
{
	enum tvtype type;

	printf("TV 종류 코드를 입력하시오: ");
	scanf("%d", &type);
	switch(type)
	{
		case tube:
				printf("브라운관 TV를 선택하셨습니다.\n");
				break;
		case lcd:
				printf("LCD TV를 선택하셨습니다.\n");
				break;
		case plasma:
				printf("PDP TV를 선택하셨습니다.\n");
				break;
		case projection:
				printf("프로젝션 TV를 선택하셨습니다.\n");
				break;
		default:
				printf("다시 선택하여 주십시오.\n");
				break;
	}
	return 0;
}
```



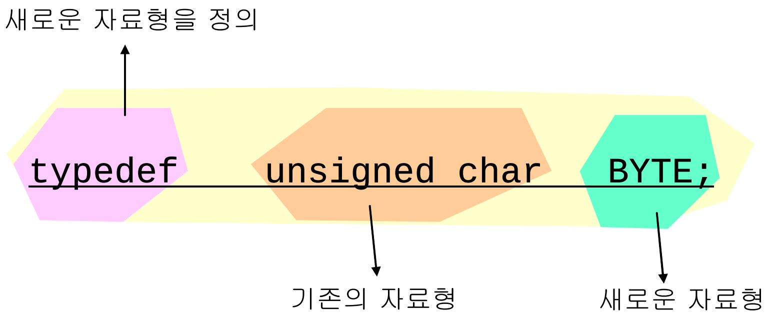
```
TV 종류 코드를 입력하시오: 3
프로젝션 TV를 선택하셨습니다.
```

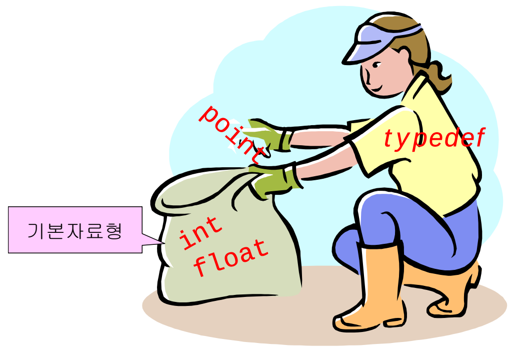
typedef

- typedef은 새로운 자료형(type)을 정의(define)

```
typedef     old_type     new_type;
```

- C의 기본 자료형을 확장시키는 역할





typedef의 예

기본자료형	재정의된 자료형
int	INT32
short	INT16
unsigned int	UINT32
unsigned short	UINT16
unsigned char	UCHAR, BYTE
char	CHAR

```
typedef int  INT32;
typedef unsigned int  UINT32;

INT32 i;                    // int i;와 같다.
UINT32 k;                   // unsigned int k;와 같다.


typedef struct point {
        int x;
        int y;
} POINT;

POINT p,q;
```

```
typedef struct complex {
        double real;
        double imag;
} COMPLEX;

COMPLEX x, y;

typedef enum { FALSE, TRUE } BOOL;

BOOL condition;            // enum { FALSE, TRUE }
condition;

typedef char * STRING_PTR;

STRING_PTR p;              // char *p;
```

typedef과 #define 비교

- 이식성을 높여준다.
  - 코드를 컴퓨터 하드웨어에 독립적으로 만들 수 있다
  - (예) int형은 2바이트이기도 하고 4바이트, int형 대신에 typedef을 이용한 INT32나 INT16을 사용하게 되면 확실하게 2바이트인지 4바이트인지를 지정할 수 있다.
- #define을 이용해도 typedef과 비슷한 효과를 낼 수 있다. 즉 다음과 같이 INT32를 정의할 수 있다.
  - #define UINT32 unsigned int
  - typedef float VECTOR[2];// #define으로는 불가능하다.
- 문서화의 역할도 한다.
  - typedef을 사용하게 되면 주석을 붙이는 것과 같은 효과

예제

```
#include <stdio.h>

typedef struct point {
	int x;
	int y;
} POINT;

POINT translate(POINT p, POINT delta);

int main(void)
{
	POINT p = { 2, 3 };
	POINT delta = { 10, 10 };
	POINT result;

	result = translate(p, delta);
	printf("새로운 점의 좌표는(%d, %d)입니다.\n", result.x, result.y);

	return 0;
}

POINT translate(POINT p, POINT delta)
{
	POINT new_p;

	new_p.x = p.x + delta.x;
	new_p.y = p.y + delta.y;

	return new_p;
}
```

```
새로운 점의 좌표는 (12, 13)입니다.
```

Q & A