

프로그래밍연습 Lab 10

문자와 문자열

[TA] 강성민, 김기현, 최석원, 최지은, 표지원

Department of Computer Science and Engineering

Seoul National University, Korea

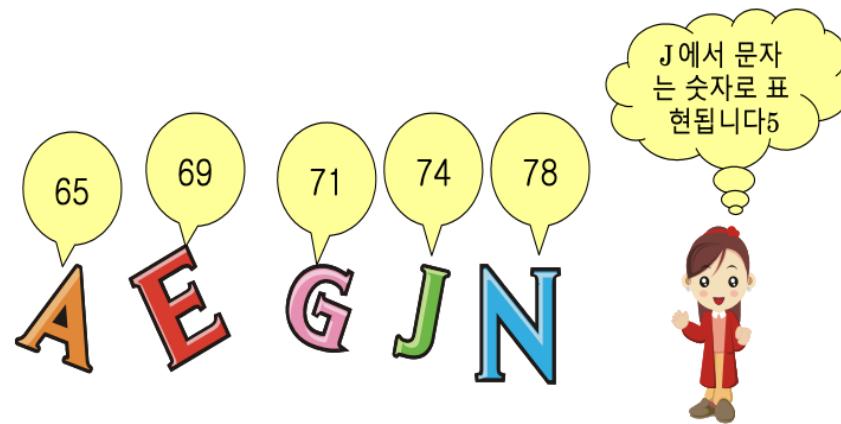
2022/11/16

이번 장에서 학습할 내용

- 문자 표현 방법
- 문자의 입출력
- 문자열 표현 방법
- 문자열이란 무엇인가?
- 문자열의 입출력
- 표준 입출력 라이브러리 함수

문자 표현 방법

- 컴퓨터에서는 각각의 문자에 숫자코드를 붙여서 표시한다.
- 아스키코드(ASCII code): 표준적인 8비트 문자코드
 - 0에서 127까지의 숫자를 이용하여 문자표현
- 유니코드(unicode): 표준적인 16비트 문자코드
 - 전세계의 모든 문자를 일관되게 표현하고 다룰 수 있도록 설계



아스키코드 출력

```
// 아스키 코드 출력
#include <stdio.h>
int main(void)
{
    unsigned char code;

    for(code = 32; code < 128; code++)
    {
        printf("아스키 코드 %d은 %c입니다.\n", code, code);
    }
    return 0;
}
```

32 : (공백) 부터
127 : □ 까지 출력



아스키 코드 32은 입니다.
아스키 코드 33은 !입니다.
...
아스키 코드 65은 A입니다.
아스키 코드 66은 B입니다.
...
아스키 코드 97은 a입니다.
아스키 코드 98은 b입니다.
...
아스키 코드 126은 ~입니다.
아스키 코드 127은 □입니다.

문자 변수와 문자 상수

문자변수

문자상수

```
// 문자 상수
#include <stdio.h>

int main(void)
{
    char code1 = 'A';
    char code2 = 65;

    printf("code1=%c, code1=%d\n", code1,code1);
    printf("code2=%c, code2=%d\n", code2,code2);
    return 0;
}
```



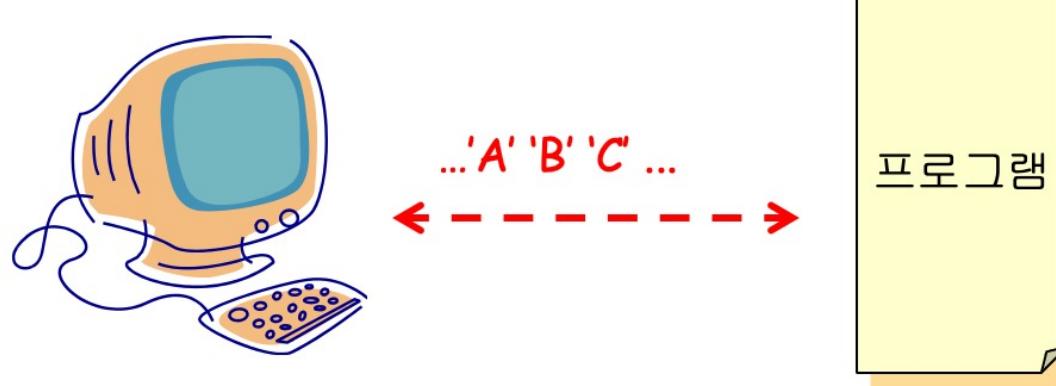
65



```
code1=A, code1=65
code2=A, code2=65
```

문자 입출력 라이브러리

입출력 함수	설명
<code>int getchar(void)</code>	하나의 문자를 읽어서 반환한다.
<code>void putchar(int c)</code>	변수 <code>c</code> 에 저장된 문자를 출력한다.
<code>int getch(void)</code>	하나의 문자를 읽어서 반환한다(버퍼를 사용하지 않음).
<code>void putch(int c)</code>	변수 <code>c</code> 에 저장된 문자를 출력한다(버퍼를 사용하지 않음).
<code>scanf("%c", &c)</code>	하나의 문자를 읽어서 변수 <code>c</code> 에 저장한다.
<code>printf("%c", c);</code>	변수 <code>c</code> 에 저장된 문자를 출력한다.



getchar() / putchar()

```
// getchar()의 사용
#include <stdio.h>
int main(void)
{
    int ch;           // 정수형에 주의
    while( (ch = getchar()) != EOF )
        putchar(ch);
    return 0;
}
```

End Of File을
나타내는 문자,
EOF는 정수형
이다.

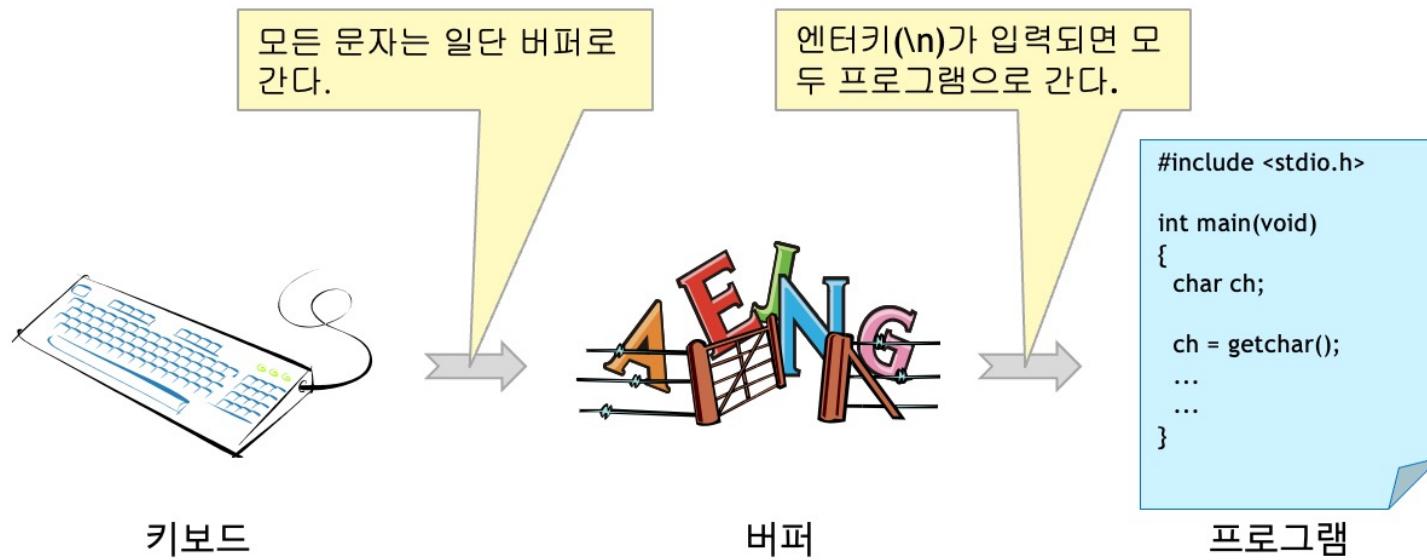


A
A
B
B
q

getchar() 출력
putchar() 출력

버퍼링

- 엔터키를 쳐야만 입력을 받는 이유



_getch(), _putch()

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

int main(void)
{
    int ch;
    while( (ch = _getch()) != 'q' )
        _putch(ch);
    return 0;
}
```

버퍼를 사용하지
않는다,
에코도 없음!



ABCDEFGH

getchar(), getche(), getchar() 비교

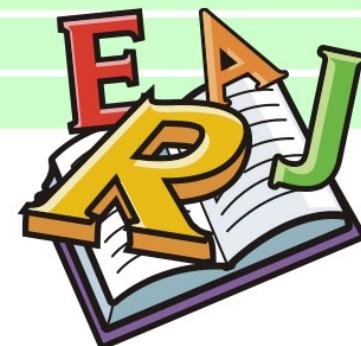
	헤더파일	버퍼사용여부	에코여부	응답성	문자수정여부
getchar()	<stdio.h>	사용함 (엔터키를 눌러입력됨)	에코	줄단위	가능
_getch()	<conio.h>	사용하지 않음	에 코 하 지 않음	문자단위	불가능
_getche()	<conio.h>	사용하지 않음	에코	문자단위	불가능



문자 처리 라이브러리 함수

- 문자를 검사하거나 문자를 변환한다.

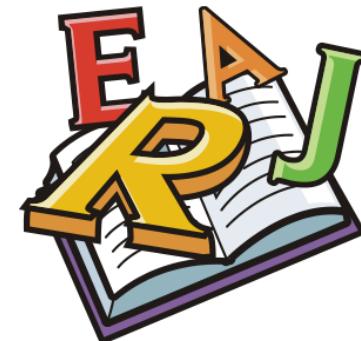
함수	설명
isalpha(c)	c가 영문자인가?(a-z, A-Z)
isupper(c)	c가 대문자인가?(A-Z)
islower(c)	c가 소문자인가?(a-z)
isdigit(c)	c가 숫자인가?(0-9)
isalnum(c)	c가 영문자이나 숫자인가?(a-z, A-Z, 0-9)
isxdigit(c)	c가 16진수의 숫자인가?(0-9, A-F, a-f)
isspace(c)	c가 공백문자인가?(' ', '\n', '\t', '\v', '\r')
ispunct(c)	c가 구두점 문자인가?
isprint(c)	C가 출력가능한 문자인가?
iscntrl(c)	c가 제어 문자인가?
isascii(c)	c가 아스키 코드인가?



문자 처리 라이브러리 함수 cont'd

- 문자를 검사하거나 문자를 변환한다.

함수	설명
toupper(c)	c를 대문자로 바꾼다.
tolower(c)	c를 소문자로 바꾼다.
toascii(c)	c를 아스키 코드로 바꾼다.



문자 처리 라이브러리 함수 활용 예제 (1)

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

```
int main( void )
```

```
{
```

```
    int c;
```

```
    while((c = getchar()) != EOF)
```

```
{
```

```
        if( islower(c) )
            c = toupper(c);
```

```
        putchar(c);
    }
```

```
    return 0;
}
```

소문자인지 검사

대문자로 변환



```
abcdef
ABCDEF
^Z
```

EOF를 키보드에서 입력하려면 ^Z

문자 처리 라이브러리 함수 활용 예제 (2)

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

int main( void )
{
    int c;

    while((c = getch()) != 'z')
    {
        printf("-----\n");
        printf("isdigit(%c) = %d\n", c, isdigit(c));
        printf("isalpha(%c) = %d\n", c, isalpha(c));
        printf("islower(%c) = %d\n", c, islower(c));
        printf("ispunct(%c) = %d\n", c, ispunct(c));
        printf("isxdigit(%c) = %d\n", c, isxdigit(c));
        printf("isprint(%c) = %d\n", c, isprint(c));
        printf("-----\n\n");
    }
    return 0;
}
```

숫자인지 검사
알파벳인지 검사
소문자인지 검사
구두점 문자인지 검사
16진수인지 검사
출력가능한지 검사



```
-----  
isdigit(') = 0  
isalpha(') = 0  
islower(') = 0  
ispunct(') = 16  
isxdigit(') = 0  
isprint(') = 16  
-----  
...
```

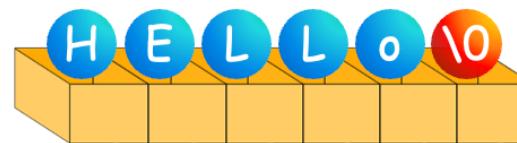
문자열 표현 방법

- **문자열(string):** 문자들이 여러 개 모인 것
 - "A "
 - "Hello World!"
 - "변수 score의 값은 %d입니다"
- **문자열 변수**
 - 변경 가능한 문자열을 저장할 수 있는 변수
 - 어디에 저장하면 좋은가?

문자열 상수



하나의 문자는 char형 변수로 저장



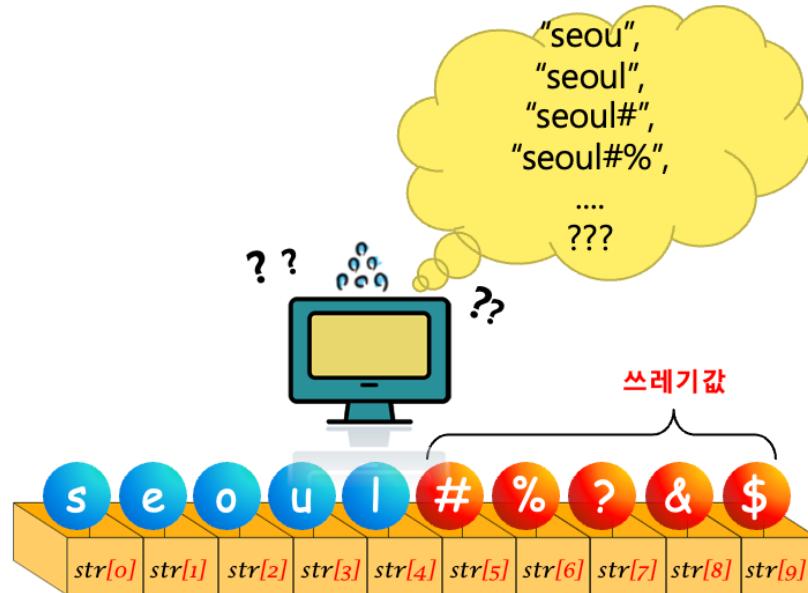
문자열은 char형 배열로 저장

NULL 문자

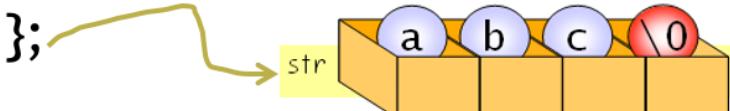
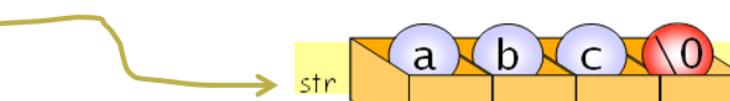
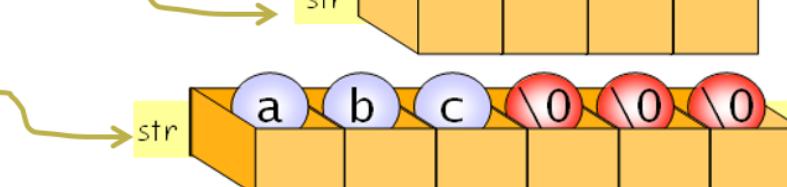
- NULL 문자: 문자열의 끝을 나타낸다.



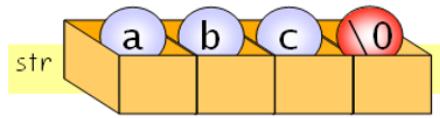
- 문자열은 어디서 종료되는지 알수가 없으므로 표시를 해주어야 한다.



문자열 변수의 초기화

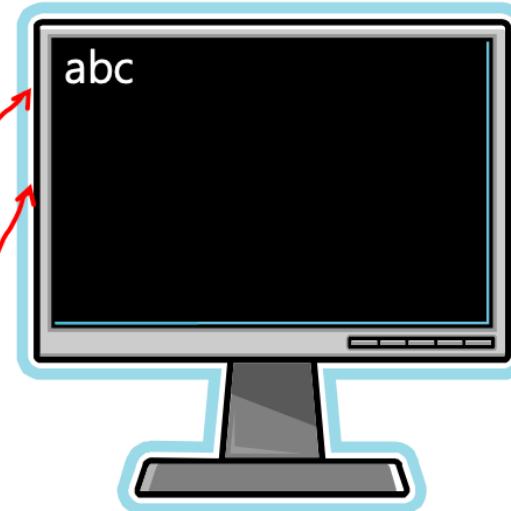
- `char str[4] = { 'a', 'b', 'c', '\0' };` 
- `char str[4] = "abc";` 
- `char str[4] = "abcdef";` 
- `char str[6] = "abc";` 
- `char str[4] = "";` 
- `char str[] = "abc";` 

문자열 출력



```
char str[] = "abc";  
printf("%s", str);
```

```
char str[] = "abc";  
printf(str);
```

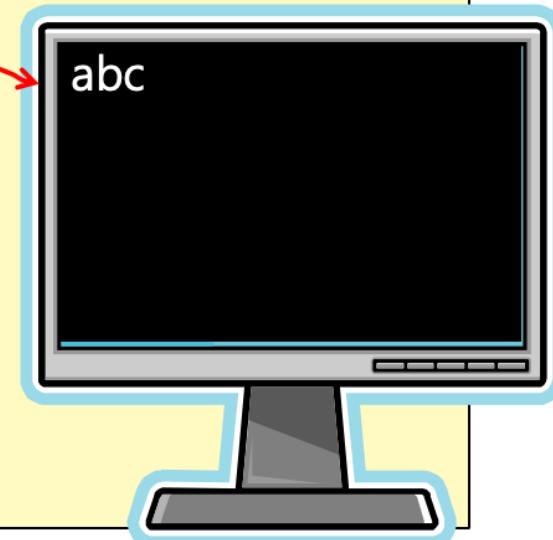


문자열 출력 예제 (1)

```
#include <stdio.h>
int main(void)
{
    int i;
    char str[4];
    str[0] = 'a';
    str[1] = 'b';
    str[2] = 'c';
    str[3] = '\0';

    i = 0;
    while(str[i] != '\0') {
        printf("%c", str[i]);
        i++;
    }
    return 0;
}
```

문자 배열에 들어 있는 문자들을 하나씩 출력하여 보자. 문자 배열에 저장된 문자열을 출력할 때는 %s를 사용하면 되지만 여기서는 문자열 처리의 기본적인 방법을 실감하기 위하여 문자 배열에 들어 있는 문자들을 하나씩 화면에 출력하다가 NULL 문자가 나오면 반복을 종료하도록 하였다.



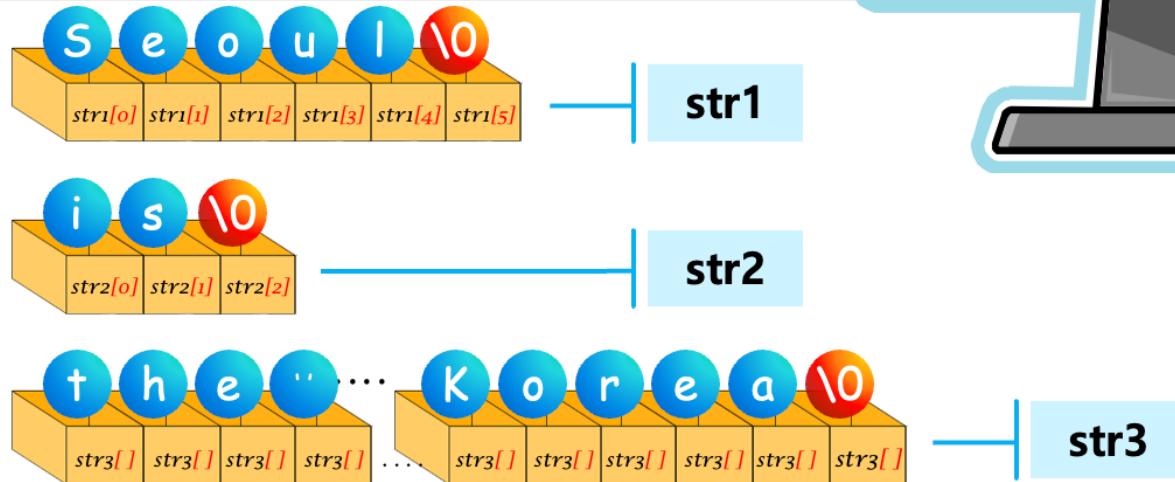
문자열 출력 예제 (2)

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

```
int main(void)
{
    char str1[6] = "Seoul";
    char str2[3] = { 'i', 's', '\0' };
    char str3[] = "the capital city of Korea./";

    printf("%s %s %s\n", str1, str2, str3);
}
```

Seoul is the capital city of Korea.



중간 실습

- 다음과 같은 결과를 출력하는 프로그램을 작성해보자.



중간 실습 결과

```
#include <stdio.h>
int main(void)
{
    char src[] = "The worst things to eat before you sleep";
    char dst[100];
    int i;
    printf("원본 문자열=%s\n", src);
    for(i=0 ; src[i] != NULL ; i++)
        dst[i] = src[i];
    dst[i] = NULL;
    printf("복사된 문자열=%s\n", dst);
    return 0;
}
```

NULL 문자가 나올 때까지 반복하면서 각각의 문자들을 새로운 배열로 복사한다.

문자열 상수

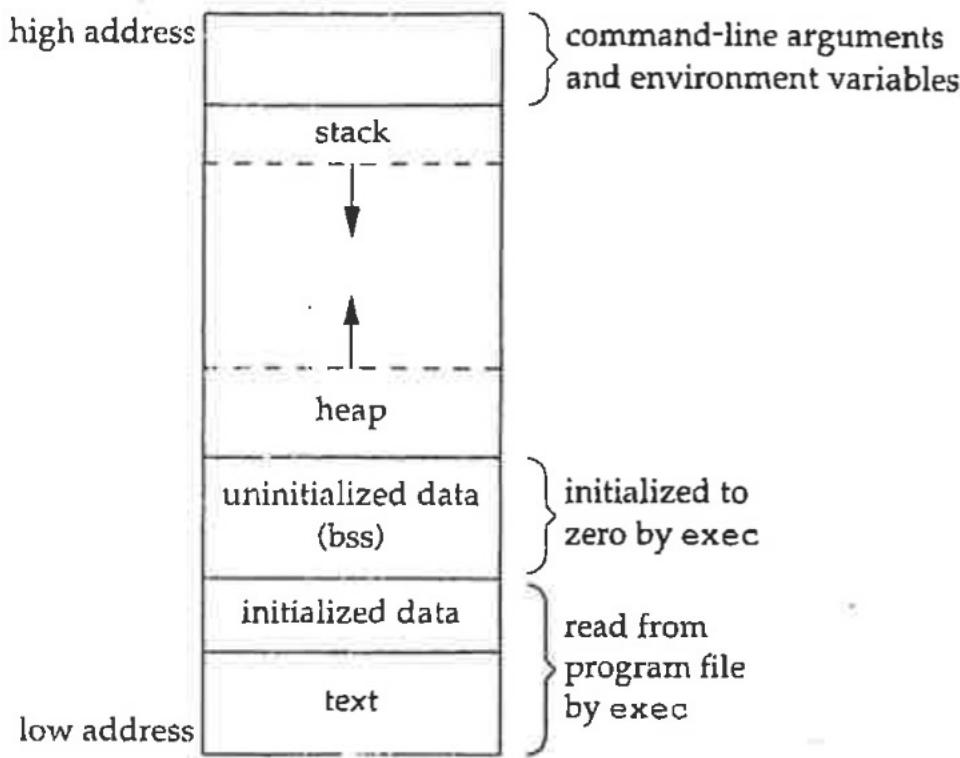
- 문자열 상수: “HelloWorld”와 같이 프로그램 소스 안에 포함된 문자열
- 문자열 상수는 메모리 영역 중에서 텍스트 세그먼트(**text segment**)에 저장

```
char *p = "HelloWorld";
```

위 문장의 정
확한 의미는
무엇일까요?



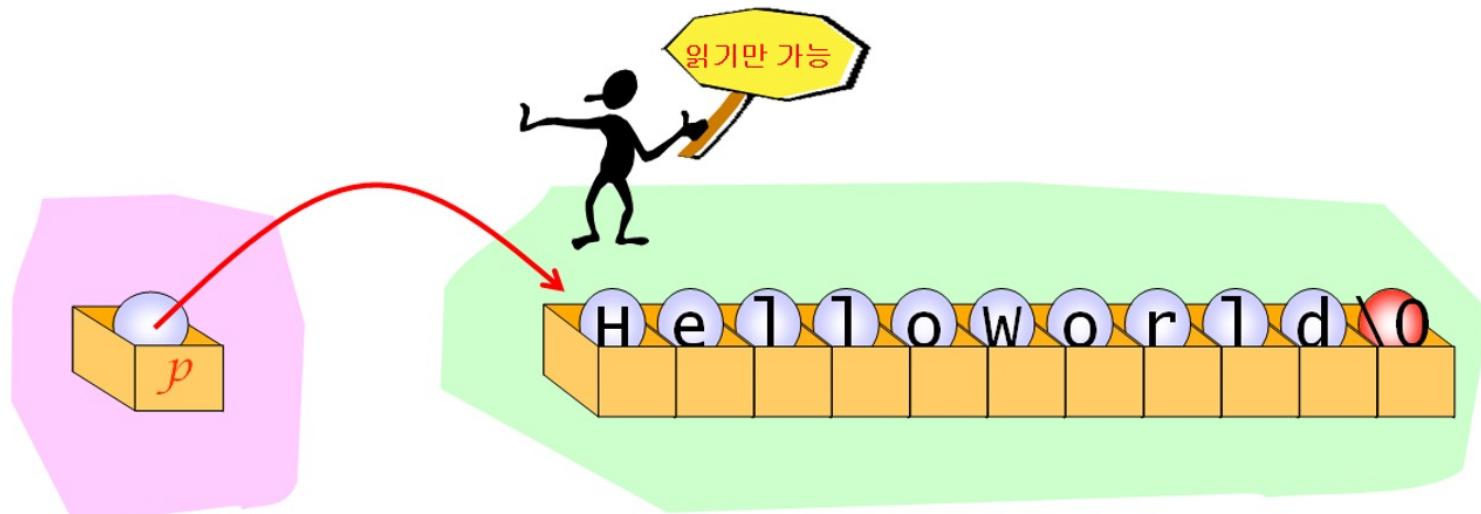
문자열 상수 cont'd



- Text segment. This is the machine instructions that are executed by the CPU. Usually the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs (text editors, the C compiler, the shells, etc.). Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.

문자열 상수 cont'd

```
char *p = "HelloWorld";
```



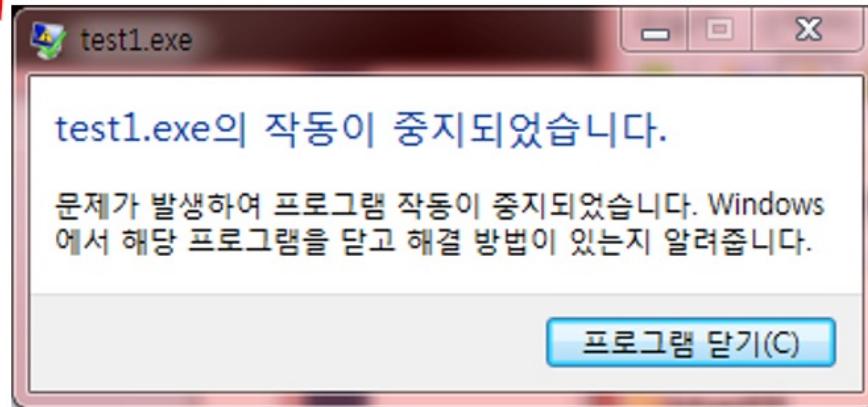
데이터 세그먼트(값을 변경
할 수 있는 메모리 영역)

텍스트 세그먼트(값을 읽기만 하고
변경할 수는 없는 메모리 영역)

문자열 상수 cont'd

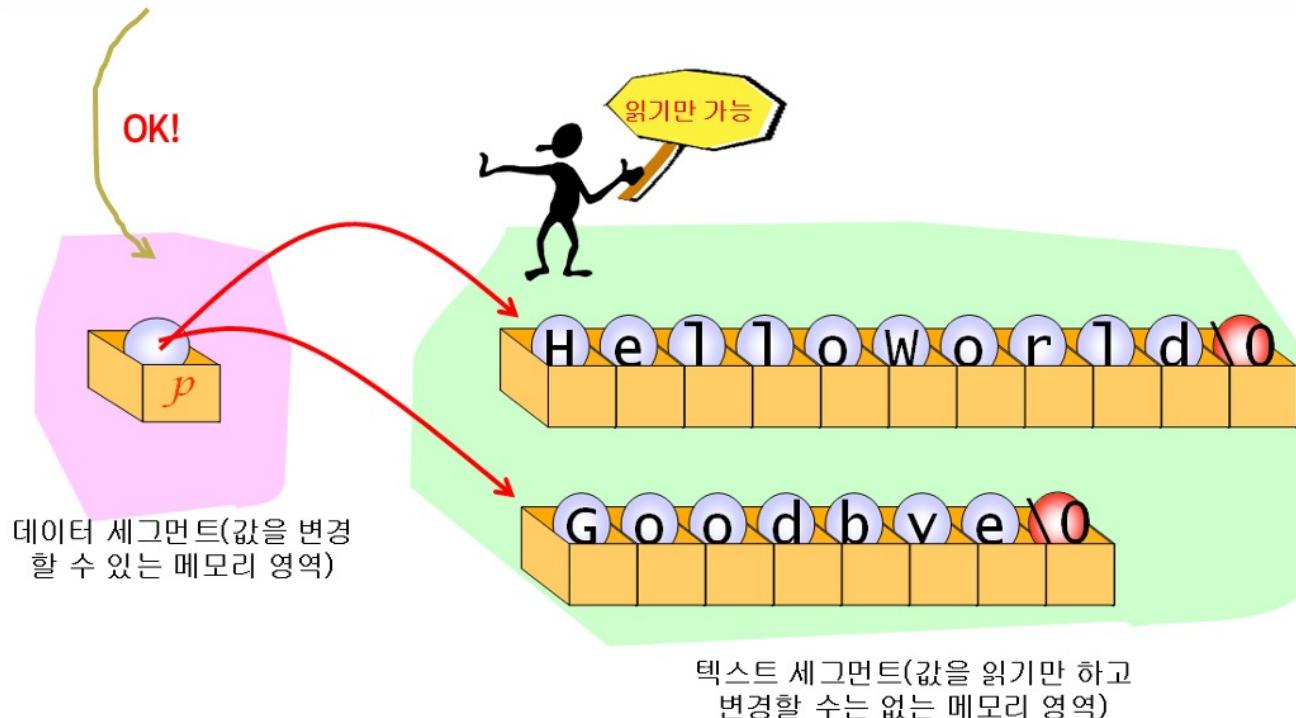
```
char *p = "HelloWorld";
p[0] = 'A';           // 또는 strcpy(p, "Goodbye");
```

p를 통하여 텍스트 세그먼트에 문자를
저장하려면 오류가 발생한다.



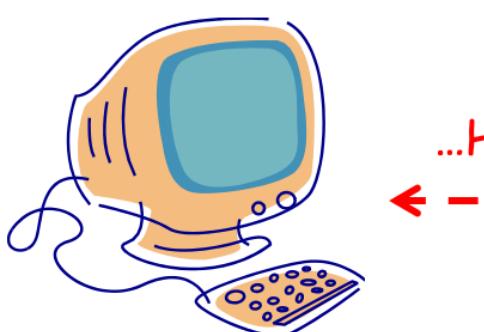
문자열 상수 cont'd

- `char *p = "HelloWorld";`
`p = "Goodbye";`

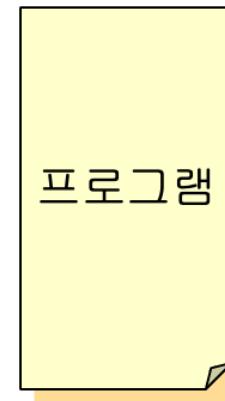


문자열 입출력 라이브러리

입출력 함수	설명
int scanf("%s", s)	문자열을 읽어서 문자배열 s[]에 저장
int printf("%s", s)	배열 s[]에 저장되어 있는 문자열을 출력한다.
char *gets(char *s)	한 줄의 문자열을 읽어서 문자 배열 s[]에 저장한다.
int puts(const char *s)	배열 s[]에 저장되어 있는 한 줄의 문자열을 출력한다.



...Hello World!...

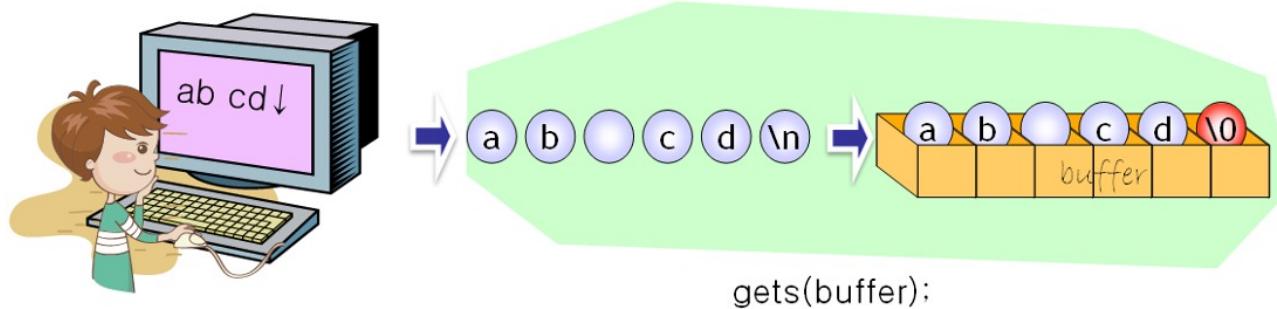


gets() 와 puts() 문자열 입출력

```
char *gets(char *buffer);
int puts(const char *str);
```

- **gets()**

- 표준 입력으로부터 엔터키가 나올 때까지 한 줄의 라인을 입력
- 문자열에 줄바꿈 문자('\n')는 포함되지 않으며 대신에 자동으로 **NULL** 문자('\0')를 추가한다.
- 입력받은 문자열은 **buffer**가 가리키는 주소에 저장된다.



gets() 와 puts() 문자열 입출력 cont'd

```
char *gets(char *buffer);
int puts(const char *str);
```

- **puts()**
 - str이 가리키는 문자열을 받아서 화면에 출력
 - NULL 문자('\0')는 줄바꿈 문자('\n')로 변경

```
char *menu = "파일열기: open, 파일닫기: close";
puts(str);
```

gets() / puts() 문자열 입출력 예제

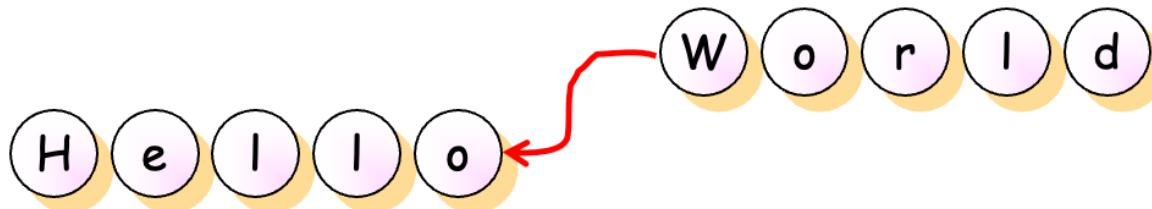
```
#include <stdio.h>
int main(void)
{
    char name[100];
    char address[100];
    printf("이름을 입력하시오: ");
    gets(name);
    printf("현재 거주하는 주소를 입력하시오: ");
    gets(address);
    puts(name);
    puts(address);
    return 0;
}
```

한 단어 이상을 입력
받을 때에 사용한다.



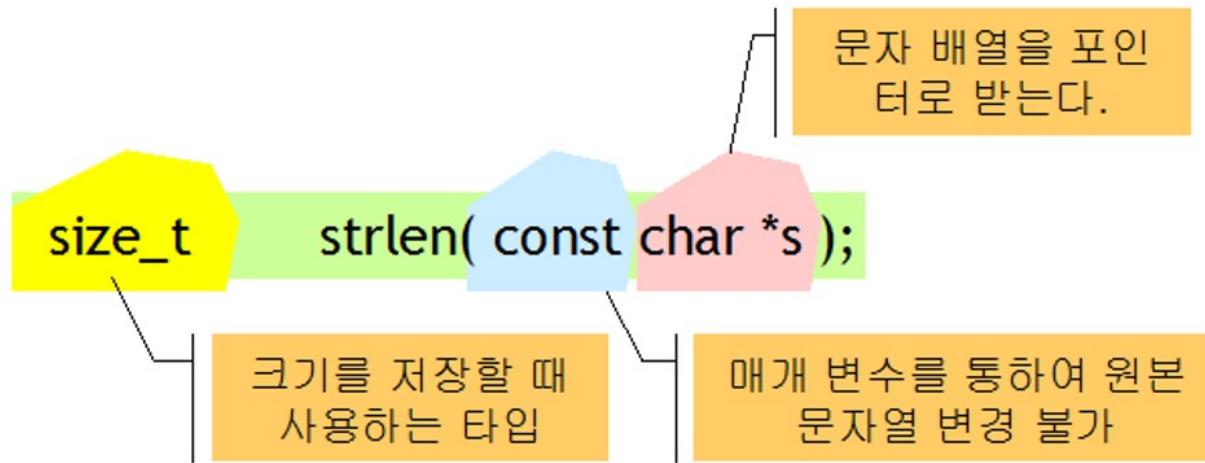
문자열 처리 라이브러리

함수	설명
strlen(s)	문자열 s의 길이를 구한다.
strcpy(s1, s2)	s2를 s1에 복사한다.
strcat(s1, s2)	s2를 s1의 끝에 붙여넣는다.
strcmp(s1, s2)	s1과 s2를 비교한다.
strncpy(s1, s2, n)	s2의 최대 n개의 문자를 s1에 복사한다.
strncat(s1, s2, n)	s2의 최대 n개의 문자를 s1의 끝에 붙여넣는다.
strncmp(s1, s2, n)	최대 n개의 문자까지 s1과 s2를 비교한다.
strchr(s, c)	문자열 s안에서 문자 c를 찾는다.
strstr(s1, s2)	문자열 s1에서 문자열 s2를 찾는다.



strlen()

- 문자열의 길이
 - strlen("Hello")는 5를 반환



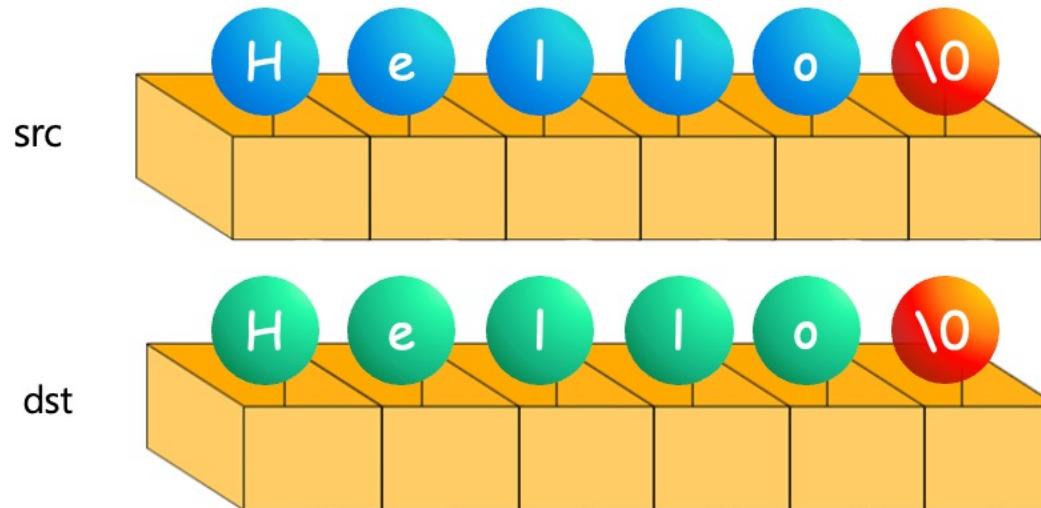
strcpy()

- 문자열 복사

```
char dst[6];
```

```
char src[6] = "Hello";
```

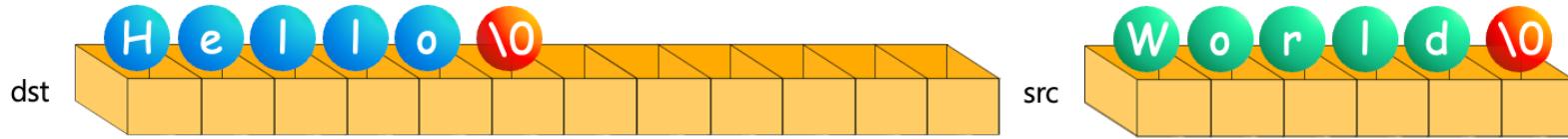
```
strcpy(dst, src);
```



strcat()

- 문자열 연결

```
char dst[12] = "Hello";
char src[6] = "World";
strcat(dst, src);
```



문자열 처리 라이브러리 활용 예제 (1)

```
// strcpy와 strcat
#include <string.h>
#include <stdio.h>

int main( void )
{
    char string[80];

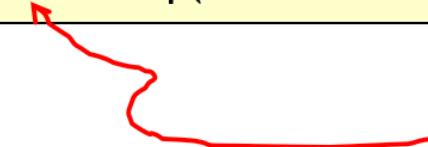
    strcpy( string, "Hello world from " );
    strcat( string, "strcpy" );
    strcat( string, "and " );
    strcat( string, "strcat!" );
    printf( "string = %s\n", string );
    return 0;
}
```



string = Hello world from strcpy and strcat!

strcmp()

```
int strcmp( const char *s1, const char *s2 );
```



반환값	s1과 s2의 관계
<0	s1이 s2보다 앞에 있다.
0	s1이 s2와 같다.
>0	s1이 s2보다 뒤에 있다.

문자열이 같으면
strcmp()는 0을
반환합니다. 주의
하세요!



문자열 처리 라이브러리 활용 예제 (2)

```
// strcmp() 함수
#include <string.h>
#include <stdio.h>

int main( void )
{
    char s1[80];      // 첫번째 단어를 저장할 문자배열
    char s2[80];      // 두번째 단어를 저장할 문자배열
    int result;

    printf("첫번째 단어를 입력하시오:");
    scanf("%s", s1);
    printf("두번째 단어를 입력하시오:");
    scanf("%s", s2);
```

문자열 처리 라이브러리 활용 예제 (2) cont'd

```
result = strcmp(s1, s2);
if( result < 0 )
    printf("%s가 %s보다 앞에 있습니다.\n", s1, s2);
else if( result == 0 )
    printf("%s가 %s와 같습니다.\n", s1, s2);
else
    printf("%s가 %s보다 뒤에 있습니다.\n", s1, s2);
return 0;
}
```

알파벳 순으로 즉 사전에
서 앞에 있다는 의미



첫번째 단어를 입력하시오:Hello
두번째 단어를 입력하시오:World
Hello가 World보다 앞에 있습니다.

strtok()

형식

char *strtok(char *s, const char *delimit);

설명

strtok 함수는 문자열 s을 토큰으로 분리한다.

만약 분리자가 ‘ ’일 경우, 토큰을 얻으려면 다음과 같이 호출한다.

```
t1 = strtok(s, " "); // 첫 번째 토큰  
t2 = strtok(NULL, " "); // 두 번째 토큰  
t3 = strtok(NULL, " "); // 세 번째 토큰  
t4 = strtok(NULL, " "); // 네 번째 토큰
```

문자열 처리 라이브러리 활용 예제 (3)

```
// strtok 함수의 사용예
```

```
#include <string.h>
```

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

```
char s[] = "Man is immortal, because he has a soul";
```

분리자

```
char seps[] = ",\t\n";
```

```
char *token;
```

```
int main( void )
```

```
{
```

// 문자열을 전달하고 다음 토큰을 얻는다.

```
token = strtok( s, seps );
```

```
while( token != NULL )
```

```
{
```

// 문자열 s에 토큰이 있는 동안 반복한다.

```
printf( "토큰: %s\n", token );
```

// 다음 토큰을 얻는다.

```
token = strtok( NULL, seps ); //
```

```
}
```

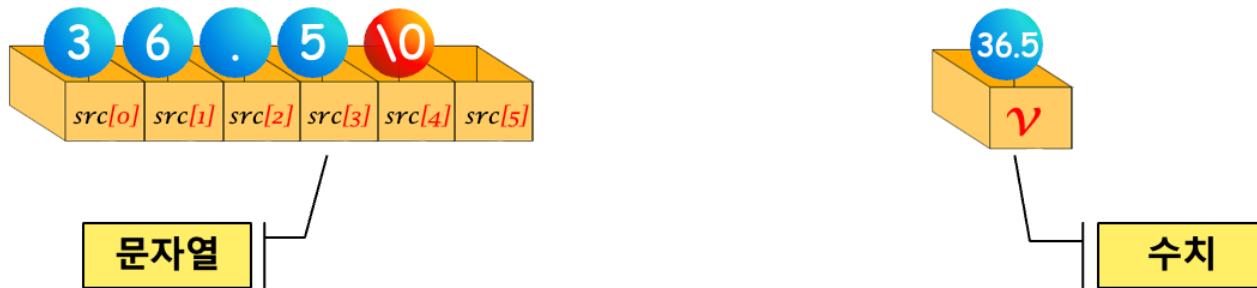
```
}
```



토큰: Man
토큰: is
토큰: immortal
토큰: because
토큰: he
토큰: has
토큰: a
토큰: soul

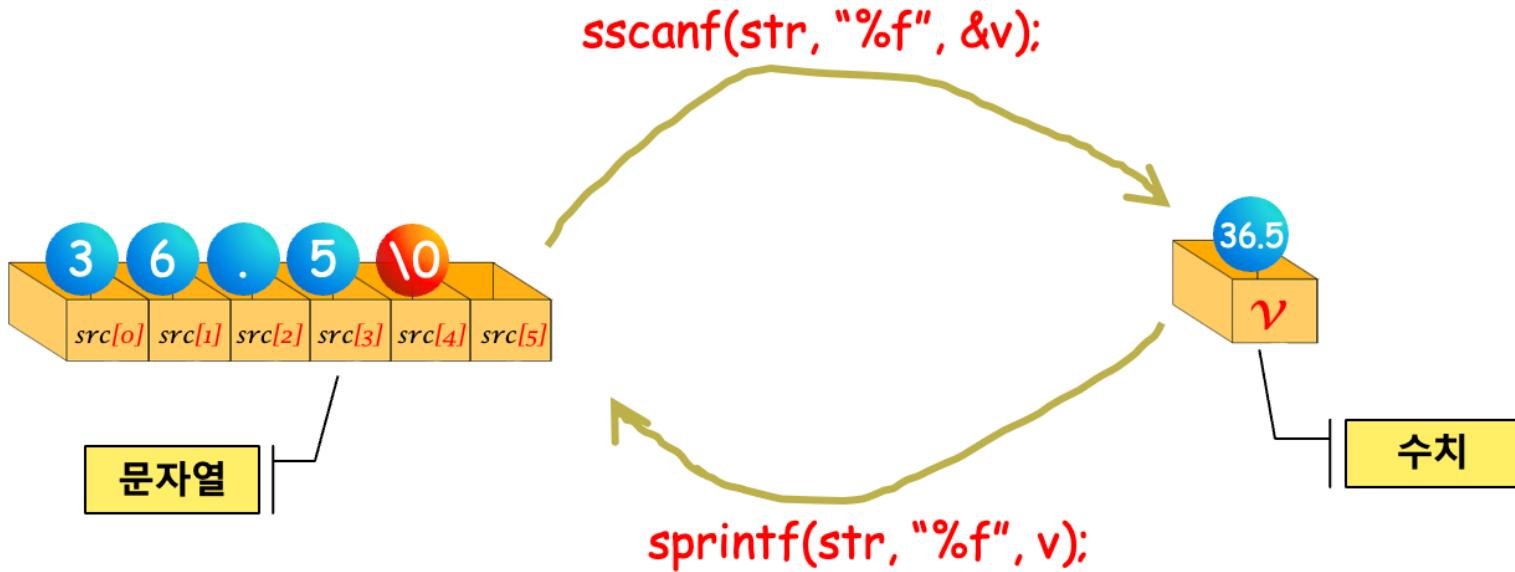
sprintf() / sscanf()

- 문자열 “36.5”와 수치값 36.5는 컴퓨터 안에서 상당히 다르게 저장된다.



sprintf() / sscanf() cont'd

- 앞에 붙은 s는 string을 의미한다.



문자열 처리 라이브러리 활용 예제 (4)

```
#include <stdio.h>
int main( void )
{
    char s1[] = "100 200 300";
    char s2[30];
    int value;

    sscanf(s1, "%d", &value);
    printf("%d\n", value);
    sprintf(s2, "%d", value);
    printf("%s\n", s2);

    return 0;
}
```

문자열에서 값을 추출 한다.

문자열에 값을 출력한다.

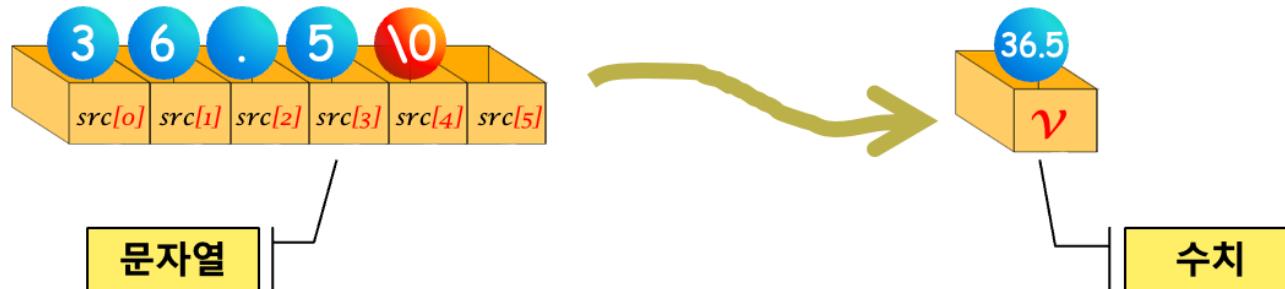


```
100
100
```

문자열을 수치로 전환하는 전용함수

- 전용 함수는 `scanf()`보다 크기가 작다.
- `stdlib.h`에 원형 정의- 반드시 포함

함수	설명
<code>int atoi(const char *str);</code>	<code>str</code> 을 <code>int</code> 형으로 변환한다.
<code>long atoi(const char *str);</code>	<code>str</code> 을 <code>long</code> 형으로 변환한다.
<code>double atof(const char *str);</code>	<code>str</code> 을 <code>double</code> 형으로 변환한다.



문자열 처리 라이브러리 활용 예제 (5)

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
    char s1[] = "100";
    char s2[] = "12.93";
    char buffer[100];
    int i;
    double d, result;

    i = atoi(s1);
    d = atof(s2);
    result = i + d;

    sprintf(buffer, "%f", result);
    printf("연산 결과는 %s입니다.\n", buffer);
    return 0;
}
```



연산 결과는 112.930000입니다.

프로그래밍연습 Lab 11

구조체

[TA] 강성민, 김기현, 최석원, 최지은, 표지원

Department of Computer Science and Engineering

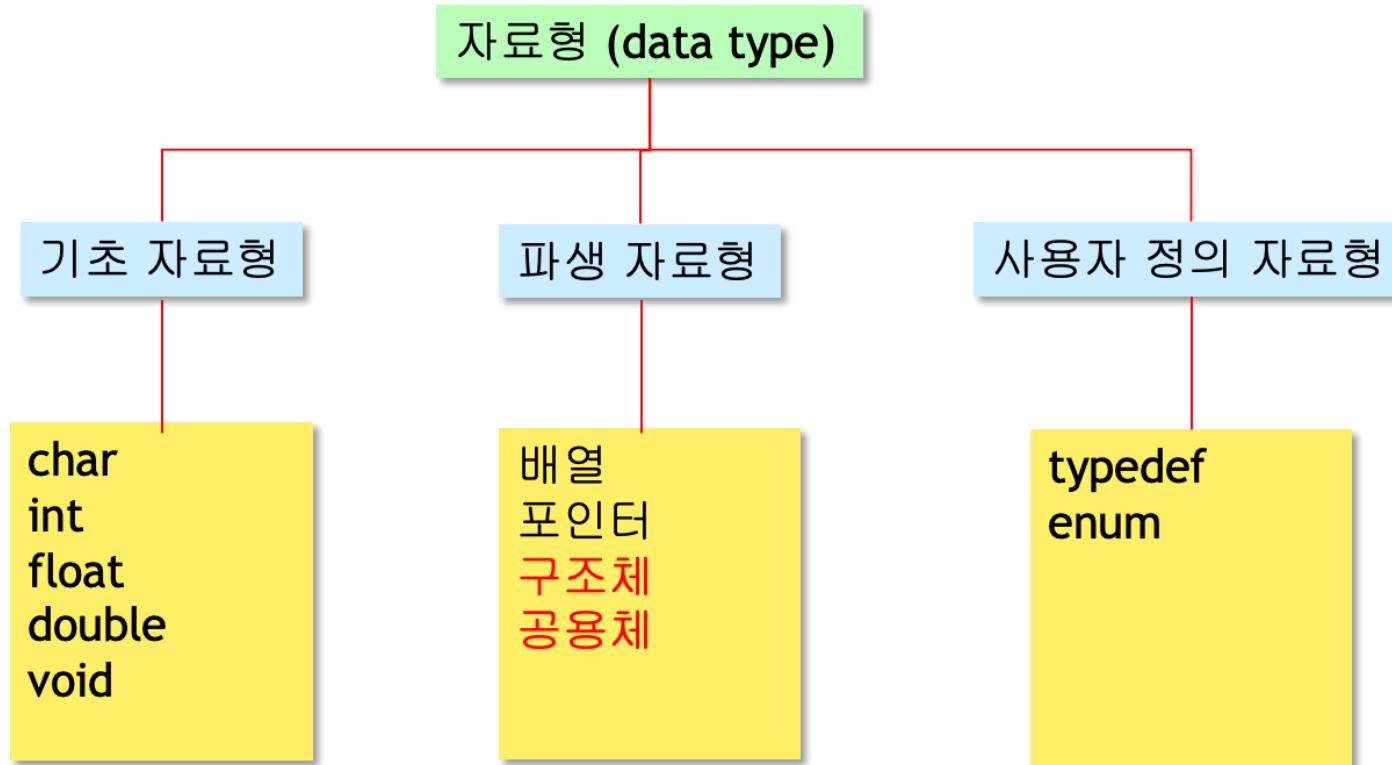
Seoul National University, Korea

2022/11/16

이번장에서 학습할 내용

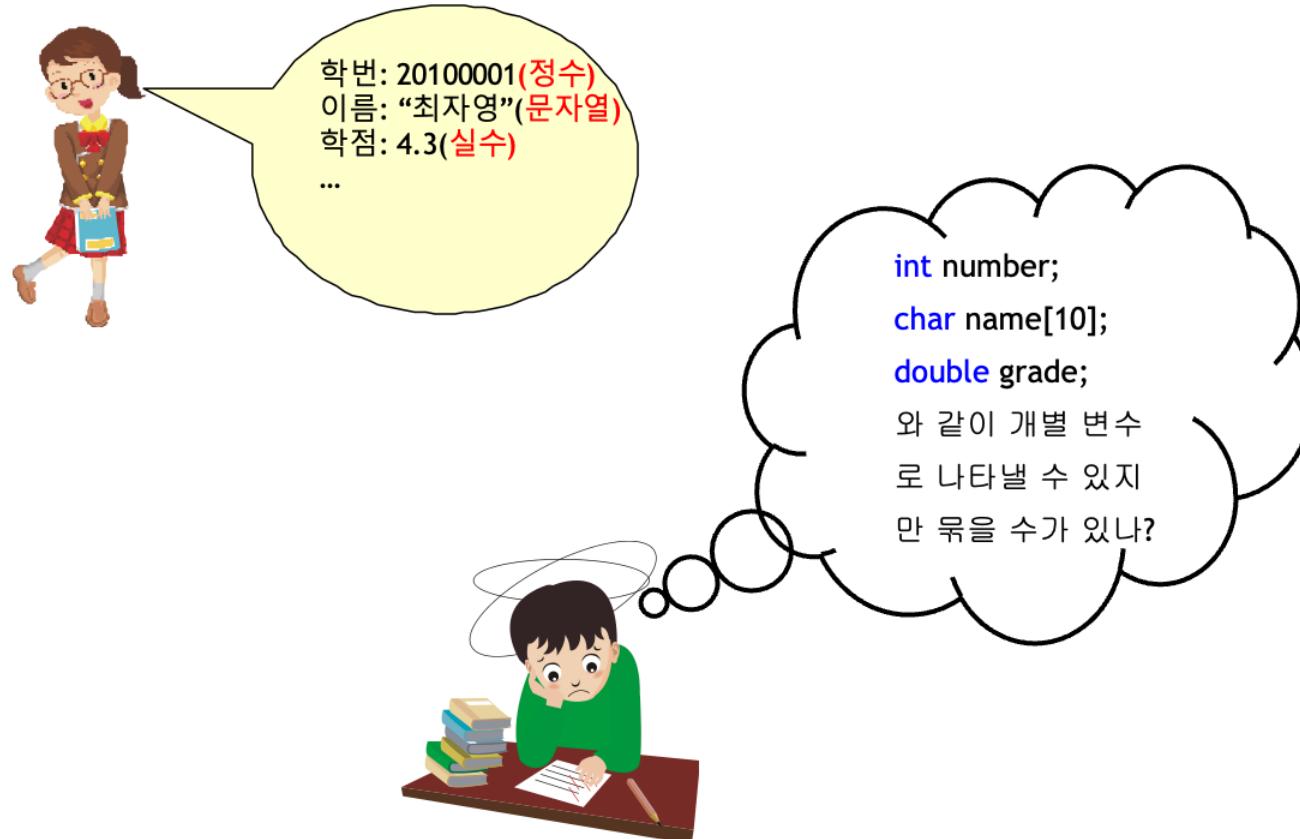
- 구조체의 개념, 정의, 초기화 방법
- 구조체와 포인터의 관계
- 공용체와 `typedef`

자료형의 분류



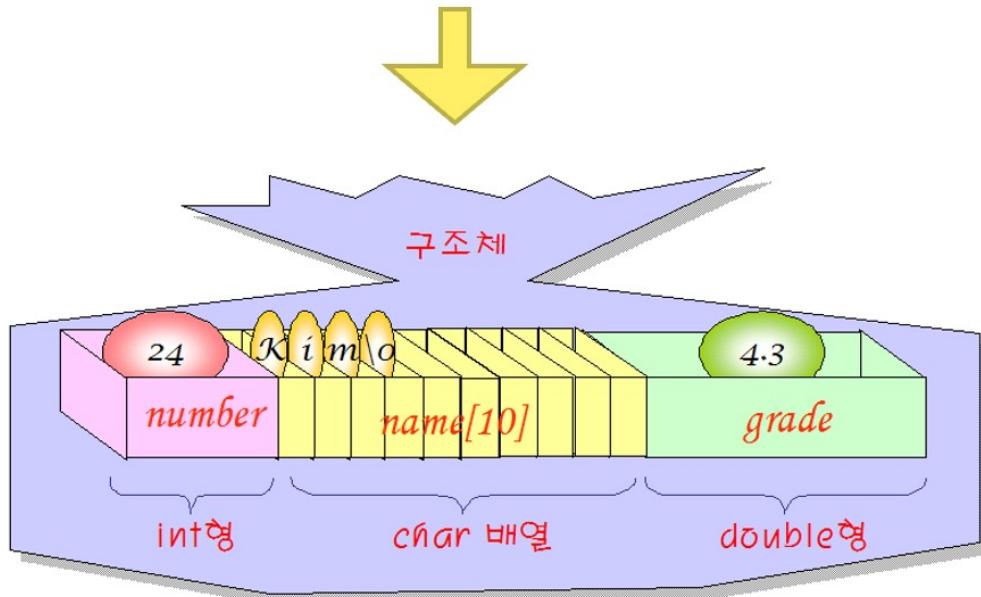
구조체의 필요성

- 학생에 대한 데이터를 하나로 모으려면?



구조체의 필요성 cont'd

```
int number;  
char name[10];  
double grade;
```

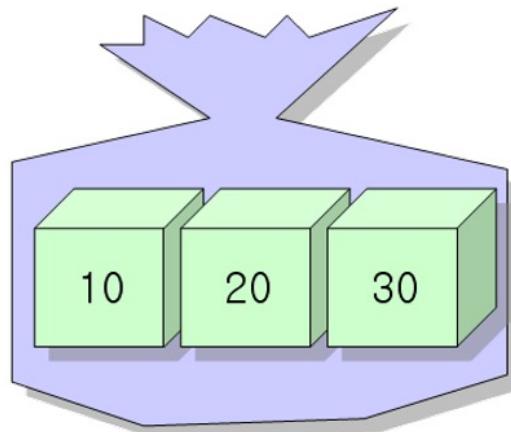


구조체를 사용하면 변수들을 하나로 묶을 수 있습니다.

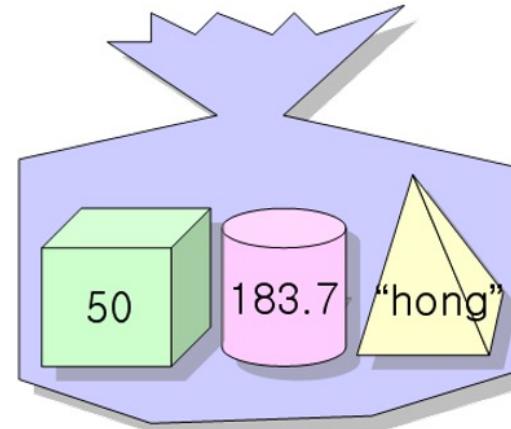


구조체 vs 배열

- 구조체 vs 배열



같은 타입의 집합



다른 타입의 집합

구조체의 선언

- 구조체 선언은 변수 선언은 아님

구조체를 정의하는 것은 와플이나 봉어빵을 만드는 틀을 정의하는 것과 같다.

와플이나 봉어빵을 실제로 만들릭 위해서는 구조체 변수를 선언하여야 한다.



구조체



구조체 변수

구조체 선언의 예시

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

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

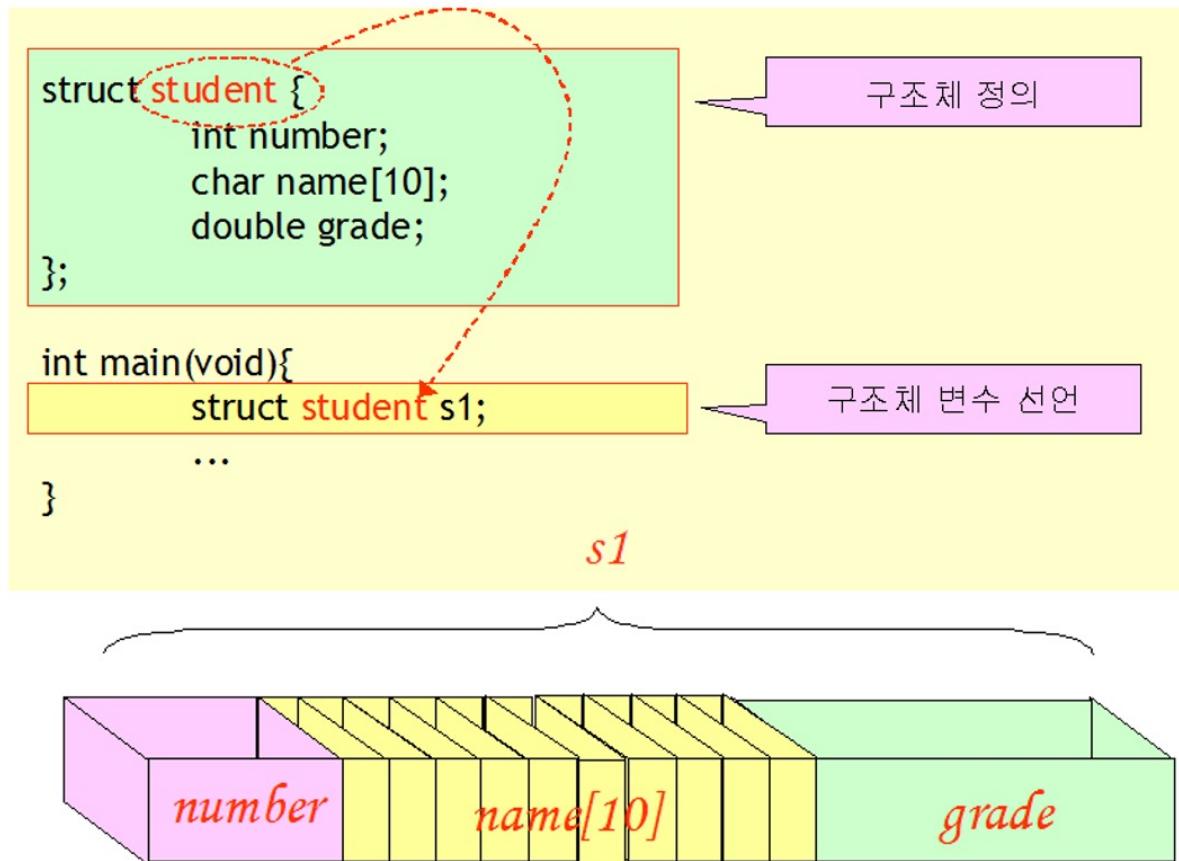
```
// 날짜
struct date {
    int month;
    int day;
    int year;
};
```

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

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

구조체 변수 선언

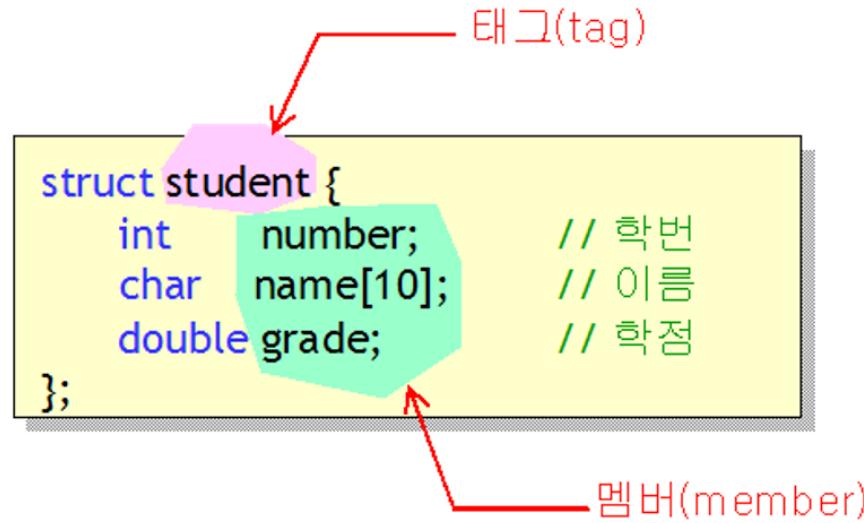
- 구조체 정의와 구조체 변수 선언은 다르다.



구조체의 선언 cont'd

- 구조체 선언 형식

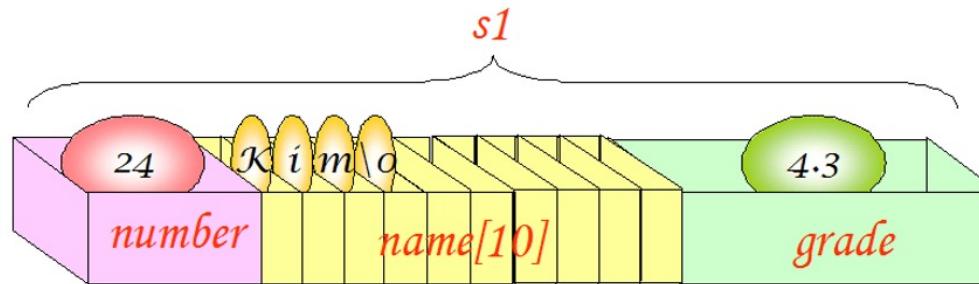
```
struct 태그 {  
    자료형      멤버1;  
    자료형      멤버2;  
    ...  
};
```



구조체 변수 초기화

- 중괄호를 이용하여 초기값을 나열한다.

```
struct student {  
    int number;  
    char name[10];  
    double grade;  
};  
struct student s1 = { 24, "Kim", 4.3 };
```



구조체 멤버 참조

- 구조체 멤버를 참조하려면 다음과 같이 .연산자를 사용한다.

```
s1.number = 26;           // 정수 멤버  
strcpy(s1.name, "Kim");  // 문자열 멤버  
s1.grade = 4.3;          // 실수 멤버
```



구조체 예제 (1)



```
...
struct student {
    int number;
    char name[10];
    double grade;
};
```

구조체 선언

```
int main(void)
```

```
{
```

```
    struct student s;
```

구조체 변수 선언

```
s.number = 20070001;
strcpy(s.name,"홍길동");
s.grade = 4.3;
```

구조체 멤버 참조

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



```
학번: 20070001
이름: 홍길동
학점: 4.300000
```

구조체 예제 (2)

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

구조체 선언



```
학번을 입력하시오: 20070001  
이름을 입력하시오: 홍길동  
학점을 입력하시오(실수): 4.3  
학번: 20070001  
이름: 홍길동  
학점: 4.300000
```

```
int main(void)
```

```
{
```

```
    struct student s;
```

구조체 변수 선언

```
printf("학번을 입력하시오: ");
```

```
scanf("%d", &s.number);
```

구조체 멤버의 주소 전달

```
printf("이름을 입력하시오: ");
```

```
scanf("%s", s.name);
```

```
printf("학점을 입력하시오(실수): ");
```

```
scanf("%lf", &s.grade);
```

```
printf("학번: %d\n", s.number);
```

```
printf("이름: %s\n", s.name);
```

```
printf("학점: %f\n", s.grade);
```

```
return 0;
```

```
}
```

다른 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 grade;
};
struct student s1; // 구조체 변수 선언
```

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

구조체 예제 (3)

```
#include <stdio.h>

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;
```



구조체 예제 (3) cont'd

```
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.y - r.p1.y;

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

return 0;
}
```



왼쪽 상단의 좌표를 입력하시오: 11
오른쪽 상단의 좌표를 입력하시오: 6 6
면적은 25이고 둘레는 20입니다.



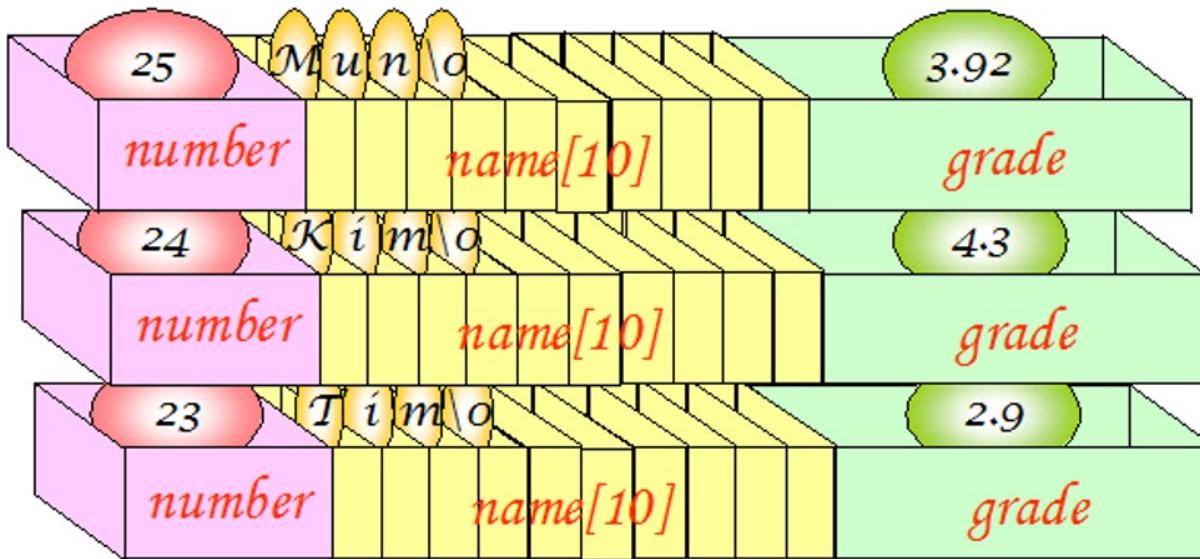
구조체 변수의 대입과 비교

- 같은 구조체 변수끼리 대입은 가능하지만 비교는 불가능하다.

```
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) )  
        printf("p1와 p2이 같습니다.")  
}
```

구조체 배열

- 구조체를 여러 개 모은 것



구조체 배열의 선언

- 구조체 배열의 선언

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

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

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

구조체 배열의 초기화

- 구조체 배열의 초기화

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

구조체 배열 예제

```
#define SIZE 3

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

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("학점을 입력하시오(실수): ");
        scanf("%lf", &list[i].grade);
    }

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



학번을 입력하시오: 20070001
이름을 입력하시오: 홍길동
학점을 입력하시오(실수): 4.3
학번을 입력하시오: 20070002
이름을 입력하시오: 김유신
학점을 입력하시오(실수): 3.92
학번을 입력하시오: 20070003
이름을 입력하시오: 이성계
학점을 입력하시오(실수): 2.87
학번: 20070001, 이름: 홍길동, 학점: 4.300000
학번: 20070002, 이름: 김유신, 학점: 3.920000
학번: 20070003, 이름: 이성계, 학점: 2.870000

구조체와 포인터



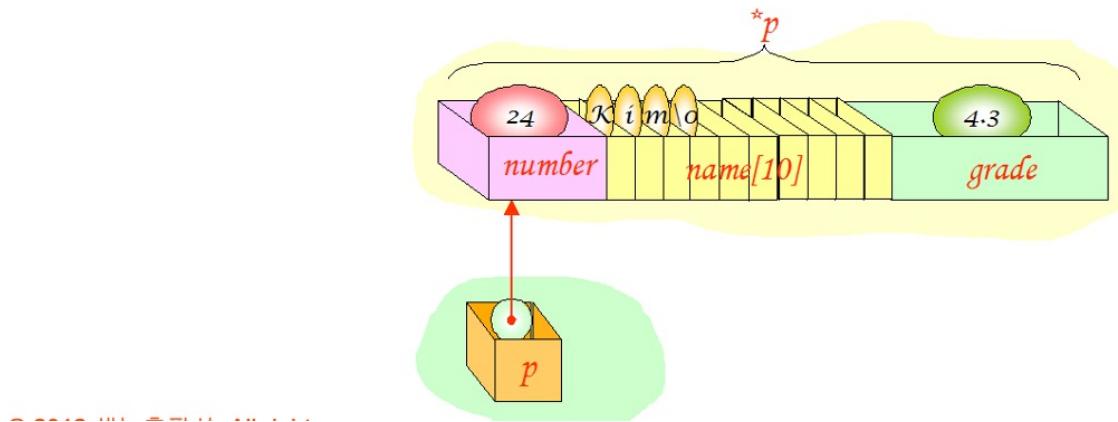
- 구조체를 가리키는 포인터
- 포인터를 멤버로 가지는 구조체

순서로 살펴봅시다.

구조체를 가리키는 포인터

- 구조체를 가리키는 포인터

```
struct student *p;  
  
struct student s = { 20070001, "홍길동", 4.3 };  
struct student *p;  
  
p = &s;  
  
printf("학번=%d 이름=%s 학점=%f \n", s.number, s.name, s.grade);  
printf("학번=%d 이름=%s 학점=%f \n", (*p).number,(*p).name,(*p).grade);
```



-> 연산자

- 연산자는 구조체 포인터로 구조체 멤버를 참조할 때 사용

```
struct student *p;
```

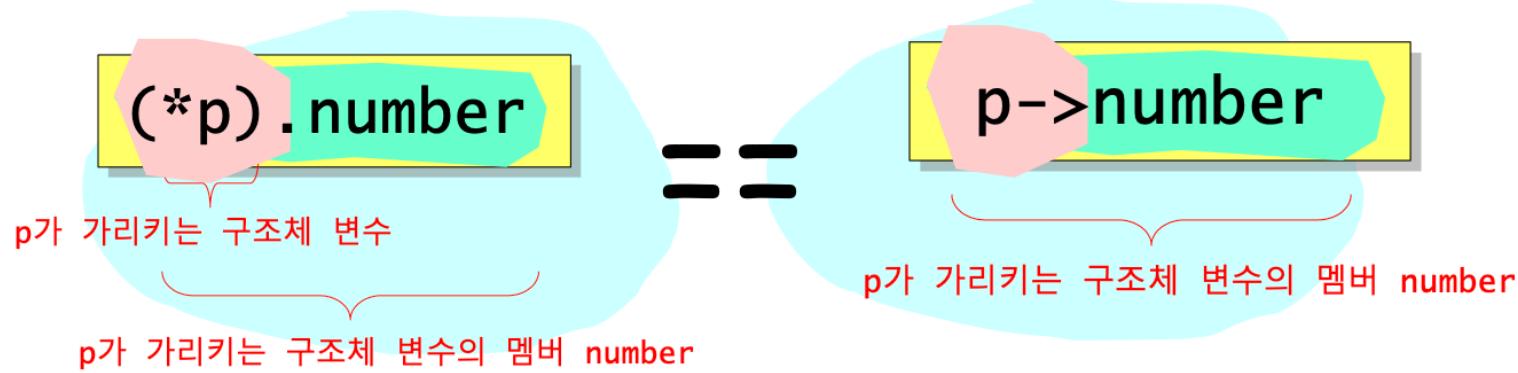
```
struct student s = { 20070001, "홍길동", 180.2 };
struct student *p;
```

```
p = &s;
```

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

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

-> 연산자 cont'd



구조체 포인터 예제

// 포인터를 통한 구조체 참조

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

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

```
int main(void)
```

```
{
```

```
    struct student s = { 20070001, "홍길동", 4.3};  
    struct student *p;
```

```
    p = &s;
```

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

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

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

```
    return 0;
```

```
}
```



```
학번=20070001 이름=홍길동 학점=4.300000
```

```
학번=20070001 이름=홍길동 학점=4.300000
```

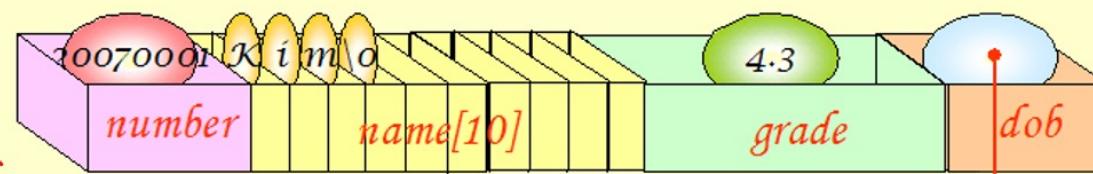
```
학번=20070001 이름=홍길동 학점=4.300000
```

포인터를 멤버로 가지는 구조체

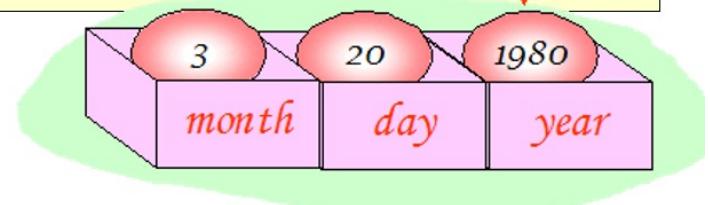
```
struct date {  
    int month;  
    int day;  
    int year;  
};
```

```
struct student {  
    int number;  
    char name[20];  
    double grade;  
    struct date *dob;  
};
```

구조체 s



구조체 d



포인터를 멤버로 가지는 구조체 cont'd

```
int main(void)
{
    struct date d = { 3, 20, 1980 };
    struct student s = { 20070001, "Kim", 4.3 };

    s.dob = &d;

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



학번: 20070001
이름: Kim
학점: 4.300000
생년월일: 1980년 3월 20일

구조체와 함수

구조체와 함수

구조체를 함수의 인수로 전달하는 경우

- 구조체의 **복사본**이 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;
}
```

구조체와 함수 cont'd

- 구조체의 포인터를 함수의 인수로 전달하는 경우
 - 시간과 공간을 절약할 수 있다.
 - 원본 훼손의 가능성이 있다.

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

포인터를 통한 구조체
의 변경을 막는다.

구조체와 함수 cont'd

- 함수에서 구조체를 반환 할 경우, 그 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.grade);

    return s;
}
```

구조체 s의 복사본
이 반환된다.

공용체

- 공용체(union)

- 같은 메모리 영역을 여러 개의 변수가 공유
- 공용체를 선언하고 사용하는 방법은 구조체와 아주 비슷

```
union example {
```

```
    char c;           // 같은 공간 공유  
    int i;           // 같은 공간 공유  
};
```



공용체 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:ȏ v.i:10000
```



공용체에 대해 타입 필드 활용하기

```
#include <stdio.h>
#include <string.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];
};
```

공용체에 대해 탑입 필드 활용하기 cont'd

```
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: %s\n", s.id.reg_number);
            printf("이름: %s\n", s.name);
            break;
        default:
            printf("타입오류\n");
            break;
    }
}
```

공용체에 대해 탑입 필드 활용하기 cont'd

```
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, "860101-1056076");
    strcpy(s2.name, "김철수");

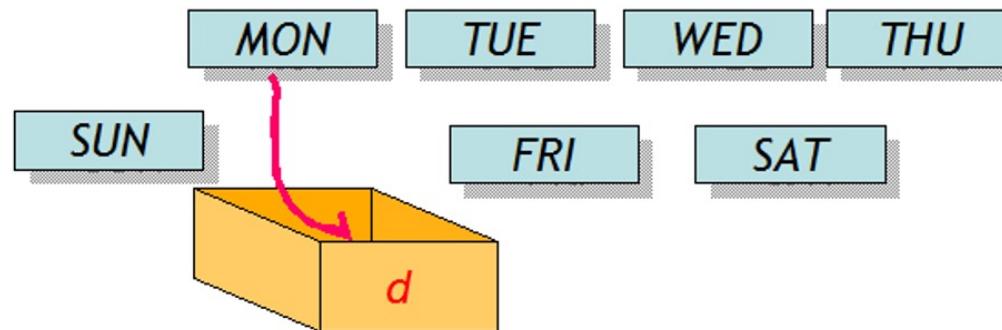
    print(s1);
    print(s2);
}
```



학번: 20070001
이름: 홍길동
주민등록번호: 860101-1056076
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.



열거형의 필요성

- 다음과 같이 프로그램을 작성할 수 있다.
 - int today;
 - today = 0; // 일요일
 - today = 1; // 월요일
- 되도록 오류를 줄이고 가독성을 높여야 된다.
- 0보다는 SUN라는 기호상수가 더 바람직하다. 의미를 쉽게 알 수 있기 때문이다.
- today에 9와 같은 의미없는 값이 대입되지 않도록 미리 차단하는 것도 필요하다.

열거형의 선언

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

태그 이름

값들을 나열

열거형 변수 선언

```
enum days today;  
today = SUN; // OK!
```

열거형 초기화

```
enum days { SUN, MON, TUE, WED, THU, FRI, SAT };      // SUN=0, MON=1, ...
enum days { SUN=1, MON, TUE, WED, THU, FRI, SAT };    // SUN=1, MON=2, ...
enum days { SUN=7, MON=1, TUE, WED, THU, FRI, SAT=6 }; // SUN=7, MON=1, ...
```



열거형 예시

```
enum colors { white, red, blue, green, black };
enum boolean { false, true };
enum levels { low, medium, high };
enum car_types { sedan, suv, sports_car, van, pickup, convertible };
```

열거형 예제

```
#include <stdio.h>

enum days { SUN, MON, TUE, WED, THU, FRI, SAT };
char *days_name[] = {
"sunday", "monday", "tuesday", "wednesday", "thursday", "friday",
"saturday" };

int main(void)
{
    enum days d;
    d = WED;
    printf("%d번째 요일은 %s입니다\n", d, days_name[d]);
    return 0;
}
```



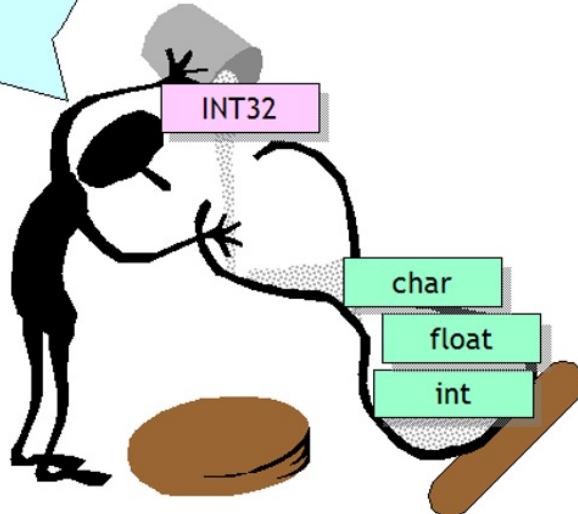
3번째 요일은 wednesday입니다

열거형과 다른 방법들 간의 비교

정수 사용	기호 상수	열거형
<pre>switch(code) { case 1: printf("LCD TV\n"); break; case 2: printf("PDP TV\n"); break; }</pre>	<pre>#define LCD 1 #define PDP 2 switch(code) { case LCD: printf("LCD TV\n"); break; case PDP: printf("PDP TV\n"); break; }</pre>	<pre>enum tvtype { LCD, PDP }; enum tvtype code; switch(code) { case LCD: printf("LCD TV\n"); break; case PDP: printf("PDP TV\n"); break; }</pre>
컴퓨터는 알기 쉬우나 사람은 기억하기 어렵다.	기호 상수를 작성할 때 오류를 저지를 수 있다.	컴파일러가 중복이 일어나지 않도록 체크한다.

typedef

typedef은 기본 자료형에 새로운 자료형을 추가하는 것입니다.



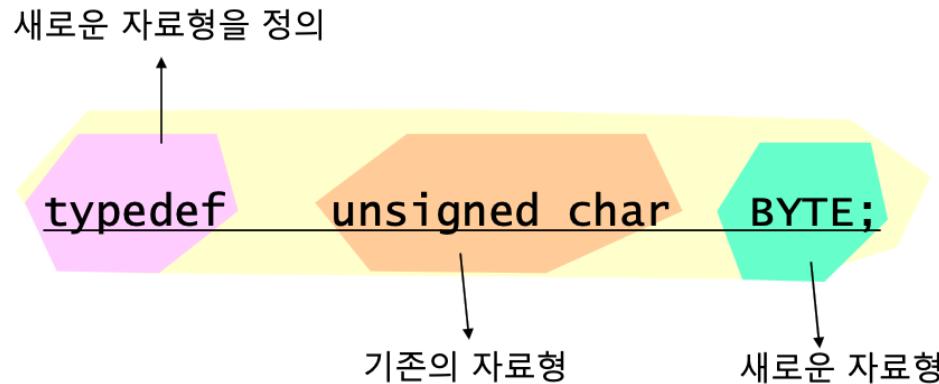
지금부터 *INT32*이라는 새로운 타입을 사용할 수 있음을 알린다.



typedef

- `typedef`은 새로운 자료형(`type`)을 정의(`define`)
- C의 기본 자료형을 확장시키는 역할

```
typedef     old_type     new_type;
```



typedef 예시 (1)

```
typedef unsigned char BYTE;
BYTE index;           // unsigned int index;와 같다.

typedef int INT32;
typedef unsigned int UINT32;

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

구조체를 새로운 타입으로 정의하기

- 구조체로 새로운 타입을 정의할 수 있다.

```
struct point {  
    int x;  
    int y;  
};  
typedef struct point POINT;  
POINT a, b;
```

typedef 예시 (2)

```
#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;
}
```

typedef 예시 (2) cont'd

```
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.

typedef 와 #define 의 비교

- 이식성을 높여준다.
 - 코드를 컴퓨터 하드웨어에 독립적으로 만들 수 있다
 - (예) int형은 2바이트이기도 하고 4바이트, int형 대신에 **typedef**을 이용한 **INT32**나 **INT16**을 사용하게 되면 확실하게 2바이트인지 4바이트인지를 지정할 수 있다.
- **#define**을 이용해도 **typedef**과 비슷한 효과를 낼 수 있다. 즉 다음과 같이 **INT32**를 정의할 수 있다.
 - **#define UINT32 unsigned int**
 - **typedef float VECTOR[2]; // #define**으로는 불가능하다.
- 문서화의 역할도 한다.
 - **typedef**을 사용하게 되면 주석을 붙이는 것과 같은 효과