IT프로그래밍

한성대학교
IT융합공학부
오희석
(ohhs@hansung.ac.kr)

Chapter 05-1. C언어가 제공하는 기본 자료형의 이해

Chapter 05. 상수와 기본 자료형

자료형은 데이터를 표현하는 방법입니다.

실수를 저장할 것이냐? 정수를 저장할 것이냐!

· 값을 저장하는 방식이 실수냐 정수냐에 따라서 달라지기 때문에 용도를 결정해야 한다.

얼마나 큰 수를 저장할 것이냐!

· 큰 수를 표현하기 위해서는 많은 수의 바이트가 필요하다.

이름 이외에 메모리 공간의 할당에 있어서 필요한 두 가지 정보

요청의 예

"아! 제가 정수를 저장할건데요. 크기는 4바이트로 하려고 합니다. 그 정도면 충분할 거에요. 그리고 변수의 이름은 num으로 할게요."

제대로 된 요청

C언어에서의 예

```
int num;
```

동일한 요청

자료형의 수는 데이터 표현방법의 수를 뜻한다. C언어가 제공하는 기본 자료형의 수가 10개라면, C언어가 제공하는 기본적인 데이터 표현방식의 수는 10개라는 뜻이 된다.

기본 자료형의 종류와 데이터의 표현범위

자료형		크기	값의 표현범위
정수형	char	1바이트	−128이상 +127이하
	short	2바이트	−32,768이상 +32,767이하
	int	4바이트	−2,147,483,648이상 +2,147,483,647이하
	long	4바이트	−2,147,483,648이상 +2,147,483,647이하
	long long	8바이트	−9,223,372,036,854,775,808이상
			+9,223,372,036,854,775,807이하
실수형	float	4바이트	$\pm 3.4\times10^{-37}$이상 $\pm 3.4\times10^{+38}$이하
	double	8바이트	$\pm 1.7\times10^{-307}$이상 $\pm 1.7\times10^{+308}$이하
	long double	8바이트 이상	double 이상의 표현범위

- 컴파일러에 따라서 약간의 차이를 보인다.
  C의 표준에서는 자료형별 상대적 크기를 표준화 할 뿐 구체적인 크기까지 언급하지는 않는다.
- 크게 정수형과 실수형으로 나뉘다.
  데이터를 표현하는 방식이 정수형과 실수형 두 가지로 나뉘므로!
- 정수형에도 실수형에도 둘 이상의 기본 자료형이 존재한다.
  표현하고자 하는 값의 크기에 따라서 적절히 선택할 수 있도록 다수의 자료형이 제공!

연산자 sizeof를 이용한 바이트 크기의 확인

```
int main(void)
{
    int num = 10;
    int sz1 = sizeof(num);
    int sz2 = sizeof(int);
    . . . .
}
```

변수 num과 int의 크기를 계산하여
그 결과로 sz1과 sz2를 초기화

sizeof 연산자의 피연산자로는 변수, 상수 및 자료형의 이름 등이 올 수 있다.
소괄호는 int와 같은 자료형의 이름에만 필수! 하지만 모든 피연산자를 대상으로 소괄호를 감싸주는 것이 일반적!

```
int main(void)
{
    char ch=9;
    int inum=1052;
    double dnum=3.1415;
    printf("변수 ch의 크기: %d \n", sizeof(ch));
    printf("변수 inum의 크기: %d \n", sizeof(inum));
    printf("변수 dnum의 크기: %d \n", sizeof(dnum));

    printf("char의 크기: %d \n", sizeof(char));
    printf("int의 크기: %d \n", sizeof(int));
    printf("long의 크기: %d \n", sizeof(long));
    printf("long long의 크기: %d \n", sizeof(long long));
    printf("float의 크기: %d \n", sizeof(float));
    printf("double의 크기: %d \n", sizeof(double));
    return 0;
}
```

실행결과

```
변수 ch의 크기: 1
변수 inum의 크기: 4
변수 dnum의 크기: 8
char의 크기: 1
int의 크기: 4
long의 크기: 4
long long의 크기: 8
float의 크기: 4
double의 크기: 8
```

정수의 표현 및 처리를 위한 일반적 자료형 선택

- **일반적인 선택은 int이다.**

  CPU가 연산하기에 가장 적합한 데이터의 크기가 int형의 크기로 결정된다.

  연산이 동반이 되면 int형으로 형 변환이 되어서 연산이 진행된다.

  따라서 연산을 동반하는 변수의 선언을 위해서는 int로 선언하는 것이 적합히다.

- **char형 short형 변수는 불필요한가?**

  연산을 수반하지 않으면서(최소한의 연산만 요구가 되면서) 많은 수의 데이터를 저장해야 한다면,

  그리고 그 데이터의 크기가 char 또는 short로 충분히 표현 가능하다면,

  char 또는 short 로 데이터를 표현 및 저장하는 것이 적절하다.

```
int main(void)
{
    char num1=1, num2=2, result1=0;
    short num3=300, num4=400, result2=0;

    printf("size of num1 & num2: %d, %d \n", sizeof(num1), sizeof(num2));
    printf("size of num3 & num4: %d, %d \n", sizeof(num3), sizeof(num4));
    printf("size of char add: %d \n", sizeof(num1+num2));
    printf("size of short add: %d \n", sizeof(num3+num4));

    result1=num1+num2;
    result2=num3+num4;
    printf("size of result1 & result2: %d, %d \n", sizeof(result1), sizeof(result2));
    return 0;
}
```

+ 연산결과의 크기가 4바이트인 이유는 피연산자가 4바이트 데이터로 형 변환 되었기 때문이다.

실행결과

```
size of num1 & num2: 1, 1
size of num3 & num4: 2, 2
size of char add: 4
size of short add: 4
size of result1 & result2: 1, 2
```

실수의 표현 및 처리를 위한 일반적 자료형 선택

- **실수 자료형의 선택기준은 정밀도**

  실수의 표현범위는 float, double 둘 다 충분히 넓다.

  그러나 8바이트 크기의 double이 float보다 더 정밀하게 실수를 표현한다.

- **일반적인 선택은 double이다.**

  컴퓨팅 환경의 발전으로 double형 데이터의 표현 및 연산이 덜 부담스럽다.

  float형 데이터의 정밀도는 부족한 경우가 많다.

실수 자료형	소수점 이하 정밀도	바이트 수
float	6자리	4
double	15자리	8
long double	18자리	12

```
int main(void)
{
    double rad;
    double area;
    printf("원의 반지름 입력: ");
    scanf("%lf", &rad);
    area = rad*rad*3.1415;
    printf("원의 넓이: %f \n", area);
    return 0;
}
```

double형 변수의 출력 서식문자 %f - printf

double형 변수의 입력 서식문자 %lf - scanf

실행결과

```
원의 반지름 입력: 2.4
원의 넓이: 18.095040
```

unsigned를 붙여서 0과 양의 정수만 표현

정수 자료형	크기	값의 표현범위
char	1바이트	−128이상 +127이하
unsigned char		0이상 (128 + 127)이하
short	2바이트	−32,768이상 +32,767이하
unsigned short		0이상 (32,768 + 32,767)이하
int	4바이트	−2,147,483,648이상 +2,147,483,647이하
unsigned int		0이상 (2,147,483,648 + 2,147,483,647)이하
long	4바이트	−2,147,483,648이상 +2,147,483,647이하
unsigned long		0이상 (2,147,483,648 + 2,147,483,647)이하
long long	8바이트	−9,223,372,036,854,775,808이상 +9,223,372,036,854,775,807이하
unsigned long long		0이상 (9,223,372,036,854,775,808 + 9,223,372,036,854,775,807)이하

- 정수 자료형의 이름 앞에는 unsigned 선언을 붙일 수 있다.
- unsigned가 붙으면, MSB도 데이터의 크기를 표현하는데 사용이 된다.
- 따라서 표현하는 값의 범위가 양의 정수로 제한이 되며 양의 정수로 두 배 늘어난다.

Chapter 05-2. 문자의 표현방식과 문자를 위한 자료형

Chapter 05. 상수와 기본 자료형

문자의 표현을 위한 약속! 아스키(ASCII) 코드!

아스키 코드	아스키 코드 값
A	65
B	66
C	67
`	96
~	126

미국 표준 협회(ANSI: American National Standards Instute)'
에 의해서 제정된
'아스키(ASCII: Amerian Standard Code for Information Interchange) 코드'

컴퓨터는 문자를 표현 및 저장하지 못한다. 따라서 문자를 표현을 목적으로 각 문자에 고유한 숫자를 지정한다.
인간이 입력하는 문자는 해당 문자의 숫자로 변환이 되어 컴퓨터에 저장 및 인식이 되고,
컴퓨터에 저장된 숫자는 문자로 변환이 되어 인간의 눈에 보여지게 된다.

```
int main(void)
{
    char ch1 = 'A';
    char ch2 = 'C';
    . . . .
}
```

```
int main(void)
{
    char ch1 = 65;
    char ch2 = 67;
    . . . .
}
```

컴파일 시 각 문자는
해당 아스키 코드 값으로 변환

따라서 실제로 컴퓨터에게 전달되는 데이터는 문자가 아닌 숫자이다.

C 프로그램상에서 문자는 작은 따옴표로 묶어서 표현

문자는 이렇게 표현되는 거구나!

```
int main(void)
{
    char ch1='A', ch2=65;
    int ch3='Z', ch4=90;

    printf("%c %d \n", ch1, ch1);
    printf("%c %d \n", ch2, ch2);
    printf("%c %d \n", ch3, ch3);
    printf("%c %d \n", ch4, ch4);
    return 0;
}
```

서식문자 %c 해당 숫자의 아스키 코드 문자를 출력해라!

```
A 65
A 65
Z 90
Z 90
```

실행결과

- 문자를 char형 변수에 저장하는 이유

  모든 아스키 코드 문자는 1바이트로도 충분히 표현가능

  문자는 덧셈, 뺄셈과 같은 연산을 동반하지 않는다. 단지 표현에 사용될 뿐이다.

  따라서 1바이트 크기인 char형 변수가 문자를 저장하기 최적의 장소이다.

  문자는 int형 변수에도 저장이 가능하다.

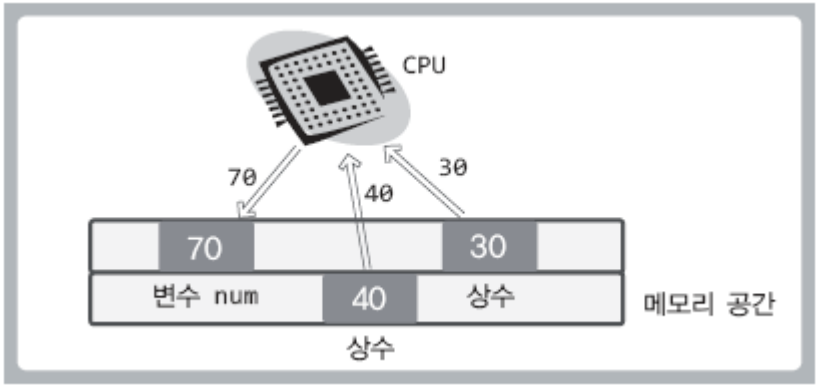
Chapter 05-3. 상수에 대한 이해

Chapter 05. 상수와 기본 자료형

이름을 지니지 않는 리터럴 상수!

```
int main(void)
{
    int num = 30 + 40;
    . . . .
}
```

연산을 위해서는 30, 40과 같이 프로그램상에 표현되는 숫자도
메모리 공간에 저장되어야 한다.
이렇게 저장되는 값은 이름이 존재하지 않으니 변경이 불가능한 상수이다.
따라서 **리터럴 상수**라 한다.



- 단계 1. 정수 30과 40이 메모리 공간에 상수의 형태로 저장된다.
- 단계 2. 두 상수를 기반으로 덧셈이 진행된다.
- 단계 3. 덧셈의 결과로 얻어진 정수 70이 변수 num에 저장된다.

메모리 공간에 저장이 되어야 CPU의 연산대상이 된다.

리터럴 상수의 자료형

```
int main(void)
{
    printf("literal int size: %d \n", sizeof(7));
    printf("literal double size: %d \n", sizeof(7.14));
    printf("literal char size: %d \n", sizeof('A'));
    return 0;
}
```

실행결과

```
literal int size: 4
literal double size: 8
literal char size: 4
```

리터럴 상수도 자료형이 결정되어야 메모리 공간에 저장이 될 수 있다.
위 예제의 실행결과는 다음 사실을 의미한다.

- **정수는 기본적으로 int형으로 표현된다.**
- **실수는 기본적으로 double형으로 표현된다.**
- **문자는 기본적으로 int형으로 표현된다.**

접미사를 이용한 다양한 상수의 표현

```
int main(void)
{
    float num1 = 5.789;    // 경고 메시지 발생
    float num2 = 3.24 + 5.12;    //경고 메시지 발생
    return 0;
}
```

실수는 double형 상수로 인식이 되어
데이터 손실에 대한 경고 메시지 발생

```
float num1 = 5.789f;        // 경고 메시지 발생 안 함
float num2 = 3.24F + 5.12F;    // 소문자 f 대신 대문자 F를 써도 된다!
```

접미사를 통해서 상수의 자료형을 변경할 수 있다.

접미사	자료형	사용의 예
U	unsigned int	unsigned int n = 1025U
L	long	long n = 2467L
UL	unsigned long	unsigned long n = 3456UL
LL	long long	long long n = 5768LL
ULL	unsigned long long	unsigned long long n = 8979ULL

[표 05-4: 정수형 상수의 표현을 위한 접미사]

접미사	자료형	사용의 예
F	float	float f = 3.15F
L	long double	long double f = 5.789L

[표 05-5: 실수형 상수의 표현을 위한 접미사]

이름을 지니는 심볼릭(Symbolic) 상수: const 상수

```
int main(void)
{
    const int MAX=100;    // MAX는 상수! 따라서 값의 변경 불가!
    const double PI=3.1415;    // PI는 상수! 따라서 값의 변경 불가!
    . . . .
}
```

```
int main(void)
{
    const int MAX;    // 쓰레기 값으로 초기화 되어버림
    MAX=100;    // 값의 변경 불가! 따라서 컴파일 에러 발생!
    . . . .
}
```

상수의 이름은

모두 대문자로 표시하고,

둘 이상의 단어를 연결할 때에는 MY_AGE와 같이 언더바를 이용해서 두 단어를 구분하는 것이 관례!

Chapter 05-4. 자료형의 변환

Chapter 05. 상수와 기본 자료형

대입 연산의 과정에서 발생하는 자동 형 변환

```
double num1=245;     // int형 정수 245를 double형으로 자동 형 변환
int num2=3.1415;     // double형 실수 3.1415를 int형으로 자동 형 변환
```

대입 연산자의 왼편을 기준으로 형 변환이 발생한다.

정수 245는 245.0의 비트 열로 재구성이 되어 변수 num1에 저장된다.

실수 3.1415는 int형 데이터 3의 비트 열로 재구성이 되어 변수 num2에 저장된다.

```
int num3=129;
char ch=num3;     // int형 변수 num3에 저장된 값이 char형으로 자동 형 변환
```

4바이트 변수 num3에 저장된 4바이트 데이터 중 상위 3바이트가 손실되어 변수 ch에 저장된다.

00000000 00000000 00000000 10000001 ➡ 10000001

자동 형 변환의 방식 정리

형 변환의 방식에 대한 유형별 정리

- 정수를 실수로 형 변환 — 3은 3.0으로 5는 5.0으로(오차가 발생하게 된다).
- 실수를 정수로 형 변환 — 소수점 이하의 값이 소멸된다.
- 큰 정수를 작은 정수로 형 변환 — 작은 정수의 크기에 맞춰서 상위 바이트가 소멸된다.

```
int main(void)
{
    double num1=245;
    int num2=3.1415;
    int num3=129;
    char ch=num3;

    printf("정수 245를 실수로: %f \n", num1);
    printf("실수 3.1415를 정수로: %d \n", num2);
    printf("큰 정수 129를 작은 정수로: %d \n", ch);
    return 0;
}
```

실행결과

```
정수 245를 실수로: 245.000000
실수 3.1415를 정수로: 3
큰 정수 129를 작은 정수로: -127
```

정수의 승격에 의한 자동 형 변환

일반적으로 CPU가 처리하기에 가장 적합한 크기의 정수 자료형을 int로 정의한다.
따라서 int형 연산의 속도가 다른 자료형의 연산속도에 비해서 동일하거나 더 빠르다.

따라서 다음과 같은 방식의
형 변환 발생

```
int main(void)
{
     short num1=15, num2=25;
     short num3=num1+num2;     // num1과 num2가 int형으로 형 변환
     . . . .
}
```

이를 가리켜 '정수의 승격(Integral Promotion)'이라 한다.

피연산자의 자료형 불일치로 발생하는 자동 형 변환

```
double num1 = 5.15 + 19;
```

두 피연산자의 자료형은 일치해야 한다. 일치하지 않으면 일치하기 위해서 자동으로 형 변환이 발생한다.

아래의 자동 형 변환 규칙을 근거로 int형 데이터 19가 double형 데이터 19.0으로 형 변환이 되어 덧셈이 진행된다.

산술연산에서의 자동 형 변환 규칙

바이트 크기가 큰 자료형이 우선시 된다.
정수형보다 실수형을 우선시 한다.
이는 데이터의 손실을 최소화 하기 위한 기준이다.

명시적 형 변환: 강제로 일으키는 형 변환

```
int main(void)
{
    int num1=3, num2=4;
    double divResult;
    divResult = num1 / num2;
    printf("나눗셈 결과: %f \n", divResult);
    return 0;
}
```

num1과 num2가 정수이기 때문에 몫만 반환이 되는 정수형 나눗셈이 진행

실행결과

```
나눗셈 결과: 0.000000
```

```
divResult = (double)num1 / num2;
```

(type)은 type형으로의 형 변환을 의미한다.

num1이 double형으로 명시적 형 변환 그리고 num1과 num2의 / 연산 과정에서의 산술적 자동 형 변환!
그 결과 실수형 나눗셈이 진행되어 divResult에는 0.75가 저장된다.

```
int main(void)
{
    int num1 = 3;
    double num2 = 2.5 * num1;
    . . . .
}
```

```
int main(void)
{
    int num1 = 3;
    double num2 = 2.5 * (double)num1;
    . . . .
}
```

추천하는 코드 작성 스타일

자동 형 변환이 발생하는 위치에 명시적 형 변환 표시를 해서 형 변환이 발생함을 알리는 것이 좋다!

Chapter 06-1. printf 함수 이야기

Chapter 06. printf함수

printf 함수와 특수문자

```
int main(void)
{
    printf("I like programming \n");
    printf("I love puppy! \n");
    printf("I am so happy \n");
    return 0;
}
```

실행결과

```
I like programming
I love puppy!
I am so happy
```

printf 함수는 첫 번째 인자로 전달된 문자열을 출력한다.

printf("앞집 강아지가 말했다. "멍~! 멍~!" 정말 귀엽다.");

잘못된 printf 함수 호출문

"앞집 강아지가 말했다. " → 음 이것은 하나의 문자열이군!

멍~! 멍~! → 이건 뭐지?

" 정말 귀엽다." → 이것도 하나의 문자열이고!

큰 따옴표는 문자열의 시작과 끝으로 해석이 되니, 큰 따옴표 자체의 출력을 원하는 경우에는 큰 따옴표 앞에 \ 문자를 붙여주기로 하자!

특수문자의 탄생 배경

컴파일러의 오해?

printf("앞집 강아지가 말했다. \"멍~! 멍~!\" 정말 귀엽다.");

제대로 된 printf 함수 호출문

특수문자의 종류

특수문자	의미하는 바
\a	경고음
\b	백스페이스(backspace)
\f	폼 피드(form feed)
\n	개 행(new line)
\r	캐리지 리턴(carriage return)
\t	수평 탭
\v	수직 탭
\'	작은 따옴표 출력
\"	큰 따옴표 출력
\?	물음표 출력
\\	역슬래쉬 출력

\f와 \v는 모니터 출력이 아닌 프린터 출력을 위해 정의된 특수문자이기 때문에 모니터의 출력에 사용하면, 이상한 문자 출력!

printf 함수의 서식지정과 서식문자들

```
int main(void)
{
    int myAge=12;
    printf("제 나이는 10진수로 %d살, 16진수로 %X살입니다. \n", myAge, myAge);
    return 0;
}
```

서식문자를 이용해서 출력할 문자열의 형태를 조합해 낼 수 있다.
즉, 출력의 서식을 지정할 수 있다.

```
제 나이는 10진수로 12살, 16진수로 C살입니다.
```

실행결과

```
int main(void)
{
    int num1=7, num2=13;
    printf("%o %#o \n", num1, num1);
    printf("%x %#x \n", num2, num2);
    return 0;
}
```

```
7 07
d 0xd
```

실행결과

#을 삽입하면 8진수 앞에 0, 16진수 앞에 0x가 삽입된다.

서식문자	출력 대상(자료형)	출력 형태
%d	char, short, int	부호 있는 10진수 정수
%ld	long	부호 있는 10진수 정수
%lld	long long	부호 있는 10진수 정수
%u	unsigned int	부호 없는 10진수 정수
%o	unsigned int	부호 없는 8진수 정수
%x, %X	unsigned int	부호 없는 16진수 정수
%f	float, double	10진수 방식의 부동소수점 실수
%Lf	long double	10진수 방식의 부동소수점 실수
%e, %E	float, double	e 또는 E 방식의 부동소수점 실수
%g, %G	float, double	값에 따라 %f와 %e 사이에서 선택
%c	char, short, int	값에 대응하는 문자
%s	char *	문자열
%p	void *	포인터의 주소 값

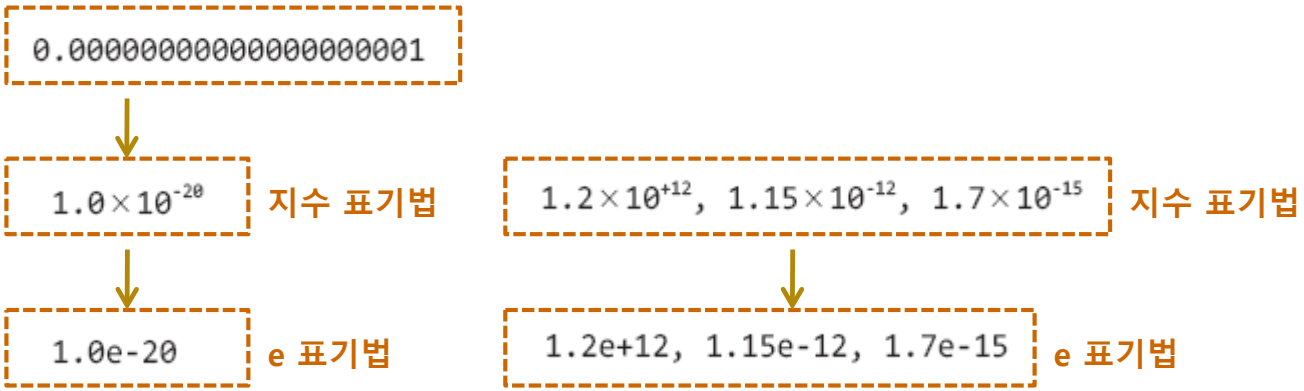
실수의 출력을 위한 서식문자들: %f, %e

```
int main(void)
{
    printf("%f \n", 0.1234);
    printf("%e \n", 0.1234);    // e 표기법 기반의 출력
    printf("%f \n", 0.12345678);
    printf("%e \n", 0.12345678);    // e 표기법 기반의 출력
    return 0;
}
```

실행결과

```
0.123400
1.234000e-001
0.123457
1.234568e-001
```

컴퓨터는 지수를 표현할 수 없으므로
e 표기법으로 지수를 대신 표현한다.



%g의 실수출력과 %s의 문자열 출력

```
int main(void)
{
    double d1=1.23e-3;    // 0.00123
    double d2=1.23e-4;    // 0.000123
    double d3=1.23e-5;    // 0.0000123
    double d4=1.23e-6;    // 0.00000123

    printf("%g \n", d1);    // %f 스타일 출력
    printf("%g \n", d2);    // %f 스타일 출력
    printf("%g \n", d3);    // %e 스타일 출력
    printf("%g \n", d4);    // %e 스타일 출력
    return 0;
}
```

%g는 실수의 형태에 따라서 %f와 %e 사이에서 적절한 형태의 출력을 진행한다.

%g와 %G의 차이점은 e 표기법의 e를 소문자로 출력하느냐 대문자로 출력하느냐에 있다.

실행결과

```
0.00123
0.000123
1.23e-005
1.23e-006
```

```
int main(void)
{
    printf("%s, %s, %s \n", "AAA", "BBB", "CCC");
    return 0;
}
```

실행결과

```
AAA, BBB, CCC
```

%s의 문자열 출력과 관련해서는 배열과 포인터 공부 후에 완벽히 이해하자!

일단은 %s의 사용법을 예제 기반으로 이해하자.

필드 폭을 지정하여 정돈된 출력 보이기

%8d
필드 폭을 8칸 확보하고, 오른쪽 정렬해서 출력을 진행한다.

%-8d
필드 폭을 8칸 확보하고, 왼쪽 정렬해서 출력을 진행한다.

```
int main(void)
{
    printf("%-8s %14s %5s \n", "이  름", "전공학과", "학년");
    printf("%-8s %14s %5d \n", "김동수", "전자공학", 3);
    printf("%-8s %14s %5d \n", "이을수", "컴퓨터공학", 2);
    printf("%-8s %14s %5d \n", "한선영", "미술교육학", 4);
    return 0;
}
```

실행결과

```
이  름          전공학과    학년
김동수          전자공학     3
이을수        컴퓨터공학     2
한선영        미술교육학     4
```

서식문자 사이에 들어가는 숫자는 필드의 폭을 의미한다.

기본 오른쪽 정렬이다. 따라서 –는 왼쪽 정렬을 의미하는 용도로 사용된다.