Types, Operations and Expressions

백 윤 철

Contents

- Variable Names
- Data Types and Size
- Constants
- Declarations
- Arithmetic Operators
- Relational and Logical Operators
- Type Conversions
- Increment and Decrement Operators
- Bitwise Operators
- Assignment Operators and Expressions
- Conditional Expressions
- Precedence and Order of Evaluation

Variable Names

- Names are made up of letters and digit
  - First character must be letter. 알파벳
  - Don’t begin variable names with underscore ‘_’.
  - Upper case and lower case letters are distinct. 다름
  - The first 31 characters of internal name are significant.
  - The standard guarantees uniqueness only 6 characters and a single case for the external names.
  - You can’t use keywords as variable names.

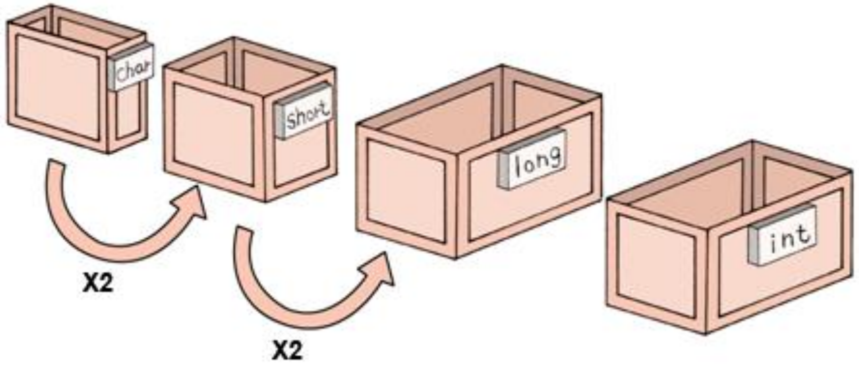
    `(ex) if, else, int, float`

Data types and Size

- Basic data types in C
  - char - a single byte, capable of holding one character in the local character set (1 byte)
  - int - an integer, typically inflecting the natural size of integer on the host machine (4bytes)
  - float - single-precision floating point (4 bytes)
  - double - double-precision floating point (8 bytes)
- Qualifier
  - Integer
    - short : 2 bytes
    - long : 8 bytes
  - character or integer
    - signed
    - unsigned

    (ex) char (-128~127)

    unsigned char (0~255)



Signed and unsigned char

1	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Unsigned char 0 ~ 255

$1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0$

Signed char -128 ~ 127

11111111 -> 2's complement -> 00000000 + 1

-> 00000001

-128

Constants

- integer constant `ex)1234`
- long constant `ex)123456789L or l`
- unsigned long constant `ex)123456789UL or ul`
- floating-point constant `ex)123.4 , 1e-2`
- character constant `ex)‘A’-> numeric value is 65 in the ASCII character set`
- string constant `ex)“I am a string” “”`
- enumeration constant `ex) enum bool { no, yes };`

Constants

- Value of integer
  - octal : leading 0
  - hexadecimal : leading 0X

  `ex) decimal 31 can be written 037 in octal and 0X1F in hex.`

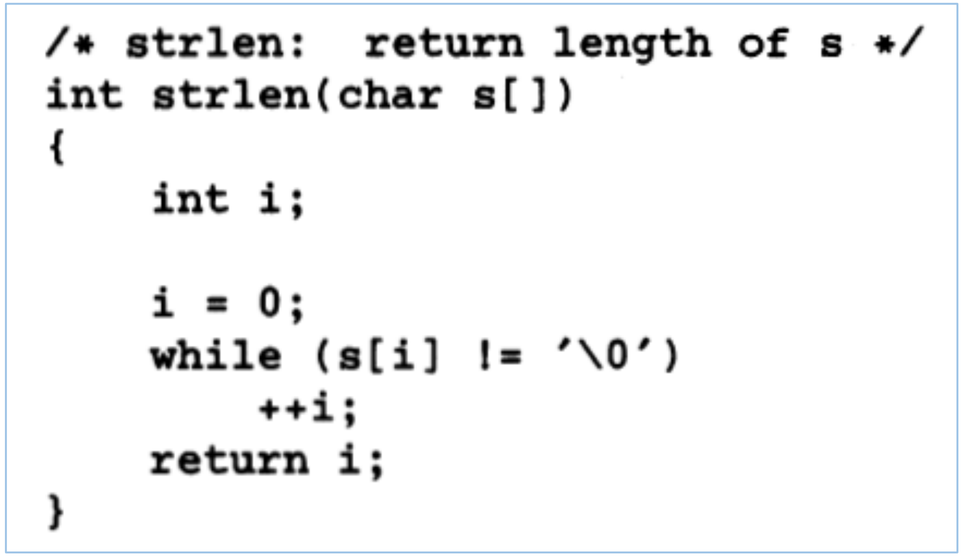
- String Constant 정수
  - Quotes are not part of string. `ex) “I am a string”`

    " "   연결의
  - String constants can be concatenated at compile time.

    `ex)“hello,” “world” is equivalent to “hello, world”.`
  - A string constant is an array of characters.
  - The internal representation of a string has a null character `’\0’` at the end.

Constants

- The standard library function `strlen(s)` returns the length of its character string argument s, excluding the ending `‘\0’`

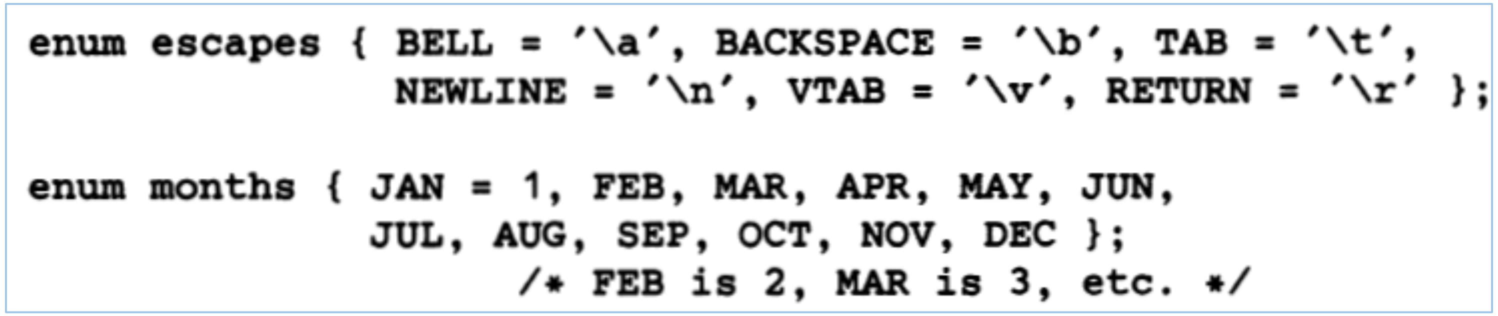


```
/* strlen:  return length of s */
int strlen(char s[])
{
    int i;

    i = 0;
    while (s[i] != ’\0’)
        ++i;
    return i;
}
```

Constants

열거

- Enumeration Constant
  - a list of constant integer values
  - The first name in an enum has value 0, the next 1, and so on, unless explicit values are specified



```
enum escapes { BELL = '\a', BACKSPACE = '\b', TAB = '\t',
               NEWLINE = '\n', VTAB = '\v', RETURN = '\r' };

enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,
              JUL, AUG, SEP, OCT, NOV, DEC };
                    /* FEB is 2, MAR is 3, etc. */
```

Constants

- All variables must be declared before use.
- A declaration specifies a type, and contains a list of one or more variables of that type.

```
int lower, upper, step;
int c, line[1000];
```

- A variable may be initialized in its declaration.

```
int i = 0;
int limit = MAXLINE + 1;
```

- External and static variables are initialized to zero by default.

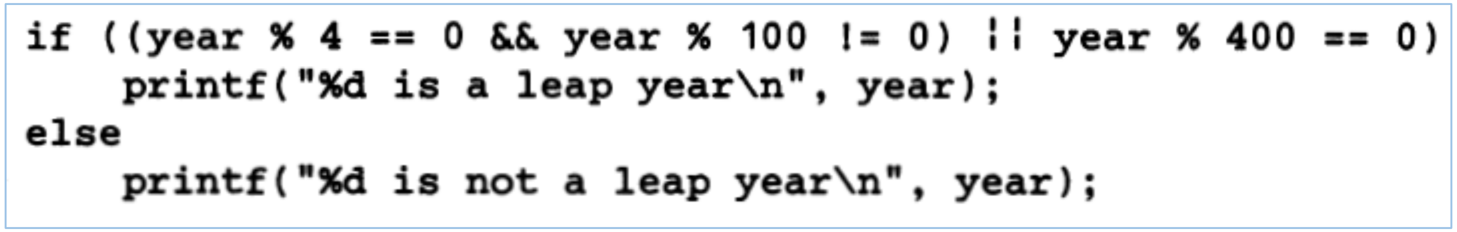
변경 X 해주는 역할

- The qualifier `const` can be applied to the declaration of any variable to specify that the elements will not be altered.

```
const double e = 2.71828182845905;
const char msg[] = “warning: ”;
```

Arithmetic Operators

- binary arithmetic operators +, -, *, /
- modulus operator %
- Integer division truncates any fractional part.
    - calculating leap year



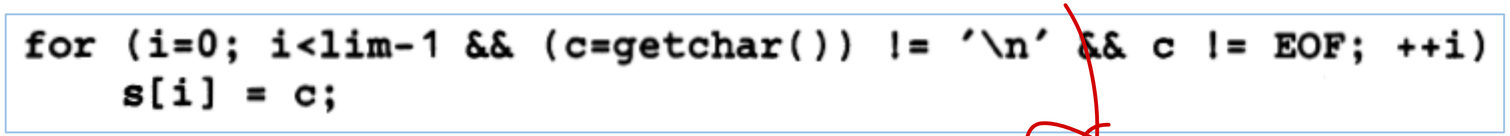
```
if ((year % 4 == 0 && year % 100 != 0) ¦¦ year % 400 == 0)
    printf("%d is a leap year\n", year);
else
    printf("%d is not a leap year\n", year);
```

- The % operator cannot be applied to float or double.

소수가 나오면 X

Relational and Logical Operators

- relational operators > >= < <= They all have same precedence.
- equality operators == != They have lower precedence than relational operators.
- Relational operators have lower precedence than arithmetic operators.
- && or || are evaluated left to right, and evaluation stops as soon as the truth or falsehood of the result is known.



```
for (i=0; i<lim-1 && (c=getchar()) != '\n' && c != EOF; ++i)
    s[i] = c;
```

- “if not valid” `if(!valid)` rather than `if(valid == 0)`

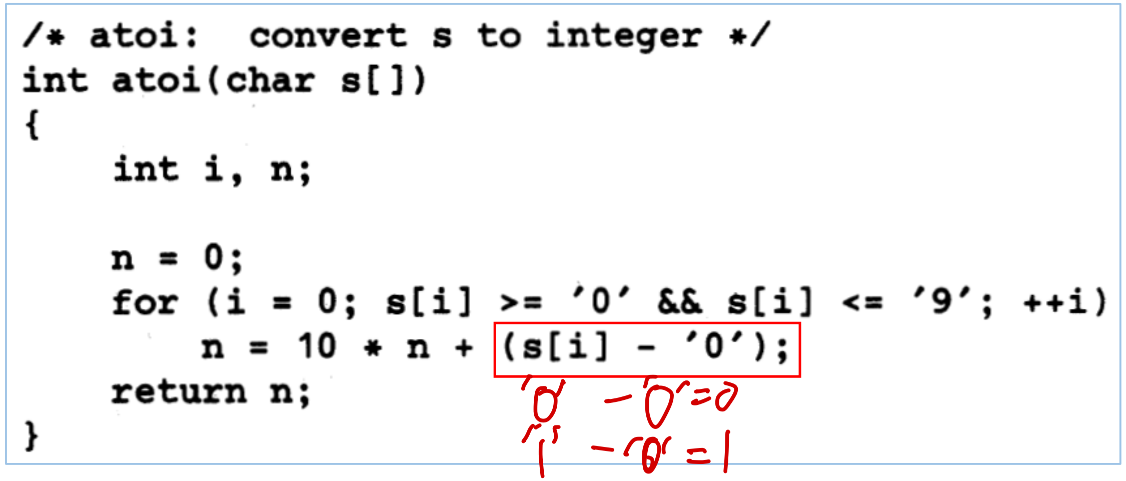
Type Conversions

피연산자

- When an operator has operands of different types, they are converted to a common type
- The only automatic conversions are those that convert a “narrower” operand into a “wider” one without losing information.

작 → 큰

- Function `atoi()`

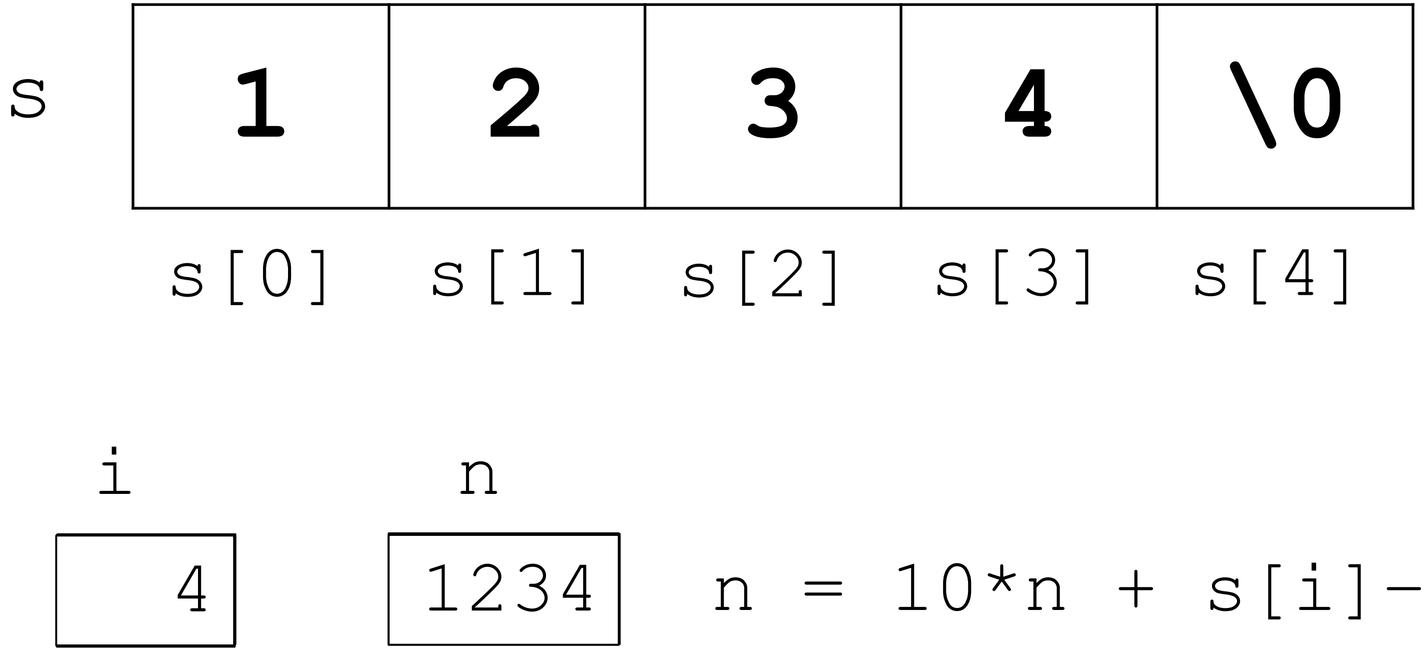


```
/* atoi:  convert s to integer */
int atoi(char s[])
{
    int i, n;

    n = 0;
    for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
        n = 10 * n + (s[i] - '0');
    return n;
}
```

'0' − '0' = 0

'1' − '0' = 1

atoi() detail



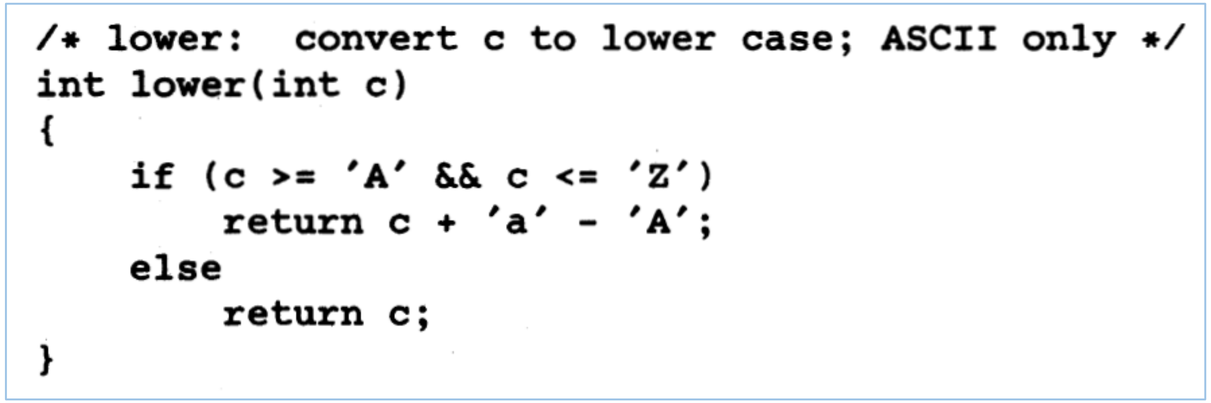
s	1	2	3	4	\0
	s[0]	s[1]	s[2]	s[3]	s[4]

i	n
4	1234

```
n = 10*n + s[i]-‘0’;
```

Type Conversions

- Function `lower()`



```
/* lower:  convert c to lower case; ASCII only */
int lower(int c)
{
    if (c >= ’A’ && c <= ’Z’)
        return c + ’a’ - ’A’;
    else
        return c;
}
```

- Simple conversion rules

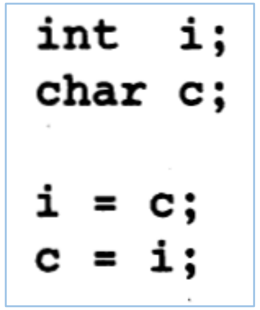
  If either operand is long double, convert the other to long double.
  Otherwise, if either operand is double, convert the other to double.
  Otherwise, if either operand is float, convert the other to float.
  Otherwise, convert char and short to int.
  Then, if either operand is long, convert the other to long.

Type Conversions

- Conversions take place across assignments

큰 → 작 정보X

작 → 큰 정보O



```
int  i;
char c;

i = c;
c = i;
```

- c is unchanged, but reverse order of assignment may lose information.
- consider followings

정보유지 X

```
float x;
int i;

x = i;
i = x;
```

정보유지 O

```
float x;
double d;

x = d;
d = x;
```

Type Conversions

- Cast : explicit type conversion

명시적

변환

```
int n;

sqrt((double) n)
```

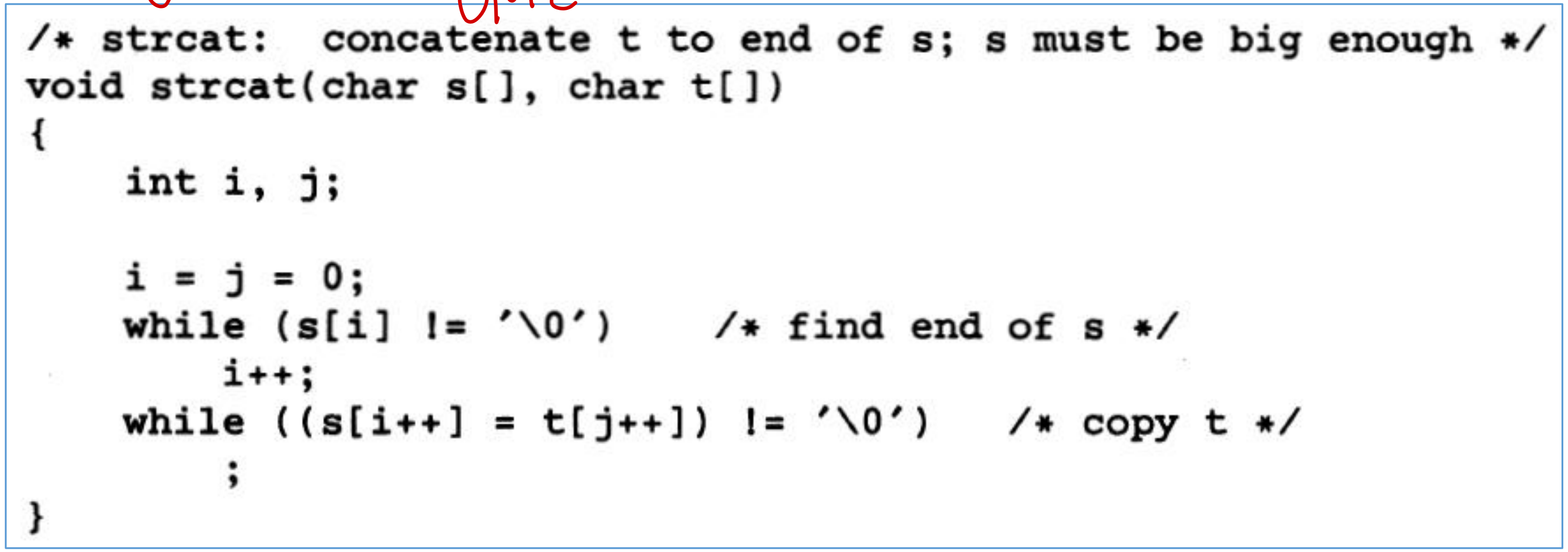
  - convert the value of n to double before passing it to sqrt()

Increment and Decrement Operators

- ++ increment, -- decrement
- `++n` increment before use, `n++` increment after use
- If n is 5, `x = n++;` set x to 5, `x = ++n;` set x to 6

문자열 연결함수

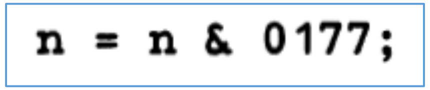
이어진



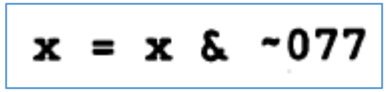
```
/* strcat:  concatenate t to end of s; s must be big enough */
void strcat(char s[], char t[])
{
    int i, j;

    i = j = 0;
    while (s[i] != '\0')     /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0')    /* copy t */
        ;
}
```

Bitwise Operators

- Six operators on integer operand
  - & bitwise AND
  - | bitwise inclusive OR
  - ^ bitwise exclusive OR
  - << left shift
  - >> right shift
  - ~ one's complement
- & is often used to mask off set of bits



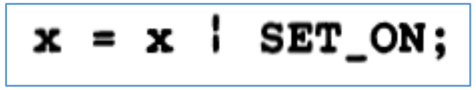
```
n = n & 0177;
```



```
x = x & ~077
```

x	0	1	0	0	0	0	0	0
~077	1	1	0	0	0	0	0	0

- | is used to turn bits on



```
x = x | SET_ON;
```

Logical and bitwise operator

L

A	B	A&&B	A||B	!A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

```
int i = 7; //00000111
int j = 8; //00001000

!i T -> F == 0
i && j T&&T -> T == 1
i || j T||T -> T == 1
```

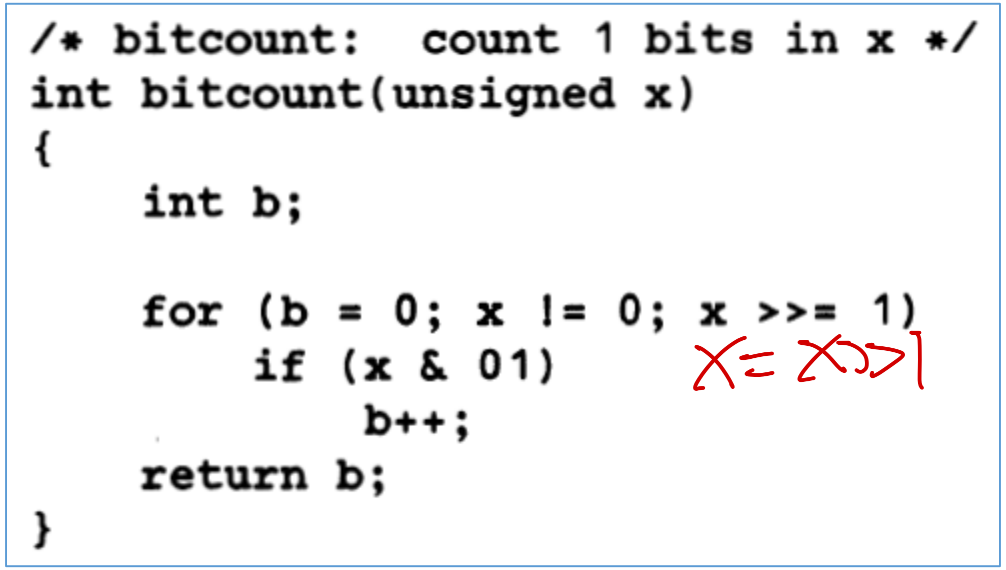
b

a	b	a&b	a|b	a^b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

```
i & j 00000000 -> 0
i | j 00001111 -> 15

i<<2 00011100 -> 28
j>>2 00000010 -> 2
```

Assignment Operators and Expressions

- `i += 2` is compressed form of `i = i + 2`
- `+=` is assignment operator
- most binary operators `( + - * / % << >> & ^ | )` have corresponding assignment operator



```
/* bitcount:  count 1 bits in x */
int bitcount(unsigned x)
{
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

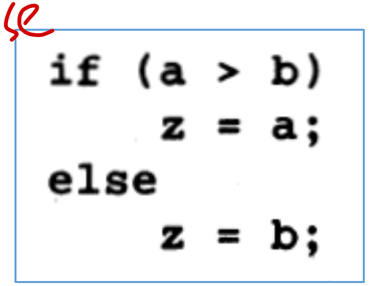
X= X>>1

x	0	0	0	0	0	0	0	0
01	0	0	0	0	0	0	0	1

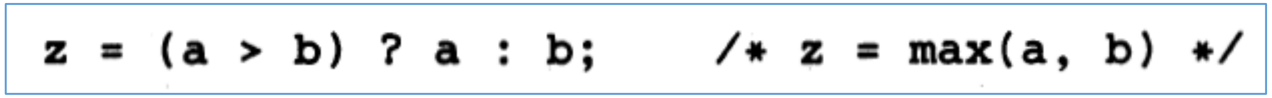
Conditional Expressions

- `expr1? expr2: expr3`
  - `expr1` is evaluated first.
  - if it is true, then the `expr2` is evaluated, and that is the value of the conditional expression.
  - Otherwise `expr3` is evaluated, and that is the value.

1= false



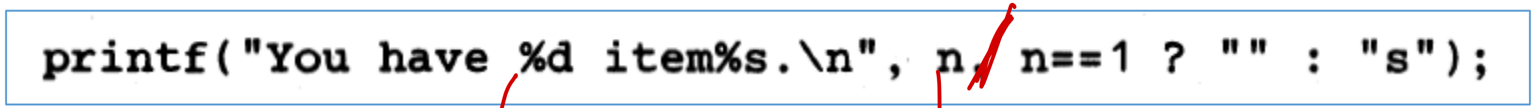
```
if (a > b)
    z = a;
else
    z = b;
```



```
z = (a > b) ? a : b;     /* z = max(a, b) */
```

간단명료

- it often leads to succinct code



```
printf("You have %d item%s.\n", n, n==1 ? "" : "s");
```

Precedence and Order of Evaluation

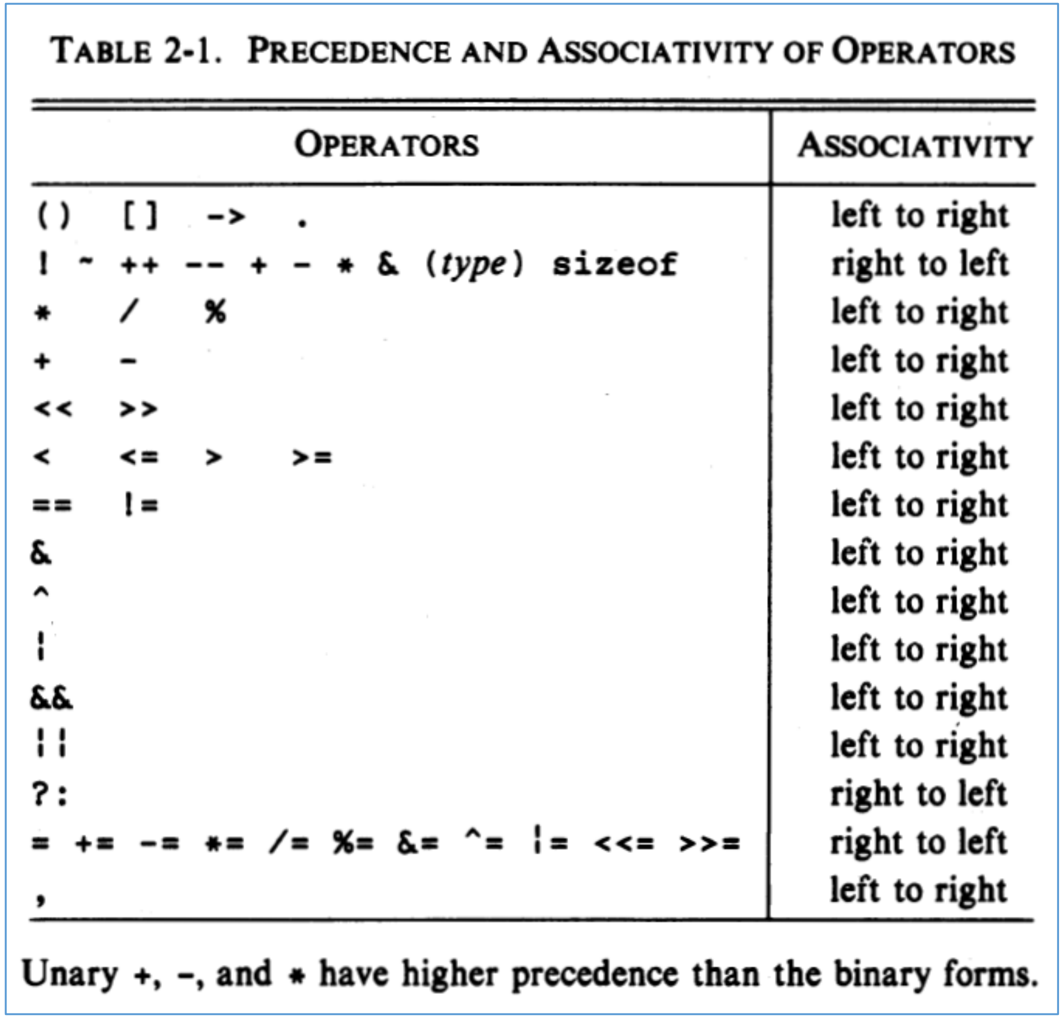


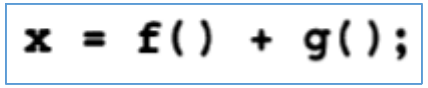
TABLE 2-1. PRECEDENCE AND ASSOCIATIVITY OF OPERATORS

OPERATORS	ASSOCIATIVITY
`()  []  ->  .`	left to right
`! ~ ++ -- + - * &` (*type*) `sizeof`	right to left
`*  /  %`	left to right
`+  -`	left to right
`<<  >>`	left to right
`<  <=  >  >=`	left to right
`==  !=`	left to right
`&`	left to right
`^`	left to right
`¦`	left to right
`&&`	left to right
`¦¦`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= ¦= <<= >>=`	right to left
`,`	left to right

Unary +, -, and * have higher precedence than the binary forms.

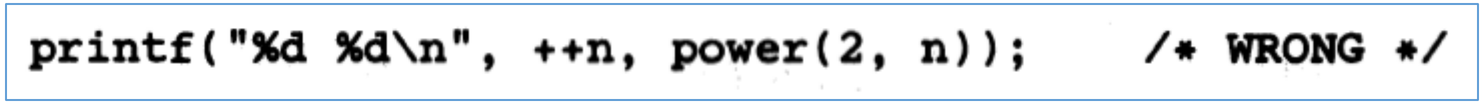
Precedence and Order of Evaluation

- C, like most languages, does not specify the order in which the operands of an operator are evaluated.
- f may be evaluated before g or vice versa.

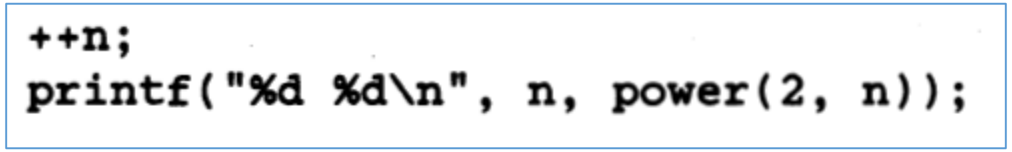


```
x = f() + g();
```

- same on function arguments evaluation



```
printf("%d %d\n", ++n, power(2, n));     /* WRONG */
```



```
++n;
printf("%d %d\n", n, power(2, n));
```

정리

- Variable Names, Data Types and Size
- Operators
- Type Conversions
- Increment and Decrement Operators
- Bitwise Operators
- Assignment Operators and Expressions
- Conditional Expressions
- Precedence and Order of Evaluation