**3 Data와 C**

□○•▶★

□Keywords: int , short , long , unsigned , char , float , double , Bool , Complex , Imaginary

□ Operator: sizeof

□ Function: scanf()

□ C의 기본적인 data type을 실습하기

□ integer types과 floating-point types의 차이점을 실습하기

□ constants와 constant type의 변수를 선언하는 방법을 실습

□ 다른 data type의 변수를 read, write하기 위한 printf(), scanf() function의 사용실습

**3.1 견본 Program**

Listing 3.1 The platinum.c Program

/\* platinum.c -- your weight in platinum \*/

#include <stdio.h>

int main(void)

{

float weight; /\* user weight \*/

float value; /\* platinum equivalent \*/

printf("Are you worth your weight in platinum?\n");

printf("Let's check it out.\n");

printf("Please enter your weight in pounds: ");

/\* get input from the user \*/

scanf("%f", &weight);

/\* assume platinum is $1700 per ounce \*/

/\* 14.5833 converts pounds avd. to ounces troy \*/

value = 1700.0 \* weight \* 14.5833;

printf("Your weight in platinum is worth $%.2f.\n", value);

printf("You are easily worth that! If platinum prices drop,\n");

printf("eat more to maintain your value.\n");

return 0;

}

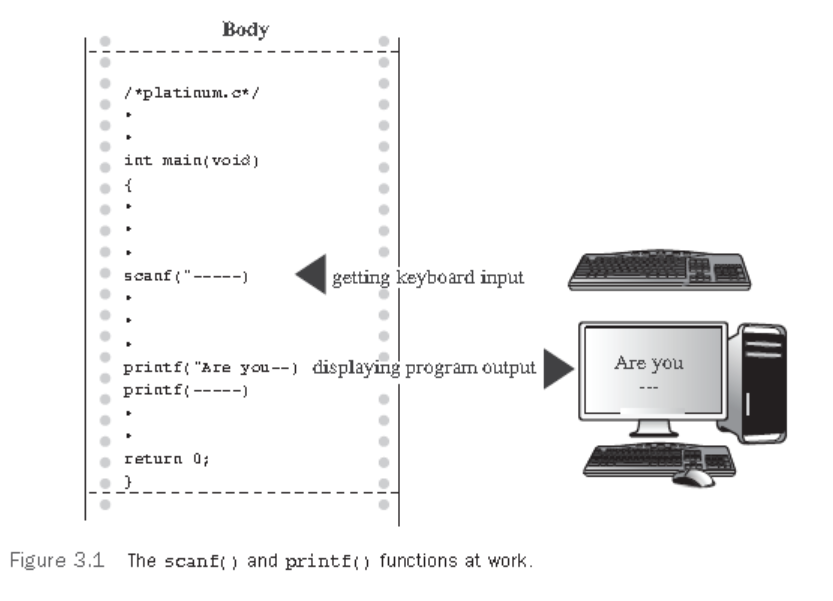
**3.1.1 Program이 하는 일은?**

□ a floating-point variable type ( float )

* decimal points를 사용하는 숫자
* floating-point value를 출력하기 위하여 printf()에서 %f specifier를 사용
  + %.2f: decimal point의 우측에 2자리 숫자를 출력

□ scanf() function는 keyboard 입력을 받는 함수

○ %f specifier는 scanf()가 keyboard로 부터 floating-point number를 입력받게 함



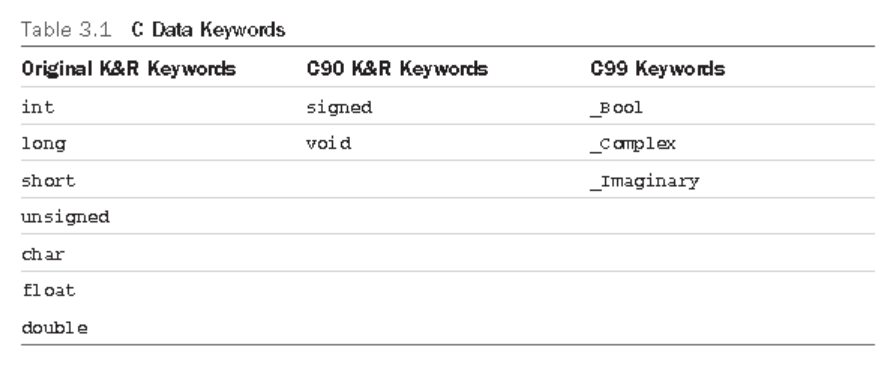
**3.2 Data Variables와 Constants**

□ program은 *data*로 작업하는 것이다

□ variable 과 constant 의 차이: variable은 program 실행 도중에 assign된 값을 갖거나 변경될 수 있다. constant는 변경이 안되는 것

**3.2.1 Data: Data-Type Keywords**

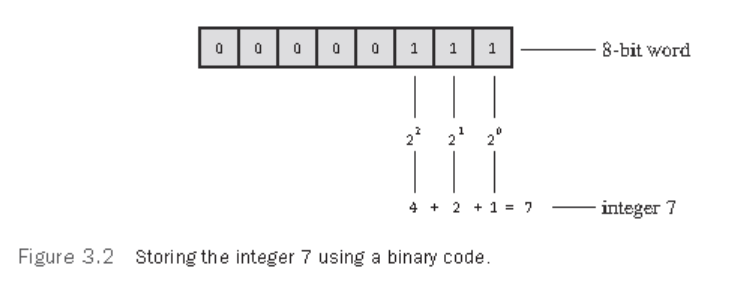
□ variable : declaration statement에서 명시된 type을 가져야 한다.



**3.2.2 Integer 대비 Floating-Point Types**

□ *integer*는 fractional part가 없는 것(소숫점 이하가 없다)

○ Integers는 binary numbers로 저장된다



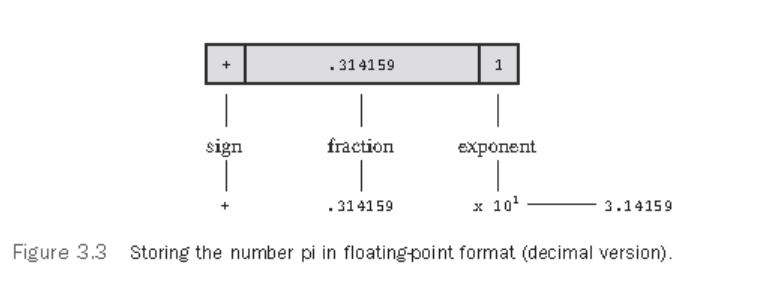
**3.2.3 Floating-Point Number**

□ *real number* 라 부른다/실수

○ Real number는 integer 숫자 간의 소숫점 이하의 숫자를 포함

○ floating-point 숫자: 2.75, 3.16E7, 7.00, and 2e–8

• Floating-point 표현은 fractional part 와 exponent part가 있고 두개의 part가 분리 저장된다



**3.3 기본적인 C Data Types**

□ how to declare a variable, how to represent a constant with a literal value, such as 5 or 2.78

**3.3.1 The int Type**

□ The int type is a signed integer.

○ use 16 bits to store an int

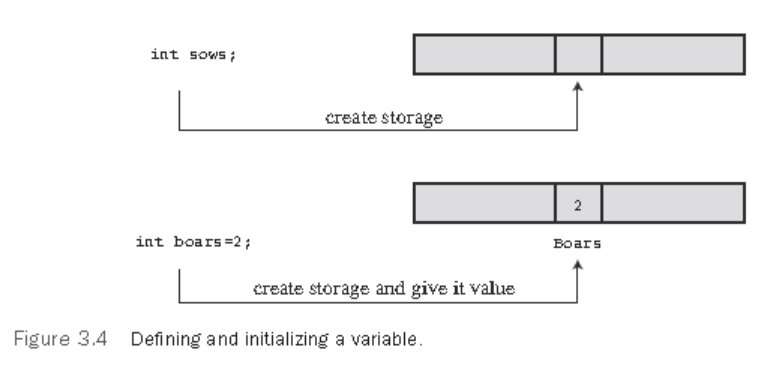
• This allows a range in values from –32768 to 32767

**3.3.2 Initializing a Variable**

int hogs = 21;

int cows = 32, goats = 14;

int dogs, cats = 94; /\* valid, but poor, form \*/



**3.3.3 Type int Constants**

□ *integer constants* , also called *integer literals* .

**3.3.4 Printing int Values**

□ The %d is called a *format specifier* because it indicates the form that printf() uses to display a value

Listing 3.2 The print1.c Program

/\* print1.c-displays some properties of printf() \*/

#include <stdio.h>

int main(void)

{

int ten = 10;

int two = 2;

printf("Doing it right: ");

printf("%d minus %d is %d\n", ten, 2, ten - two );

printf("Doing it wrong: ");

printf("%d minus %d is %d\n", ten ); // forgot 2 arguments

return 0;

}

**3.3.5 Octal and Hexadecimal**

□ C assumes that integer constants are decimal, or base 10, numbers

□ octal (base 8) and hexadecimal (base 16) numbers

* the hexadecimal value 35 is the bit pattern 0011 0101
* In C, special prefixes indicate which number base you are using
* A prefix of 0x or 0X (zero-ex) means that you are specifying a hexadecimal value, so 16 is written as 0x10 , or 0X10 , in hexadecimal
* a 0 (zero) prefix means that you are writing in octal.

• the decimal value 16 is written as 020 in octal

**3.3.6 Displaying Octal and Hexadecimal**

□ To display an integer in octal notation instead of decimal, use %o instead of %d

□ To display an integer in hexadecimal, use %x

□ If you want to display the C prefixes, you can use specifiers %#o , %#x , and %#X to generate the 0 , 0x ,

and 0X prefixes respectively.

Listing 3.3 The bases.c Program

/\* bases.c--prints 100 in decimal, octal, and hex \*/

#include <stdio.h>

int main(void)

{

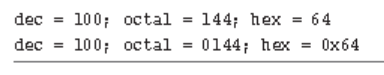
int x = 100;

printf("dec = %d; octal = %o; hex = %x\n", x, x, x);

printf("dec = %d; octal = %#o; hex = %#x\n", x, x, x);

return 0;

}



**3.3.7 Other Integer Types**

□ to modify the basic integer type: short , long , and unsigned .

* The type short int
* The type long int
* The type long long int
* The type unsigned int
* The types unsigned long int
* The keyword signed
* any of the signed types to make your intent explicit
  + - * short , short int , signed short , and signed short int are all names for the same type.

**3.3.8 Declaring Other Integer Types**

long int estine;

long johns;

short int erns;

short ribs;

unsigned int s\_count;

unsigned players;

unsigned long headcount;

unsigned short yesvotes;

long long ago;

**3.3.9 Integer Overflow**

□ uses the %u specifier to display unsigned int values

/\* toobig.c-exceeds maximum int size on our system \*/

#include <stdio.h>

int main(void)

{

int i = 2147483647;

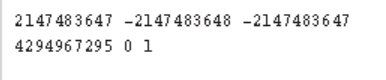
unsigned int j = 4294967295;

printf("%d %d %d\n", i, i+1, i+2);

printf("%u %u %u\n", j, j+1, j+2);

return 0;

}



* The unsigned integer j reaches its maximum value, it starts over at the beginning.
* the int variable i begins at –2147483648

**3.3.10 long Constants and long long Constants**

□ treats the integer 7 as 16 bits and the integer 7L as 32 bits

□ The l and L suffixes can also be used with octal and hex integers, as in 020L and 0x10L .

□ use an ll or LL suffix to indicate a long long value, as in 3LL

□ Add a u or U to the suffix for unsigned long long , as in 5ull or 10LLU or 6LLU or 9Ull

**3.3.11 Printing short , long , long long , and unsigned types**

□ To print an unsigned int number, use the %u notation. To print a long value, use the %ld format specifier.

* use %lx to print a long integer in hexadecimal format and %lo to print in octal format
* use an h prefix for short types
* %hd displays a short integer in decimal form, and %ho displays a short integer in octal form.
* use the %lu notation for printing unsigned long types

Listing 3.4 The print2.c Program

/\* print2.c-more printf() properties \*/

#include <stdio.h>

int main(void)

{

unsigned int un = 3000000000; /\* system with 32-bit int \*/

short end = 200; /\* and 16-bit short \*/

long big = 65537;

long long verybig = 12345678908642;

printf("un = %u and not %d\n", un, un);

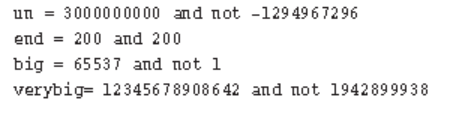
printf("end = %hd and %d\n", end, end);

printf("big = %ld and not %hd\n", big, big);

printf("verybig= %lld and not %ld\n", verybig, verybig);

return 0;

}



**3.4 Using Characters: Type char**

□ the char type actually stores integers

* The most commonly used code in the U.S. is the ASCII code
* the integer value 65 represents an uppercase *A* .
* The standard ASCII code runs numerically from 0 to 127
* developed a standard called ISO/IEC 10646 for character sets.: International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC)
* the Unicode standard

**3.4.1 Declaring Type char Variables**

char response;

char itable, latan;

**3.4.2 Character Constants and Initialization**

char grade = 'A';

* A single character contained between single quotes is a C *character constant* .

char broiled; /\* declare a char variable \*/

broiled = 'T'; /\* OK \*/

broiled = T; /\* NO! Thinks T is a variable \*/

broiled = "T"; /\* NO! Thinks "T" is a string \*/

char grade = 65; /\* ok for ASCII, but poor style \*/

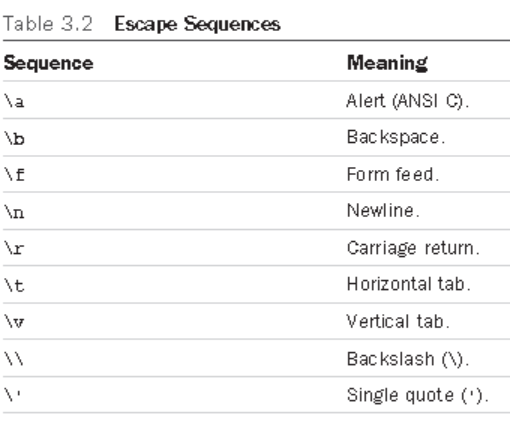
**3.4.2 Nonprinting Characters**

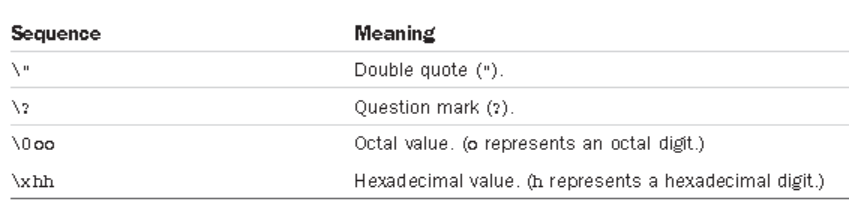
□ represent actions such as backspacing or going to the next line or making the terminal bell ring (or speaker beep). How can these be represented?

* just use the ASCII code.

char beep = 7;

* to use special symbol sequences. These are called *escape sequences*





char nerf = '\n';

**3.4.3 printing Characters**

□ uses %c to indicate that a character should be printed

Listing 3.5 The charcode.c Program

/\* charcode.c-displays code number for a character \*/

#include <stdio.h>

int main(void)

{

char ch;

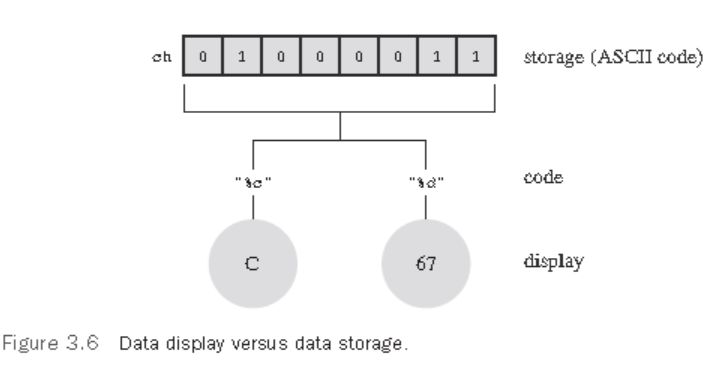
printf("Please enter a character.\n");

scanf("%c", &ch); /\* user inputs character \*/

printf("The code for %c is %d.\n", ch, ch);

return 0;

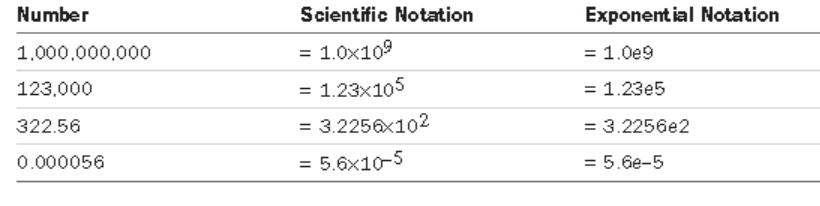
}



**3.5 Types float , double , and long double**

□ make use of *floating-point* number

* float , double , or long double .



□use 32 bits to store a floating-point number.

* Eight bits are used to give the exponent its value and sign, and 24 bits are used to represent the nonexponent part, called the *mantissa* or *significand* , and its sign.

□a double (for double precision) floating-point type

* use all 32 additional bits for the nonexponent part.
* increases the number of significant figures and reduces round-off errors.

**3.6 Declaring Floating-Point Variables**

float noah, jonah;

double trouble;

float planck = 6.63e-34;

long double gnp;

**3.7 Printing Floating-Point Values**

□ uses the %f format specifier to print type float and double

□ uses %e to print them in exponential notation

□ The long double type requires the %Lf , %Le , and %La specifiers to print that type.

Listing 3.7 The showf\_pt.c Program

/\* showf\_pt.c -- displays float value in two ways \*/

#include <stdio.h>

int main(void)

{

float aboat = 32000.0;

double abet = 2.14e9;

long double dip = 5.32e-5;

printf("%f can be written %e\n", aboat, aboat);

// next line requires C99 or later compliance

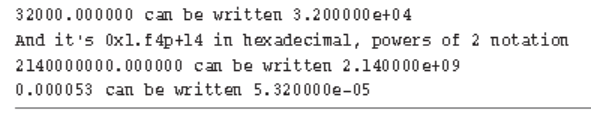
printf("And it's %a in hexadecimal, powers of 2 notation\n", aboat);

printf("%f can be written %e\n", abet, abet);

printf("%Lf can be written %Le\n", dip, dip);

return 0;

}



**3.8 Beyond the Basic Types**

□ What about a character string type?

□ C does have other types derived from the basic types

○ arrays, pointers, structures, and unions.

**3.9 type sizes**

□ C has a built-in operator called sizeof that gives sizes in bytes

* provide a %zd specifier for this type used by sizeof

• require %u or %lu

Listing 3.8 The typesize.c Program

//\* typesize.c -- prints out type sizes \*/

#include <stdio.h>

int main(void)

{

/\* c99 provides a %zd specifier for sizes \*/

printf("Type int has a size of %zd bytes.\n", sizeof(int));

printf("Type char has a size of %zd bytes.\n", sizeof(char));

printf("Type long has a size of %zd bytes.\n", sizeof(long));

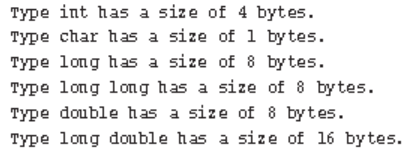
printf("Type long long has a size of %zd bytes.\n",sizeof(long long));

printf("Type double has a size of %zd bytes.\n",sizeof(double));

printf("Type long double has a size of %zd bytes.\n",sizeof(long double));

return 0;

}



**3.10 Using Data Types**

□ When you initialize a variable, match the constant type to the variable type

int apples = 3; /\* RIGHT \*/ int oranges = 3.0; /\* POOR FORM \*/

□ When you initialize a variable of one numeric type to a value of a different type, C converts the value to match the variable

int cost = 12.99; /\* initializing an int to a double \*/

○ converting floating-point values to integers

* throws away the decimal part ( truncation ) instead of rounding

float pi = 3.1415926536; /\* initializing a float to a double \*/

3.11 Arguments and Pitfalls

□ printf(): The items of information passed to a function -> arguments

* A series of characters in quotes, such as "Hello, pal." , is called a string
* The printf() and scanf() functions use the first argument to indicate how many additional arguments are coming

printf("%d cats ate %d cans of tuna\n", cats, cans);

□ C now has a function-prototyping mechanism

* checks whether a function call has the correct number and correct kind of arguments

