Sams Teach Yourself C in 24 Hours

Hour 4 - Data Types and Names in C

What's in a name? That which we call a rose
By any other name would smell as sweet.

—W. Shakespeare

You learned how to make a valid name for a C function in Hour 3, "The Essentials of C Programs." Now, you're going to learn more about naming a variable and the C keywords reserved by the C compiler in this hour.

Also in this hour you're going to learn about the four data types of the C language in detail:

- char data type
- int data type
- float data type
- double data type

C Keywords

The C language reserves certain words that have special meanings to the language. Those reserved words are sometimes called C keywords. You should not use the C keywords as variable, constant, or function names in your program. The following are the 32 reserved C keywords:

Keyword	Description
auto	Storage class specifier
break	Statement
case	Statement
char	Type specifier
const	Storage class modifier
continue	Statement
default	Label
do	Statement
double	Type specifier
else	Statement
enum	Type specifier
extern	Storage class specifier
float	Type specifier
for	Statement
goto	Statement
if	Statement
int	Type specifier
long	Type specifier
register	Storage class specifier
return	Statement
short	Type specifier
signed	Type specifier
sizeof	Operator
static	Storage class specifier
struct	Type specifier
switch	Statement
typedef	Statement
union	Type specifier
unsigned	Type specifier
void	Type specifier
volatile	Storage class modifier
while	Statement

Don't worry if you can't remember all the C keywords the first time through. In the rest of the book, you'll become more familiar with them and start to use many of the keywords through examples and exercises.

Note that all C keywords are written in lowercase letters. As I've mentioned, C is a case-sensitive language. Therefore, int, as shown in the list here, is considered as a C keyword, but INT is not.

The char Data Type

An object of the char data type represents a single character of the character set used by your computer. For example, A is a character, and so is a. But 7 is a number.

But a computer can only store numeric code. Therefore, characters such as A, a, B, b, and so on all have a unique numeric code that is used

by computers to represent the characters. Usually, a character takes 8 bits (that is, 1 byte) to store its numeric code.

For many computers, the ASCII codes are the de facto standard codes to represent a character set. (ASCII, just for your information, stands for American Standard Code for Information Interchange.) The original ASCII character set has only 128 characters because it uses the lower 7 bits that can represent 2^7 (that is, 128) characters.

On IBM-compatible PCs, however, the character set is extended to contain a total of 256 (that is, 2^8) characters. Appendix C, "ASCII Character Set," gives a list of the 256 characters.

Character Variables

A variable that can represent different characters is called a character variable.

You can set the data type of a variable to char by using the following declaration format:

```
char  variablename;
```

where variablename is the place where you put the name of a variable.

If you have more than one variable to declare, you can either use the following format:

```
char  variablename1;
char  variablename2;
char  variablename3;
```

or this one:

```
char  variablename1, variablename2, variablename3;
```

Character Constants

A character enclosed in single quotes (`) is called a character constant. For instance, `A', `a', `B', and `b' are all character constants that have their unique numeric values in the ASCII character set.

You can remember to use the single quote (`), instead of the double quote ("), in character constants because a character constant represents a single character. You'll see later in the book that the double quote is used to indicate a string of characters.

From the ASCII character set listed in Appendix C, you will find that the unique numeric (decimal) values of `A', `a', `B', and `b' are 65, 97, 66, and 98, respectively. Therefore, given x as a character variable, for instance, the following two assignment statements are equivalent:

```
x = `A';
x = 65;
```

So are the following two statements:

```
x = `a';
x = 97;
```

Later in this hour, you'll see a program (in Listing 4.2) that converts numeric values back to the corresponding characters.

The Escape Character (\)

Actually, you first saw the escape character (\) in Hour 2, "Writing Your First C Program," when you learned to use the newline character (\n) to break a message into two pieces. In the C language, the backslash (\) is called the escape character; it tells the computer that a special character follows.

For instance, when the computer sees \ in the newline character \n, it knows that the next character, n, causes a sequence of a carriage return and a line feed.

Besides the newline character, several other special characters exist in the C language, such as the following:

Character	Description
\b	The backspace character; moves the cursor to the left one character.
\f	The form-feed character; goes to the top of a new page.
\r	The return character; returns to the beginning of the current line.
\t	The tab character; advances to the next tab stop.

Printing Out Characters

You already know that the printf() function, defined in the C header file stdio.h, can be used to print out messages on the screen. In this section, you're going to learn to use the character format specifier, %c, which indicates to the printf() function that the argument to be printed is a character. Let's first have a look at the program in Listing 4.1, which prints out characters on the screen.

TYPE
Listing 4.1. Printing out characters on the screen.

```
1:  /* 04L01.c: Printing out characters */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     char c1;
```

```
7:     char c2;
8:
9:     c1 = `A';
10:    c2 = `a';
11:    printf("Convert the value of c1 to character: %c.\n", c1);
12:    printf("Convert the value of c2 to character: %c.\n", c2);
13:    return 0;
14: }
```

OUTPUT

After the executable file of 04L01.c in Listing 4.1 is created, you can run it to see what will be printed out on the screen. On my machine, the executable file is named as 04L01.exe. The following is the output printed on the screen after I run the executable from a DOS prompt:

```
C:\app> 04L01
Convert the value of c1 to character: A.
Convert the value of c2 to character: a.
C:\app>
```

ANALYSIS

As you know, line 2 includes the header file, stdio.h, for the printf() function. Lines 5_14 make up the main() function body.

Lines 6 and 7 declare two character variables, c1 and c2, while lines 9 and 10 assign c1 and c2 with the character constants `A' and `a', respectively.

Note that the %c format specifier is used in the printf() function in lines 11 and 12, which tells the computer that the contents contained by c1 and c2 should be printed as characters. When the two statements in lines 11 and 12 are executed, two characters are formatted and output to the screen, based on the numeric values contained by c1 and c2, respectively.

Now look at the program shown in Listing 4.2. This time, %c is used to convert the numeric values back to the corresponding characters.

TYPE

Listing 4.2. Converting numeric values back to characters.

```
1:  /* 04L02.c: Converting numeric values back to characters */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     char c1;
7:     char c2;
8:
9:     c1 = 65;
10:    c2 = 97;
11:    printf("The character that has the numeric value of 65 is: %c.\n", c1);
12:    printf("The character that has the numeric value of 97 is: %c.\n", c2);
13:    return 0;
14: }
```

OUTPUT

The following is the output printed on the screen after I run the executable file, 04L02.exe, from a DOS prompt:

```
C:\app> 04L02
The character that has the numeric value of 65 is: A.
The character that has the numeric value of 97 is: a.
C:\app>
```

ANALYSIS

The program in Listing 4.2 is similar to the one in Listing 4.1 except for the two statements in lines 9 and 10. Note that in lines 9 and 10 of Listing 4.2, the character variables c1 and c2 are assigned 65 and 97, respectively.

As you know, 65 is the numeric value (decimal) of the A character in the ASCII character set; 97 is the numeric value of a. In lines 11 and 12, the %c format specifier converts the numeric values, 65 and 97, into the A and a, respectively. The A and a characters are then printed out on the screen.

The int Data Type

You saw the integer data type in Hour 3. The int keyword is used to specify the type of a variable as an integer. Integer numbers are also called whole numbers, which have no fractional part or decimal point. Therefore, the result of an integer division is truncated, simply because any fraction part is ignored.

Depending on the operating system and the C compiler you're using, the length of an integer varies. On most UNIX workstations, for example, an integer is 32 bits long, which means that the range of an integer is from 2147483647 (that is, 2^{31}_1) to -2147483648. The range of a 16-bit integer is from 32767 (that is, 2^{15}_1) to -32768.

The C compiler I'm using for this book is Visual C++ 1.5, which only provides the 16-bit integer, while a 32-bit version of Visual C++, such as Visual C++ 4.0 or Visual C++ 5.0, supports the 32-bit integer.

Declaring Integer Variables

You also saw the declaration of an integer in Hour 3. The following shows the basic declaration format:

```
int  variablename;
```

Similar to the character declaration, if you have more than one variable to declare, you can use either the format like this

```
int  variablename1;
int  variablename2;
int  variablename3;
```

or like this:

```
int  variablename1, variablename2, variablename3;
```

Here variablename1, variablename2, and variablename3 indicate the places where you put the names of int variables.

Showing the Numeric Values of Characters

Like the character format specifier (%c) that is used to format a single character, %d, called the integer format specifier, is used to format an integer. You might recall that in line 16 of Listing 3.2, %d is used in the printf() function to format the second argument of the function to an integer.

In this section you're going to study a program, shown in Listing 4.3, that can print out the numeric values of characters by using the integer format specifier %d with printf().

TYPE
Listing 4.3. Showing the numeric values of characters.

```
1:  /* 04L03.c: Showing the numeric values of characters */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     char c1;
7:     char c2;
8:
9:     c1 = `A';
10:    c2 = `a';
11:    printf("The numeric value of A is: %d.\n", c1);
12:    printf("The numeric value of a is: %d.\n", c2);
13:    return 0;
14: }
```

OUTPUT
I get the following output on the screen after running the executable file, 04L03.exe:

```
C:\app> 04L03
The numeric value of A is: 65.
The numeric value of a is: 97.
C:\app>
```

ANALYSIS
You may find that the program in Listing 4.3 is quite similar to the one in Listing 4.1. As a matter of fact, I simply copied the source code from Listing 4.1 to Listing 4.3 and made changes in lines 11 and 12. The major change I made was to replace the character format specifier (%c) with the integer format specifier (%d).

The two statements in lines 11 and 12 format the two character variables (c1 and c2) by using the integer format specifier %d, and then print out two messages showing the numeric values 65 and 97 that represent, respectively, the characters A and a in the ASCII character set.

The float Data Type

The floating-point number is another data type in the C language. Unlike an integer number, a floating-point number contains a decimal point. For instance, 7.01 is a floating-point number; so are 5.71 and -3.14. A floating-point number is also called a real number.

A floating-point number is specified by the float keyword in the C language. Floating-point numbers can be suffixed with f or F to specify float. A floating-point number without a suffix is double by default. The double data type is introduced later in this lesson.

Like an integer number, a floating-point number has a limited range. The ANSI standard requires that the range be at least plus or minus $1.0*10^{37}$. Normally, a floating-point number is represented by taking 32 bits. Therefore, a floating-point number in C is of at least six digits of precision. That is, for a floating-point number, there are at least six digits (or decimal places) on the right side of the decimal point.

Not like an integer division from which the result is truncated and the fraction part is discarded, a floating-point division produces another floating-point number. A floating-point division is carried out if both the divisor and the dividend, or one of them, are floating-point numbers.

For instance, 571.2 / 10.0 produces another floating-point number, 57.12. So do 571.2 / 10 and 5712 / 10.0.

Declaring Floating-Point Variables

The following shows the declaration format for a floating-point variable:

```
float  variablename;
```

Similar to the character or integer declaration, if you have more than one variable to declare, you can either use the format like this:

```
float  variablename1;
float  variablename2;
float  variablename3;
```

or like the following one:

```
float  variablename1, variablename2, variablename3;
```

The Floating-Point Format Specifier (%f)

Also, in C, you can use the floating-point format specifier (%f) to format your output. Listing 4.4 gives an example showing how to use the format specifier %f with the printf() function.

TYPE
Listing 4.4. Printing out results of integer and floating-point divisions.

```
1:  /* 04L04.c: Integer vs. floating-point divisions */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int int_num1, int_num2, int_num3;   /* Declare integer variables */
7:     float flt_num1, flt_num2, flt_num3; /* Declare floating-point variables */
8:
9:     int_num1 = 32 / 10;    /* Both divisor and dividend are integers */
10:    flt_num1 = 32 / 10;
11:    int_num2 = 32.0 / 10;  /* The divisor is an integer */
12:    flt_num2 = 32.0 / 10;
13:    int_num3 = 32 / 10.0;  /* The dividend is an integer */
14:    flt_num3 = 32 / 10.0;
15:
16:    printf("The integer divis. of 32/10 is: %d\n", int_num1);
17:    printf("The floating-point divis. of 32/10 is: %f\n", flt_num1);
18:    printf("The integer divis. of 32.0/10 is: %d\n", int_num2);
19:    printf("The floating-point divis. of 32.0/10 is: %f\n", flt_num2);
20:    printf("The integer divis. of 32/10.0 is: %d\n", int_num3);
21:    printf("The floating-point divis. of 32/10.0 is: %f\n", flt_num3);
22:    return 0;
23: }
```

OUTPUT
The following output is a copy from the screen after the executable file, 04L04.exe, is run on my machine (I did get several warning messages about type conversions while I was compiling the program in Listing 4.4, but I ignored them all because I'd like to create an executable file and show you the differences between the int data type and the float data type.):

```
C:\app> 04L04
The integer divis. of 32/10 is: 3
The floating-point divis. of 32/10 is: 3.000000
The integer divis. of 32.0/10 is: 3
The floating-point divis. of 32.0/10 is: 3.200000
The integer divis. of 32/10.0 is: 3
The floating-point divis. of 32/10.0 is: 3.200000
C:\app>
```

ANALYSIS
Inside the main() function, the two statements in lines 6 and 7 declare three integer variables, int_num1, int_num2, and int_num3, and three floating-point variables, flt_num1, flt_num2, and flt_num3.

Lines 9 and 10 assign the result of 32/10 to int_num1 and flt_num1, respectively; 32.0 / 10 to int_num2 and flt_num2 in lines 11 and 12, and 32 / 10.0 to int_num3 and flt_num3 in lines 13 and 14.

Then, lines 16_21 print out the values contained by the three int variables and the three floating-point variables. Note that, %d is used for the integer variables, and the floating-point specifier (%f) is used for formatting the floating-point variables in the printf() function.

Because the truncation occurs in the integer division of 32 / 10, flt_num1 contains 3.000000, not 3.200000, which you can see from the second line of the output. However, flt_num2 and flt_num3 are assigned 3.200000, because both 32.0 / 10 and 32 / 10.0 are considered as the floating-point division.

But int_num2 and int_num3, as integer variables, discard respectively the fraction parts of the floating-point divisions of 32.0 / 10 and 32 / 10.0. Therefore, you just see the integer 3 in both the third and fifth lines of the output.

The double Data Type

In the C language, a floating-point number can also be represented by another data type, called the double data type. In other words, you can specify a variable by the double keyword, and assign the variable a floating-point number.

The difference between a double data type and a float data type is that the former normally uses twice as many bits as the latter. Therefore, a double floating-point number is of at least 10 digits of precision, although the ANSI standard does not specify it for the double data type.

In Hour 8, "More Operators," you'll learn to use the sizeof operator to obtain the length of a data type, such as char, int, float, or double, specified on your computer system.

Using Scientific Notation

The C language uses scientific notation to help you write lengthy floating-point numbers.

In scientific notation, a number can be represented by the combination of the mantissa and the exponent. The format of the notation is that the mantissa is followed by the exponent, which is prefixed by e or E. Here are two examples:

```
[mantissa]e[exponent],
```

and

`[mantissa]E[exponent].`

For instance, 5000 can be represented by 5e3 in scientific notation. Likewise, -300 can be represented by -3e2, and 0.0025 by 2.5e-3.

Correspondingly, the format specifier, %e or %E, is used to format a floating-point number in scientific notation. The usage of %e or %E in the printf() function is the same as %f.

Naming a Variable

You've learned how to make a valid function name. In this section, let's focus on naming variables. Function names and variable names are both identifiers in C.

The following are all the characters you can use to make a valid variable name:

- Characters A through Z and a through z.
- Digit characters 0 through 9, which can be used in any position except the first of a variable name.
- The underscore character (_).

For instance, stop_sign, Loop3, and _pause are all valid variable names.

Now, let's see what you cannot use in variable naming:

- A variable name cannot contain any C arithmetic signs.
- A variable name cannot contain any dots (.).
- A variable name cannot contain any apostrophes (`).
- A variable name cannot contain any other special symbols such as *, @, #, ?, and so on.

Some invalid variable names, for example, are, 4flags, sum-result, method*4, and what_size?.

WARNING

Never use the C keywords reserved in the C language, or the names of the standard C library functions, as variable names in your C program.

Summary

In this lesson you've learned about the following:

- The C keywords reserved in the C language
- The char data type and the %c format specifier
- The int data type and the %d format specifier
- The float data type and the %f format specifier
- Floating-point numbers can be suffixed with f or F to specify float. A floating-point number without suffix is double by default.
- The possible ranges of the char, int, and float data types
- The double data type
- Scientific notation and the %e and %E format specifiers
- The rules you have to follow to make a valid variable name

In the next lesson you'll learn more about the printf() function and other functions to deal with input and output.

Q&A

Q Why do characters have their unique numeric values?

A Characters are stored in computers in the form of bits. The combinations of bits can be used to represent different numeric values. A character has to have a unique numeric value in order to distinguish itself. Many computer systems support the ASCII character set, which contains a set of unique numeric values for up to 256 characters.

Q How can you declare two character variables?

A There are two ways to do the declaration. The first one is

```
      …char variable-name1, variable-name2;

char  variable-name1;
      char  variable-name2;
```

Q What are %c, %d, and %f?

A These are format specifiers. %c is used to obtain the character format; %d is for the integer format; %f is for the floating-point format. %c, %d, and %f are often used with C functions such as printf().

Q What are the main differences between the int data type (integer) and the float data type (floating-point)?

A First, an integer does not contain any fraction parts, but a floating-point number does. A floating-point number must have a decimal point. In C, the float data type takes more bits than the int data type. In other words, the float data type has a larger range of numeric values than the int data type. Also, the integer division truncates the fraction part. For instance, the integer division of 16/10 produces a result of 1, not 1.6.

Workshop

To help solidify your understanding of this hour's lesson, you are encouraged to answer the quiz questions and finish the exercises provided

in the Workshop before you move to the next lesson. The answers and hints to the questions and exercises are given in Appendix E, "Answers to Quiz Questions and Exercises."

Quiz

1. Are the integer divisions of 134/100 and 17/10 equal?
2. Is the result of 3000 + 1.0 a floating-point number? How about 3000/1.0?
3. How can you represent the following numbers in scientific notation?
    - 3500
    - 0.0035
    - -0.0035
4. Are the following variable names valid?
    - 7th_calculation
    - Tom's_method
    - _index
    - Label_1

Exercises

1. Write a program that prints out the numeric values of characters Z and z.
2. Given two numeric values, 72 and 104, write a program to print out the corresponding two characters.
3. For a 16-bit integer variable, can you assign the variable with an integer value of 32768?
4. Given the declaration double dbl_num = 123.456;, write a program that prints out the value of dbl_num in both floating-point and scientific notation formats.
5. Write a program that can print out the numeric value of the newline character (\n). (Hint: assign `\n' to a character variable.)