Sams Teach Yourself C in 24 Hours

Hour 8 - More Operators

Civilization advances by extending the number of important operations we can perform without thinking about them.

—A. N. Whitehead

In Hour 6, "Manipulating Data with Operators," you learned about some important operators in C, such as the arithmetic assignment operators, the unary minus operator, the increment and decrement operators, and the relational operators. In this lesson you'll learn about more operators, including

- The sizeof operator
- Logical operators
- Bit-manipulation operators
- The conditional operator

Measuring Data Sizes

You may remember in Hour 4, "Data Types and Names in C," I mentioned that each data type has its own size. Depending on the operating system and the C compiler you're using, the size of a data type varies. For example, on most UNIX workstations, an integer is 32 bits long, while most C compilers only support 16-bit integers on a DOS-based machine.

So, how do you know the size of a data type on your machine? The answer is that you can measure the data type size by using the sizeof operator provided by C.

The general form of the sizeof operator is

```
sizeof (expression)
```

Here expression is the data type or variable whose size is measured by the sizeof operator. The value of the size is returned, in units of bytes, by the sizeof operator. For instance, if an integer is 16 bits long, the value returned by the sizeof operator will be 2 (bytes). (Note that 8 bits are equal to 1 byte.)

The parentheses are optional in the general form of the operator. If the expression is not a C keyword for a data type, the parentheses can be discarded.

For instance, the following statement

```
size = sizeof(int);
```

measures the size of the int data type and returns the number of bytes required by the data type to an int variable size.

The program in Listing 8.1 finds the sizes of the char, int, float, and double data types on my machine.

TYPE
Listing 8.1. Using the sizeof operator.

```
1:  /* 08L01.c: Using the sizeof operator */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     char   ch = ` `;
7:     int    int_num = 0;
8:     float  flt_num = 0.0f;
9:     double dbl_num = 0.0;
10:
11:    printf("The size of char is: %d-byte\n", sizeof(char));
12:      printf("The size of ch is: %d-byte\n", sizeof ch );
13:        printf("The size of int is: %d-byte\n", sizeof(int));
14:          printf("The size of int_num is: %d-byte\n", sizeof int_num);
15:            printf("The size of float is: %d-byte\n", sizeof(float));

16:          printf("The size of flt_num is: %d-byte\n", sizeof flt_num);
17:        printf("The size of double is: %d-byte\n", sizeof(double));
18:      printf("The size of dbl_num is: %d-byte\n", sizeof dbl_num);
19:    return 0;
20: }
```

OUTPUT
After this program is compiled and linked, an executable file, 08L01.exe, is created. The following is the output printed on the screen after the executable is run from a DOS prompt on my machine:

```
C:\app> 08L01
The size of char is: 1-byte
The size of ch is: 1-byte
The size of int is: 2-byte
The size of int_num is: 2-byte
The size of float is: 4-byte
The size of flt_num is: 4-byte
The size of double is: 8-byte
The size of dbl_num is: 8-byte
```

```
C:\app>
```

ANALYSIS
Line 2 in Listing 8.1 includes the header file stdio.h for the printf() function used in the statements inside the main() function body. Lines 6_9 declare a char variable (ch), an int variable (int_num), a float variable (flt_num), and a double variable (dbl_num), respectively. Also, these four variables are initialized. Note that in line 8, the initial value to flt_num is suffixed with f to specify float. (As you learned in Hour 4, you can use f or F to specify the float type for a floating-point number.)

Lines 11 and 12 display the size of the char data type, as well as the char variable ch. Note that the sizeof operator is used in both line 11 and line 12 to obtain the number of bytes the char data type or the variable ch can have. Because the variable ch is not a keyword in C, the parentheses are discarded for the sizeof operator in line 12.

The first two lines in the output are printed out by executing the two statements in line 11 and 12, respectively. From the output, you see that the size of the char data type is 1 byte long, which is the same as the size of the variable ch. This is not surprising because the variable ch is declared as the char variable.

Likewise, lines 13 and 14 print out the sizes of the int data type and the int variable int_num by using the sizeof operator. You see that the size of each is 2 bytes.

Also, by using the sizeof operator, lines 15_18 give the sizes of the float data type, the float variable flt_num, the double data type, and the double variable dbl_num, respectively. The results in the output section show that the float data type and the variable flt_num have the same size (4 bytes). The double data type and the variable dbl_num are both 8 bytes long.

From the output you see that char uses 1 byte, while an int uses 2 bytes. Accordingly, while variables of type char have been used to store integers, they cannot store integers of the same range as a variable of type int can.

Everything Is Logical

Now, it's time for you to learn about a new set of operators: logical operators.

There are three logical operators in the C language:

&& The logical AND operator
|| The logical OR operator
! The logical negation operator

The logical AND operator (&&) evaluates the truth or falseness of pairs of expressions. If both expressions are true, the logical AND operator returns 1. Otherwise, the operator returns 0.

However, the logical OR operator (||) returns 1 if at least one of the expressions is true. The || operator returns 0 if both expressions are false.

Only one operand (or expression) can be taken by the logical negation operator (!). If the operand is true, the ! operator returns 0; otherwise, the operator returns 1.

NOTE
In C, if an expression or operator returns a nonzero value, the expression returns TRUE. If an expression or operator returns 0, the expression returns FALSE. In other words, TRUE can be used to represent any nonzero value returned by an expression or operator; FALSE is equivalent to 0.

The following three sections contain examples that show you how to use the three logical operators.

The Logical AND Operator (&&)

A general format of using the logical AND operator is:

```
exp1 && exp2
```

where exp1 and exp2 are two expressions evaluated by the AND operator.

We can have a table that shows the return values of the AND operator under the following conditions when exp1 and exp2 return 1 or 0, respectively. See Table 8.1, which can be called the truth table of the AND operator.

Table 8.1. The values returned by the AND operator.

exp1	exp2	Value Returned by &&
1	1	1
1	0	0
0	1	0
0	0	0

Listing 8.2 is an example of using the logical AND operator (&&).

TYPE
Listing 8.2. Using the logical AND operator (&&).

```
1:  /* 08L02.c: Using the logical AND operator */
```

```
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int   num;
7:
8:     num = 0;
9:     printf("The AND operator returns: %d\n",
10:           (num%2 == 0) && (num%3 == 0));
11:    num = 2;
12:    printf("The AND operator returns: %d\n",
13:           (num%2 == 0) && (num%3 == 0));
14:    num = 3;
15:    printf("The AND operator returns: %d\n",
16:           (num%2 == 0) && (num%3 == 0));
17:    num = 6;
18:    printf("The AND operator returns: %d\n",
19:           (num%2 == 0) && (num%3 == 0));
20:
21:    return 0;
22: }
```

OUTPUT

After this program is compiled and linked, an executable file, 08L02.exe, is created. The following is the output printed on the screen after the executable is run from a DOS prompt on my machine:

```
C:\app> 08L02
The AND operator returns: 1
The AND operator returns: 0
The AND operator returns: 0
The AND operator returns: 1
C:\app>
```

ANALYSIS

In Listing 8.2, an integer variable, num, is declared in line 6 and initialized for the first time in line 8. Lines 9 and 10 print out the value returned by the logical AND operator in the following expression:

```
(num%2 == 0) && (num%3 == 0)
```

Here you see two relational expressions, num%2 == 0 and num%3 == 0. In Hour 3, "The Essentials of C Programs," you learned that the arithmetic operator % can be used to obtain the remainder after its first operand is divided by the second operand. Therefore, num%2 yields the remainder of num divided by 2. The relational expression num%2 == 0 returns 1 (TRUE) if the remainder is equal to 0—that is, the value of num can be divided evenly by 2. Likewise, if the value of num can be divided by 3, the relational expression num%3 == 0 returns 1 as
well. Then, according to the truth table of the && operator (see Table 8.1), we know that the combination of the logical AND operator (&&) and the two relational expressions yields 1 if the two relational expressions both return 1; otherwise, it yields 0.

In our case, when num is initialized to 0 in line 8, both 0%2 and 0%3 yield remainders of 0 so that the two relational expressions return TRUE. Therefore, the logical AND operator returns 1.

However, when num is assigned with the value of 2 or 3 as shown in lines 11 and 14, the logical AND operator in line 13 or line 16 returns 0. The reason is that 2 or 3 cannot be divided by both 2 and 3.

Line 17 then assigns num the value of 6. Because 6 is a multiple of both 2 and 3, the logical
AND operator in line 19 returns 1, which is printed out by the printf() function in lines 18 and 19.

From the program in Listing 8.2, you see several single statements spanning into multiple lines. The output from the program in Listing 8.2 shows the values returned by the AND operator when num is assigned with different values.

The Logical OR Operator (||)

As mentioned earlier, the logical OR operator returns 1 if at least one of the expressions is true. The || operator returns 0 if both expressions are false.

A general format of using the logical OR operator is:

```
exp1 || exp2
```

where exp1 and exp2 are two expressions evaluated by the OR operator.

Table 8.2 shows the truth table of the OR operator.

exp1	exp2	Value Returned by ‖
1	1	1
1	0	1
0	1	1
0	0	0

The program in Listing 8.3 shows how to use the logical OR operator (||).

TYPE
Listing 8.3. Using the logical OR operator (||).

```
1:  /* 08L03.c: Using the logical OR operator */
```

```
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int   num;
7:
8:     printf("Enter a single digit that can be divided\nby both 2 and 3:\n");
9:     for (num = 1; (num%2 != 0) || (num%3 != 0); )
10:       num = getchar() - 48;
11:    printf("You got such a number: %d\n", num);
12:    return 0;
13: }
```

OUTPUT

The following is the output printed on the screen after the executable, 08L03.exe, is run from a DOS prompt on my machine. The numbers in bold font are what I entered. (The Enter key is pressed after each number is entered.) In the range of 0_9, 0 and 6 are the only two numbers that can be divided evenly by both 2 and 3:

```
C:\app> 08L03
Enter a single digit that can be divided
by both 2 and 3:
2
3
4
5
6
You got such a number: 6
C:\app>
```

ANALYSIS

In Listing 8.3, an integer variable, num, is declared in line 6. Line 8 of Listing 8.3 prints out a headline asking the user to enter a single digit. Note that there is a newline character (\n) in the middle of the headline message in the printf() function to break the message into two lines.

In line 9, the integer variable num is initialized in the first expression field of the for statement. The reason to initialize num with 1 is that 1 is such a number that cannot be divided by either 2 nor 3. Thus, the for loop is guaranteed to be executed at least once.

The key part of the program in Listing 8.3 is the logical expression in the for statement:

```
(num%2 != 0) || (num%3 != 0)
```

Here the relational expressions num%2 != 0 and num%3 != 0 are evaluated. According to the truth table of the || operator (see Table 8.2), we know that if one of the relational expression returns TRUE, i.e., the value of num cannot be divided completely by either 2 or 3. Then the logical expression returns 1, which allows the for loop to continue.

The for loop stops only if the user enters a digit that can be divided by both 2 and 3. In other words, when both the relational expressions return FALSE, the logical OR operator yields 0, which causes the termination of the for loop.

You can rewrite the program in Listing 8.3 with the if statement, too.

The Logical Negation Operator (!)

A general format of using the logical OR operator is:

```
!expression
```

where expression is an expression operated by the negation operator.

The truth table of the negation operator is shown in Table 8.3.

Table 8.3. The values returned by the ! operator.

expression	Value Returned by !
1	0
0	1

TYPE

Now, let's take a look at the example, shown in Listing 8.4, that demonstrates how to use the logical negation operator (!).

Listing 8.4. Using the logical negation operator (!).

```
1:  /* 08L04.c: Using the logical negation operator */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int   num;
7:
8:     num = 7;
9:     printf("Given num = 7\n");
10:    printf("!(num < 7)  returns: %d\n", !(num < 7));
11:    printf("!(num > 7)  returns: %d\n", !(num > 7));
12:    printf("!(num == 7) returns: %d\n", !(num == 7));
13:    return 0;
14: }
```

OUTPUT
The following result is obtained by running the executable file 08L04.exe:

```
C:\app> 08L04
Given num = 7
!(num < 7)  returns: 1
!(num > 7)  returns: 1
!(num == 7) returns: 0
C:\app>
```

ANALYSIS
In line 8, note that an integer variable, num, is initialized with 7, which is then displayed by the printf() function in line 9.

In line 10, the relational expression num < 7 returns FALSE (that is, 0), because the value of num is not less than 7. However, by using the logical negation operator, !(num < 7) yields 1. (Refer to the truth table of the ! operator shown in Table 8.3.)

Similarly, the logical expression !(num > 7) returns 1 in line 11.

Because num has the value of 7, the relational expression num == 7 is true; however, the logical expression !(num == 7) in line 12 returns 0 due to the logical negation operator (!).

Manipulating Bits

In previous hours, you learned that computer data and files are made of bits (or bytes). There is even an operator in C_the sizeof operator_that can be used to measure the number of bytes for data types.

In this section, you'll learn about a set of operators that enable you to access and manipulate specific bits.

There are six bit-manipulation operators in the C language:

Operator	Description
&	The bitwise AND operator
|	The bitwise OR operator
^	The bitwise exclusive OR (XOR) operator
~	The bitwise complement operator
>>	The right-shift operator
<<	The left-shift operator

The following two sections give explanations and examples of the bit-manipulation operators.

TIP

It's easy to convert a decimal number into a hex or a binary. Each digit in a hex number consists of four bits. A bit represents a digit in a binary number. Table 8.4 shows the hex numbers (0_F) and their corresponding binary and decimal representations.

Table 8.4. Numbers expressed in different formats.

Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Let's see how to convert a decimal number into a binary, or vice versa. As we know, that binary is a 2-based numbering system. Each digit in a binary number is called a bit and can be 1 or 0. If the position of a bit in a binary number is n, then the bit can have a value of 2 to the power of n. The position of a bit in a binary number is counted from the right of the binary number. The most-right bit is at the position of zero. Thus, given a binary number 1000, we can calculate its decimal value like this:

```
1000 -> 1 * 2^3 + 0 *2^2 + 0 * 2^1 + 0 * 2^0-> 2^3 -> 8 (decimal)
```

That is, the decimal vale of the binary number 1000 is 8.

If we want to convert a decimal number, for example 10, to its binary counterpart, we have the following process:

```
10 -> 2^3 + 2^1 -> 1 *2^3 + 0 * 2^2 + 1 *2^1 + 0 * 2^0 -> 1010 (binary)
```

Likewise, you can convert the rest of the decimal numbers in Table 8.4 to their binary counterparts, or vice versa.

Using Bitwise Operators

The general forms of the bitwise operators are as follows:

```
x & y
x | y
x ^ y
~x
```

Here x and y are operands.

The & operator compares each bit of x to the corresponding bit in y. If both bits are 1, 1 is placed at the same position of the bit in the result. If one of the bits, or two of them, is 0, 0 is placed in the result.

For instance, the expression with two binary operands, 01 & 11, returns 01.

The | operator, however, places 1 in the result if either operand is 1. For example, the expression 01 | 11 returns 11.

The ^ operator places 1 in the result if either operand, but not both, is 1. Therefore, the expression 01 ^ 11 returns 10.

Finally, the ~ operator takes just one operand. This operator reverses each bit in the operand. For instance, ~01 returns 10.

Table 8.5 shows more examples of using the bitwise operators in decimal, hex, and binary formats (in the left three columns). The corresponding results, in binary, hex, and decimal formats, are listed in the right three columns. The hex numbers are prefixed with 0x.

Table 8.5. Examples of using bitwise operators.

	Decimal		Results		
Expressions	Hex	Binary	Decimal	Hex	Binary
12 & 10	0x0C & 0x0A	1100 &1010	8	0x08	1000
12 | 10	0x0C | 0x0A	1100 | 1010	14	0x0E	1110
12 ^ 10	0x0C ^ 0x0A	1100 ^ 1010	6	0x06	0110
~12	~0x000C	~0000000000001100	65523	FFF3	1111111111110011

TYPE
Note that the complementary value of 12 is 65523, because the unsigned integer data type (16-bit) has the maximum number 65535. In other words, 65,523 is the result of subtracting 12 from 65,535. (The unsigned data modifier is introduced in Hour 9, "Playing with Data Modifiers and Math Functions.")

The program in Listing 8.5 demonstrates the usage of the bitwise operators.

Listing 8.5. Using bitwise operators.

```
1:  /* 08L05.c: Using bitwise operators */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int   x, y, z;
7:
8:     x = 4321;
9:     y = 5678;
10:    printf("Given x = %u, i.e., 0X%04X\n", x, x);
11:    printf("      y = %u, i.e., 0X%04X\n", y, y);
12:    z = x & y;
13:    printf("x & y  returns: %6u, i.e., 0X%04X\n", z, z);
14:    z = x | y;
15:    printf("x | y  returns: %6u, i.e., 0X%04X\n", z, z);
16:    z = x ^ y;
17:    printf("x ^ y  returns: %6u, i.e., 0X%04X\n", z, z);
18:    printf("  ~x   returns: %6u, i.e., 0X%04X\n", ~x, ~x);
19:    return 0;
20: }
```

OUTPUT
After the executable, 08L05.exe, is created and run from a DOS prompt, the following output is shown on the screen:

```
C:\app> 08L05
Given x = 4321, i.e., 0X10E1
      y = 5678, i.e., 0X162E
x & y  returns:  4128, i.e., 0X1020
x | y  returns:  5871, i.e., 0X16EF
     x ^ y  returns:  1743, i.e., 0X06CF
       ~x   returns: 61214, i.e., 0XEF1E
     C:\app>
```

ANALYSIS
In Listing 8.5, three integer variables, x, y, and z, are declared in line 6. Lines 8 and 9 set x and y to 4321 and 5678, respectively. Lines 10 and 11 then print out the values of x and y in both decimal and hex formats. The hex numbers are

prefixed with 0X.

The statement in line 12 assigns the result of the operation made by the bitwise AND operator (&) with the variables x and y. Then, line 13 displays the result in both decimal and hex formats.

Lines 14 and 15 perform the operation specified by the bitwise operator (|) and print out the result in both decimal and hex formats. Similarly, lines 16 and 17 give the result of the operation made by the bitwise XOR operator (^).

Last, the statement in line 18 prints out the complementary value of x by using the bitwise complement operator (~). The result is displayed on the screen in both decimal and hex formats.

Note that the unsigned integer format specifier with a minimum field width of 6, %6u, and the uppercase hex format specifier with the minimum width of 4, %04X, are used in the printf() function. The unsigned integer data type (that is, the non-negative integer data type) is chosen so that the complementary value of an integer can be shown and understood easily. More details on the unsigned data modifier are introduced in Hour 9.

WARNING

Don't confuse the bitwise operators & and | with the logical operators && and ||. For instance,

```
(x=1) & (y=10)

(x=1) && (y=10)
```

Using Shift Operators

There are two shift operators in C. The >> operator shifts the bits of an operand to the right; the << operator shifts the bits to the left.

The general forms of the two shift operators are

```
x >> y
x << y
```

Here x is an operand that is going to be shifted. y contains the specified number of places to shift.

For instance, the 8 >> 2 expression tells the computer to shift 2 bits of the operand 8 to the right, which returns 2 in decimal. The following:

$8 >> 2 \rightarrow (1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 0 * 2^0) >> 2$

produces the following:

$(0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0) \rightarrow 0010$ (binary) -> 2 (decimal).

Likewise, the 5 << 1 expression shifts 1 bit of the operand 5, and yields 10 in decimal.

The program in Listing 8.6 prints out more results by using the shift operators.

TYPE
Listing 8.6. Using the shift operators.

```
1:  /* 08L06.c: Using shift operators */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int   x, y, z;
7:
8:     x = 255;
9:     y = 5;
10:    printf("Given x = %4d, i.e., 0X%04X\n", x, x);
11:    printf("      y = %4d, i.e., 0X%04X\n", y, y);
12:    z = x >> y;
13:    printf("x >> y  returns: %6d, i.e., 0X%04X\n", z, z);
14:    z = x << y;
15:    printf("x << y  returns: %6d, i.e., 0X%04X\n", z, z);
16:    return 0;
17: }
```

OUTPUT

The following output is obtained by running the executable, 08L06.exe, from a DOS prompt:

```
C:\app> 08L06
Given x =  255, i.e., 0X00FF
      y =    5, i.e., 0X0005
x >> y  returns:      7, i.e., 0X0007
x << y  returns:   8160, i.e., 0X1FE0
C:\app>
```

ANALYSIS

Three integer variables, x, y, and z, are declared in line 6 of Listing 8.6. x is initial-ized with

255 in line 8; y is set to 5 in line 9. Then, lines 10 and 11 display the values of x and y on the screen.

The statement in line 12 shifts y bits of the operand x to the right, and then assigns the result to z. Line 13 prints out the result of the shifting made in line 12. The result is 7 in decimal, or 0X0007 in hex.

Lines 14 and 15 shift the operand x to the left by y bits and display the result on the screen, too. The result of the left-shifting is 8160 in

decimal, or 0x1FE0 in hex.

TIP

The operation of the shift-right operator (>>) is equivalent to dividing by powers of 2. In other words, the following:

```
x >> y
```

```
x / 2^y
```

Here x is a non-negative integer.
On the other hand, shifting to the left is equivalent to multiplying by powers of 2; that is,

```
x << y
```

```
x * 2^y
```

What Does x?y:z Mean?

In C, ?: is called the conditional operator, which is the only operator that takes three operands. The general form of the conditional operator is

```
x ? y : z
```

Here x, y, and z are three operands. Among them, x contains the test condition, and y and z represent the final value of the expression. If x returns nonzero (that is, TRUE), y is chosen; otherwise, z is the result.

For instance, the expression x > 0 ? `T' : `F' returns `T' if the value of x is greater than 0. Otherwise, the expression returns `F'.

Listing 8.7 demonstrates the usage of the conditional operator in the C language.

TYPE
Listing 8.7. Using the conditional operator.

```
1:  /* 08L07.c: Using the ?: operator */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int   x;
7:
8:     x = sizeof(int);
9:     printf("%s\n",
10:       (x == 2) ? "The int data type has 2 bytes." : "int doesn't have 2
                     Âbytes.");
11:    printf("The maximum value of int is: %d\n",
12:       (x != 2) ? ~(1 << x * 8 - 1) : ~(1 << 15) );
13:    return 0;
14: }
```

OUTPUT
I get the following output shown on the screen when I run the executable 08L07.exe from a DOS prompt on my machine:

```
C:\app> 08L07
The int data type has 2 bytes.
The maximum value of int is: 32767
C:\app>
```

ANALYSIS
In Listing 8.7, the size of the int data type is measured first in line 8, by using the sizeof operator and the number of bytes assigned to the integer variable x.

Lines 9 and 10 contain one statement, in which the conditional operator (?:) is used to test whether the number of bytes saved in x is equal to 2. (Here you see another example that a single statement can span multiple lines.) If the x == 2 expression returns nonzero (that is, TRUE), the string of The int data type has 2 bytes. is printed out by the printf() function in the statement. Otherwise, the second string, int doesn't have 2 bytes., is displayed on the screen.

In addition, the statement in lines 11 and 12 tries to find out the maximum value of the int data type on the current machine. The x != 2 expression is evaluated first in the statement. If the expression returns nonzero (that is, the byte number of the int data type is not equal to 2), the ~(<< x * 8 - 1) expression is evaluated, and the result is chosen as the return
value. Here the ~(1 << x * 8 - 1) expression is a general form to calculate the maximum value of the int data type, which is equivalent to 2 ** (x * 8 - 1) - 1. (The complement operator, ~, and the shift operator, <<, were introduced in the previous sections of this hour.)

On the other hand, if the test condition x != 2 in line 12 returns 0, which means the value of x is indeed equal to 2, the result of the ~(1 << 15) expression is chosen. Here you may have already figured out that ~(1 << 15) is equivalent to 2^{15}_1, which is the maximum value that the 16-bit int data type can have.

The result displayed on the screen shows that the int data type on my machine is 2 bytes
(or 16 bits) long, and the maximum value of the int data type is 32767.

Summary

In this lesson you've learned the following:

- The sizeof operator returns the number of bytes that a specified data type can have. You can use the operator to measure the size of a

data type on your machine.

- The logical AND operator (&&) returns 1 only if both its two operands (that is, expressions) are TRUE. Otherwise, the operator returns 0.
- The logical OR operator (||) returns 0 only if both its two operands are FALSE. Otherwise, the operator returns 1.
- The logical negation operator (!) reverses the logical value of its operand.
- There are six bit-manipulation operators: the bitwise AND operator (&), the bitwise OR operator (|), the bitwise XOR operator (^), the bitwise complement operator (~), the right-shift operator (>>), and the left-shift operator (<<).
- The conditional operator (?:) is the only operator in C that can take three operands.

In next lesson you'll learn about the data type modifiers in the C language.

Q&A

Q Why do we need the sizeof operator?

A The sizeof operator can be used to measure the sizes of all data types defined in C. When you write a portable C program that needs to know the size of an integer variable, it's a bad idea to hard-code the size as 16 or 32. The better way to tell the program the size of the variable is to use the sizeof operator, which returns the size of the integer variable at runtime.

Q What's the difference between | and ||?

A | is the bitwise OR operator that takes two operands. The | operator compares each bit of one operand to the corresponding bit in another operand. If both bits are 0, 0 is placed at the same position of the bit in the result. Otherwise, 1 is placed in the result.

On the other hand, ||, the logical OR operator, requires two operands (or expressions). The operator returns 0 (that is, FALSE) if both its operands are false. Otherwise, a nonzero value (that is, TRUE) is returned by the || operator.

Q Why is 1 << 3 equivalent to 1 * 2^3?

A The 1 << 3 expression tells the computer to shift 3 bits of the operand 1 to the left. The binary format of the operand is 0001. (Note that only the lowest four bits are shown here.) After being shifted 3 bits to the left, the expression returns 1000, which is equivalent to 1 * 2^3+0 * 2^2+0 * 2^1+0 * 2^0; that is, 1 * 2^3.

Q What can the conditional operator (?:) do?

A If there are two possible answers under certain conditions, you can use the ?: operator to pick up one of the two answers based on the result made by testing the conditions. For instance, the expression (age > 65) ? "Retired" : "Not retired" tells the computer that if the value of age is greater than 65, the string of Retired should be chosen; otherwise, Not retired is chosen.

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. What do the (x=1) && (y=10) and (x=1) & (y=10) expressions return, respectively?
2. Given x = 96, y = 1, and z = 69, what does the expression !y ? x == z : y return?
3. What do the ~0011000000111001 and ~1100111111000110 expressions return?
4. Given x=9, what does (x%2==0)||(x%3==0) return? How about (x%2==0)&&(x%3==0)?
5. Is 8 >> 3 equivalent to 8 / 2^3? How about 1 << 3?

Exercises

1. Given x = 0xEFFF and y = 0x1000, what do ~x and ~y return, respectively, in the hex format?
2. Taking the values of x and y assigned in exercise 1, write a program that prints out the return values of !x and !y by using both the %d and %u formats in the printf() function.

3. Given x = 123 and y = 4, write a program that displays the results of the x << y and x >> y expressions.
4. Write a program that shows the return values (in hex) of the 0xFFFF^0x8888, 0xABCD & 0x4567, and 0xDCBA | 0x1234 expressions.
5. Use the ?: operator and the for statement to write a program that keeps taking the characters entered by the user until the character q is accounted. (Hint: Put the x!='q' ? 1 : 0 expression to the second field in the for statement.)