Sams Teach Yourself C in 24 Hours

Hour 9 - Playing with Data Modifiers and Math Functions

If at first you don't succeed, transform your data.

—Murphy's Laws of Computers

In Hour 4, "Data Types and Names in C," you learned about several data types, such as char, int, float, and double, in the C language. In this hour, you'll learn about four data modifiers that enable you to have greater control over the data. The C keywords for the four data modifiers are

- signed
- unsigned
- short
- long

You're also going to learn about several mathematical functions provided by the C language, such as

- The sin() function
- The cos() function
- The tan() function
- The pow() function
- The sqrt() function

Enabling or Disabling the Sign Bit

As you know, it's very easy to express a negative number in decimal. All you need to do is put a minus sign in front of the absolute value of the number. But how does the computer represent a negative number in the binary format?

Normally, one bit can be used to indicate whether the value of a number represented in the binary format is negative. This bit is called the sign bit. The following two sections introduce two data modifiers, signed and unsigned, that can be used to enable or disable the sign bit.

The signed Modifier

For integers, the leftmost bit can be used as the sign bit. For instance, if the int data type is 16 bits long and the rightmost bit is counted as bit 0, you can use bit 15 as a sign bit. When the sign bit is set to 1, the C compiler knows that the value represented by the data variable is negative.

There are several ways to represent a negative value of the float or double data types. The implementations of the float and double data types are beyond the scope of this book. You can refer to Kernighan and Ritchie's book The C Programming Language for more details on the implementations of negative values of the float or double type.

The C language provides a data modifier, signed, that can be used to indicate to the compiler that the int or char data type uses the sign bit. By default, the int data type is a signed quantity. But the ANSI standard does not require the char data type be signed; it's up to the compiler vendors. Therefore, if you want to use a signed character variable, and make sure the compiler knows it, you can declare the character variable like this:

```
signed char ch;
```

so that the compiler knows that the character variable ch is signed, which means the variable can take a value in the range of -128 (that is, -2^7) to 127 (that is, 2^7_1).

(Remember that for an unsigned character variable, the range is 0 to 255; that is, 2^8_1.)

TIP

> To represent a negative number in the binary format, you can first get its equivalent positive value's binary format. Then you perform the complement operation on the binary, and finally, add one to the complemented binary.
> For instance, given a negative integer -12345, how can you represent it in the binary format?
> First, you need to find the binary format for the positive integer 12345, which is 0011000000111001.
> Then, you perform the complement operation on 0011000000111001; that is, ~0011000000111001, and obtain the following:
>
> ```
> 1100111111000110
> ```
>
> And finally, adding 1 to 1100111111000110 gives you 1100111111000111, which is the binary format of the negative integer -12345.

The unsigned Modifier

The C language also gives you the unsigned modifier, which can be used to tell the C compiler that no sign bit is needed in the specified data type.

Like the signed modifier, the unsigned modifier is meaningful only to the int and char data types.

For instance, the declaration

```
unsigned int x;
```

tells the C compiler that the integer variable x can only assume positive values from 0 to 65535 (that is, 2^{16}_1), if the int data type is 16 bits long.

In fact, unsigned int is equivalent to unsigned according to the ANSI standard. In other words, unsigned int x; is the same as unsigned x;.

Also, the ANSI standard allows you to indicate that a constant is of type unsigned by suffixing u or U to the constant. For instance,

```
unsigned int x, y;
x = 12345U;
y = 0xABCDu;
```

Here, the unsigned integer constants 12345U and 0xABCDu are assigned to variables x and y, respectively.

The program in Listing 9.1 is an example of using the signed and unsigned modifiers.

TYPE
Listing 9.1. Modifying data with signed and unsigned.

```
1:  /* 09L01.c: Using signed and unsigned modifiers */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     signed char  ch;
7:     int          x;
8:     unsigned int y;
9:
10:    ch = 0xFF;
11:    x = 0xFFFF;
12:    y = 0xFFFFu;
13:    printf("The decimal of signed 0xFF is %d.\n", ch);
14:    printf("The decimal of signed 0xFFFF is %d.\n", x);
15:    printf("The decimal of unsigned 0xFFFFu is %u.\n", y);
16:    printf("The hex of decimal 12345 is 0x%X.\n", 12345);
17:    printf("The hex of decimal -12345 is 0x%X.\n", -12345);
18:    return 0;
19: }
```

On my machine, the executable file of the program in Listing 9.1 is named 09L01.exe. (Note that when you compile the program in Listing 9.1, you'll see a warning message regarding the assignment statement ch = 0xFF; in line 10, due to the fact that ch is declared as a signed char variable. You can ignore the warning message.)

The following is the output printed on the screen after I run the executable from a DOS prompt:

```
C:\app> 09L01
The decimal of signed 0xFF is -1
The decimal of signed 0xFFFF is -1.
The decimal of unsigned 0xFFFFu is 65535.
The hex of decimal 12345 is 0x3039.
The hex of decimal -12345 is 0xCFC7.
C:\app>
```

OUTPUT
As you see in Listing 9.1, line 6 declares a signed char variable, ch. The int variable x and the unsigned int variable y are declared in lines 7 and 8, respectively. The three variables, ch, x, and y, are initialized in lines 10_12. Note that in line 12, u is suffixed to 0xFFFF to indicate that the constant is an unsigned integer.

ANALYSIS
The statement in line 13 displays the decimal value of the signed char variable ch. The output on the screen shows that the corresponding decimal value of 0xFF is -1 for the signed char variable ch.

Lines 14 and 15 print out the decimal values of the int variable x (which is signed by default) and the unsigned int variable y, respectively. Note that for the variable y, the unsigned integer format specifier %u is used in the printf() function in line 15. (Actually, you might recall that %u was used to specify the unsigned int data type as the display format in the previous hour.)

Based on the output, you see that 0xFFFF is equal to -1 for the signed int data type, and 65535 for the unsigned int data type. Here, the integer data type is 16 bits long.

Lines 16 and 17 print out 0x3039 and 0xCFC7, which are the hex formats of the decimal values of 12345 and -12345, respectively. According to the method mentioned in the last section, 0xCFC7 is obtained by adding 1 to the complemented value of 0x3039.

Changing Data Sizes

Sometimes, you want to reduce the memory taken by variables, or you need to increase the storage space of certain data types. Fortunately, the C language gives you the flexibility to modify sizes of data types. The two data modifiers, short and long, are introduced in the following two sections.

The short Modifier

A data type can be modified to take less memory by using the short modifier. For instance, you can apply the short modifier to an integer variable that is 32 bits long, which might reduce the memory taken by the variable to as little as 16 bits.

You can use the short modifier like this:

```
short x;
```

```
unsigned short y;
```

By default, a short int data type is a signed number. Therefore, in the short x; statement, x is a signed variable of short integer.

The long Modifier

If you need more memory to keep values from a wider range, you can use the long modifier to define a data type with increased storage space.

For instance, given an integer variable x that is 16 bits long, the declaration

```
long int x;
```

increases the size of x to 32 bits. In other words, after the modification, x is capable of holding a range of values from -2147483648 (that is, -2^{31}) to 2147483647 (that is, 2^{31}_1).

The ANSI standard allows you to indicate that a constant has type long by suffixing l or L to the constant. For instance:

```
long int x, y;
x = 1234567891;

y = 0xABCD1234L;
```

Here, the constants of the long int data type, 1234567891 and 0xABCD1234L, are assigned to variables x and y, respectively.

Also, you can declare a long integer variable simply like this:

```
long x;
long int x;
```

Listing 9.2 contains a program that can print out the numbers of bytes for different modified data types.

TYPE
Listing 9.2. Modifying data with short and long.

```
1:  /* 09L02.c: Using short and long modifiers */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     printf("The size of short int is: %d.\n",
7:         sizeof(short int));
8:     printf("The size of long int is: %d.\n",
9:         sizeof(long int));
10:    printf("The size of float is: %d.\n",
11:        sizeof(float));
12:    printf("The size of double is: %d.\n",
13:        sizeof(double));
14:    printf("The size of long double is: %d.\n",
15:        sizeof(long double));
16:    return 0;
17: }
```

I obtain the following output printed on the screen after I run the executable 09L02.exe from a DOS prompt:

```
C:\app> 09L02
The size of short int is: 2.
The size of long int is: 4.
The size of float is: 4.
The size of double is: 8.
The size of long double is: 10.
C:\app>
```

OUTPUT
In Listing 9.2, the sizeof operator and printf() function are used to measure the sizes of the modified data types and display the results on the screen.

For instance, lines 6 and 7 obtain the size of the short int data type and print out the number of the byte, 2, on the screen. From the output, you know that the short int data type is 16 bits (that is, 2 bytes) long on my machine.

ANALYSIS
Likewise, lines 8 and 9 find the size of the long int data type is 4 bytes (that is, 32 bits) long, which is the same length as the float data type obtained in lines 10 and 11.

Lines 12 and 13 obtain the size of the double data type, which is 8 bytes (that is, 64 bits) on my machine. Then, after being modified by the long modifier, the size of the double data type is increased to 10 bytes (that is, 80 bits), which is printed out by the printf() function in lines 14 and 15.

Adding h, l, or L to Format Specifiers

You can add h into the integer format specifier (like this: %hd, %hi, or %hu) to specify that the corresponding number is a short int or unsigned short int.

On the other hand, using %ld or %Ld specifies that the corresponding datum is long int. %lu or %Lu is then used for the long unsigned int data.

The program in Listing 9.3 shows the usage of %hd, %lu, and %ld.

TYPE
Listing 9.3. Using %hd, %ld, and %lu.

```
1:  /* 09L03.c: Using %hd, %ld, and %lu specifiers */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     short int         x;
7:     unsigned int      y;
8:     long int          s;
9:     unsigned long int t;
10:
11:    x = 0xFFFF;
12:    y = 0xFFFFU;
13:    s = 0xFFFFFFFFl;
14:    t = 0xFFFFFFFFL;
15:    printf("The short int of 0xFFFF is %hd.\n", x);
16:    printf("The unsigned int of 0xFFFF is %u.\n", y);
17:    printf("The long int of 0xFFFFFFFF is %ld.\n", s);
18:    printf("The unsigned long int of 0xFFFFFFFF is %lu.\n", t);
19:    return 0;
20: }
```

After the executable 09L03.exe is created and run from a DOS prompt, the following output is shown on the screen:

```
C:\app> 09L03
The short int of 0xFFFF is -1.
The unsigned int of 0xFFFF is 65535.
The long int of 0xFFFFFFFF is -1.
The unsigned long int of 0xFFFFFFFF is 4294967295
C:\app>
```

OUTPUT
There are four data types declared in Listing 9.3: the short int variable x, the unsigned int variable y, the long int variable s, and the unsigned long int variable t. The four variables are initialized in lines 6_9.

ANALYSIS
To display the decimal values of x, y, s, and t, the format specifiers %hd, %u, %ld, and %lu are used, respectively, in lines 15_18 to convert the corresponding hex numbers to decimal numbers. The output from the program in Listing 9.3 shows that values contained by x, y, s, and t have been correctly displayed on the screen.

Mathematical Functions in C

Basically, the math functions provided by the C language can be classified into three groups:

- Trigonometric and hyperbolic functions, such as acos(), cos(), and cosh().
- Exponential and logarithmic functions, such as exp(), pow(), and log10().
- Miscellaneous math functions, such as ceil(), fabs(), and floor().

You have to include the header file math.h in your C program before you can use any math functions defined in the header file. Appendix B, "ANSI C Library Functions," lists all the math functions available in C.

The following two sections introduce several math functions and tell you how to use them in your programs.

Calling sin(), cos(), and tan()

You should appreciate that C gives you a set of functions to deal with trigonometric or hyperbolic calculations, if you think those calculations are very tough.

For instance, given an angle x in radians, the sin(x) expression returns the sine of the angle.

The following formula can be used to convert the value of an angle in degrees into the value in radians:

```
radians = degree * (3.141593 / 180.0).
```

Here, 3.141593 is the approximate value of pi. If needed, you can use more decimal digits from pi.

Now, let's have a look at the syntax of the sin(), cos(), and tan() functions.

The syntax for the sin() function is

```
#include <math.h>
double sin(double x);
```

Here, the double variable x contains the value of an angle in radians. The sin() function returns the sine of x in the double data type.

The syntax for the cos() function is

```
#include <math.h>
double cos(double x);
```

Here, the double variable x contains the value of an angle in radians. The cos() function returns the cosine of x in the double data type.

The syntax for the tan() function is

```
#include <math.h>
double tan(double x);
```

Here, the double variable x contains the value of an angle in radians. The tan() function returns the tangent of x in the double data type.

Listing 9.4 demonstrates how to use the sin(), cos(), and tan() functions.

TYPE
Listing 9.4. Calculating trigonometric values with sin(), cos(), and tan().

```
1:  /* 09L04.c: Using sin(), cos(), and tan() functions */
2:  #include <stdio.h>
3:  #include <math.h>
4:
5:  main()
6:  {
7:     double x;
8:
9:     x = 45.0;                  /* 45 degree */
10:    x *= 3.141593 / 180.0;     /* convert to radians */
11:    printf("The sine of 45 is:    %f.\n", sin(x));
12:    printf("The cosine of 45 is:  %f.\n", cos(x));
13:    printf("The tangent of 45 is: %f.\n", tan(x));
14:    return 0;
15: }
```

The following output is displayed on the screen when the executable 09L04.exe is executed:

```
C:\app> 09L04
The sine of 45 is:    0.707107.
The cosine of 45 is:  0.707107.
The tangent of 45 is: 1.000000.
C:\app>
```

OUTPUT
Note that the header file math.h is included in line 3, which is required by the C math functions.

ANALYSIS
The double variable x in Listing 9.4 is initialized with 45.0 in line 9. Here, 45.0 is the value of the angle in degrees, which is converted into the corresponding radians in line 10.

Then, the statement in line 11 calculates the sine of x by calling the sin() function and prints out the result on the screen. Similarly, line 12 obtains the cosine of x and shows it on the screen as well. Because x contains the value of a 45-degree angle, it's not surprising to see that both the sine and cosine values are the same, about 0.707107.

Line 13 gives the tangent value of x by using the tan() function. As you might know, the tangent of x is equal to the sine of x divided by the cosine of x. Because the sine of a 45-degree angle is the same as the cosine of a 45-degree angle, the tangent of a 45-degree angle is equal to 1. The result (in the floating-point format) of 1.000000, in the third line of the listing's output, proves it.

You can declare a constant PI initialized to 3.141593, and another constant initialized to 180.0. Or, simply declare a single constant initialized to the result of 3.141593/180.0. In Hour 23, "The C Preprocessor," you'll learn to use the C preprocessor #define directive to do so.

Calling pow() and sqrt()

The pow() and sqrt() functions are two other useful math functions in C.

The syntax for the pow() function is

```
#include <math.h>
double pow(double x, double y);
```

Here, the value of the double variable x is raised to the power of y. The pow() function returns the result in the double data type.

The syntax for the sqrt() function is

```
#include <math.h>
double sqrt(double x);
```

Here, the sqrt() function returns the non-negative square root of x in the double data type. The function returns an error if x is negative.

In fact, if you set up the second argument in the pow() function to 0.5, and x contains a non-negative value, the two expressions, pow(x, 0.5) and sqrt(x), are equivalent.

Now, take a look at how to call the pow() and sqrt() functions in the program shown in Listing 9.5.

TYPE
Listing 9.5. Applying the pow() and sqrt() functions.

```
1:  /* 09L05.c: Using pow() and sqrt() functions */
2:  #include <stdio.h>
3:  #include <math.h>
4:
5:  main()
6:  {
7:     double x, y, z;
8:
```

```
9:     x = 64.0;
10:    y = 3.0;
11:    z = 0.5;
12:    printf("pow(64.0, 3.0) returns: %7.0f\n", pow(x, y));
13:    printf("sqrt(64.0) returns:     %2.0f\n", sqrt(x));
14:    printf("pow(64.0, 0.5) returns: %2.0f\n", pow(x, z));
15:    return 0;
16: }
```

The following output is displayed on the screen after the executable 09L05.exe is executed:

```
C:\app> 09L05
pow(64.0, 3.0) returns: 262144
sqrt(64.0) returns:     8
pow(64.0, 0.5) returns: 8
C:\app>
```

OUTPUT

The three double variables in Listing 9.5, x, y, and z, are initialized with 64.0, 3.0, and 0.5, respectively, in lines 9_11.

ANALYSIS

The pow() function in line 12 takes x and y and then calculates the value of x raised to the power of y. Because the fractional part is all decimal digits of 0s, the format specifier %7.0f is used in the printf() function to convert only the non-fractional part of the value. The result is shown on the screen as 262144.

In line 13, the non-negative square root of x is calculated by calling the sqrt() function. As in line 12, the format specifier %2.0f is used in line 13 to convert the non-fractional part of the value returned from the sqrt() function, because the fractional part consists of decimal digits of 0s. As you see in the output, the non-negative square root of x is 8.

As I mentioned earlier, the pow(x, 0.5) expression is equivalent to the sqrt(x) expression. Thus, it's no surprise to see that pow(x, z) in the statement of line 14 produces the same result as sqrt(x) does in line 13.

NOTE

All floating-point calculations, including both the float and double data types, are done in double-precision arithmetic. That is, a float data variable must be converted to a double in order to carry on the calculation. After the calculation, the double has to be converted back to a float before the result can be assigned to the float variable. Therefore, a float calculation may take more time.

The main reason that C supports the float data type is to save memory space, because the double data type takes twice as much memory space for storage as the float data type does.

Summary

In this lesson you've learned the following:

- The signed modifier can be used to enable the sign bit for the char and int data types.
- All int variables in C are signed by default.
- The unsigned modifier can be used to disable the sign bit for the char and int data types.
- The memory space taken by a data variable can be reduced or increased by using the short, or long, data modifier respectively.
- There is a set of C library functions, such as sin(), cos(), and tan(), that can be used to perform trigonometric or hyperbolic computations.
- There is another group of math functions in C—for example, pow()—that can perform exponential and logarithmic calculation.
- The sqrt() function returns a non-negative square root. The expression sqrt(x) is equivalent to the pow(x, 0.5) expression, if x has a non-negative value.
- The header file math.h must be included in your C program if you call some math functions defined in the header file.

In the next lesson you'll learn several very important control flow statements in C.

Q&A

Q Which bit can be used as the sign bit in an integer?

A The leftmost bit can be used as the sign bit for an integer. For instance, assume the int data type is 16 bits long. If you count the bit position from right to left, and the first bit counted is bit 0, then bit 15 is the leftmost bit that can be used as the sign bit.

Q What can the %lu format specifier do?

A The %lu format specifier can be used to convert the corresponding datum to the unsigned long int data type. In addition, the %lu format specifier is equivalent to %Lu.

Q When do I use short and long?

A If you need to save memory space, and you know the value of an integer data variable stays within a smaller range, you can try to use the short modifier to tell the C compiler to reduce the default memory space assigned to the variable, for instance, from 32 bits to 16 bits.

On the other hand, if a variable has to hold a number that is beyond the current range of a data type, you can use the long modifier to increase the storage space of the variable in order to hold the number.

Q Does the sin() function take a value in degrees or in radians?

A Like other trigonometric math functions in C, the sin() function takes a value in radians. If you have an angle in degrees, you have to convert it into the form of radians. The formula is:

```
radians = degree * (3.141593 / 180.0).
```

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. Given an int variable x and an unsigned int variable y, as well as x = 0x8765 and y = 0x8765, is x equal to y?
2. Given that the int data type is 16 bits long, what is the hex format of the decimal number -23456?
3. Which format specifier, %ld or %lu, should be used to specify an unsigned long int variable?
4. What is the name of the header file you have to include if you're calling some C math functions from your C program?

Exercises

1. Given the following statements,

   ```
   int x;
   unsigned int y;
   x = 0xAB78;
   y = 0xAB78;
   ```

   write a program to display the decimal values of x and y on the screen.

2. Write a program to measure the sizes of short int, long int, and long double on your machine.
3. Give the binary representations of the following:
   - 512
   - -1
   - 128
   - -128
4. Write a program to display the decimal value given in quiz question 2 in the hex format. Does the result from the program match your answer to that question?
5. Given an angle of 30 degrees, write a program to calculate its sine and tangent values.
6. Write a program to calculate the non-negative square root of 0x19A1.