Sams Teach Yourself C in 24 Hours

Hour 3 - The Essentials of C Programs

The whole is equal to the sum of its parts.

—Euclid

In Hour 2, "Writing Your First C Program," you saw and wrote some simple C programs. You also learned about the basic structure of a C program. You know that a program written in C has to be compiled before it can be executed. In this lesson you'll learn more essentials within a C program, such as

- Constants and variables
- Expressions
- Statements
- Statement blocks
- C function types and names
- Arguments to functions
- The body of a function
- Function calls

The Basics of the C Program

As a building is made of bricks, a C program is made of basic elements, such as expressions, statements, statement blocks, and function blocks. These elements are discussed in the following sections. But first, you need to learn two smaller but important elements, constant and variable, which make up expressions.

Constants and Variables

As its name implies, a constant is a value that never changes. A variable, on the other hand, can be used to present different values.

You can think of a constant as a music CD-ROM; the music saved in the CD-ROM is never changed. A variable is more like an audio cassette: You can always update the contents of the cassette by simply overwriting the old songs with new ones.

You can see many examples in which constants and variables are in the same statement. For instance, consider the following:

```
i = 1;
```

where the symbol 1 is a constant because it always has the same value (1), and the symbol i is assigned the constant 1. In other words, i contains the value of 1 after the statement is executed. Later, if there is another statement,

```
i = 10;
```

after it is executed, i is assigned the value of 10. Because i can contain different values, it's called a variable in the C language.

Expressions

An expression is a combination of constants, variables, and operators that are used to denote computations.

For instance, the following:

```
(2 + 3) * 10
```

is an expression that adds 2 and 3 first, and then multiplies the result of the addition by 10. (The final result of the expression is 50.)

Similarly, the expression 10 * (4 + 5) yields 90. The 80/4 expression results in 20.

Here are some other examples of expressions:

Expression	Description
6	An expression of a constant.
i	An expression of a variable.
6 + i	An expression of a constant plus a variable.
exit(0)	An expression of a function call.

Arithmetic Operators

As you've seen, an expression can contain symbols such as +, *, and /. In the C language, these symbols are called arithmetic operators. Table 3.1 lists all the arithmetic operators and their meanings.

Table 3.1. C arithmetic operators.

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication

/	Division
%	Remainder (or modulus)

You may already be familiar with all the arithmetic operators, except the remainder (%) operator. % is used to obtain the remainder of the first operand divided by the second operand. For instance, the expression

```
6 % 4
```

yields a value of 2 because 4 goes into 6 once with a remainder of 2.

The remainder operator, %, is also called the modulus operator.

Among the arithmetic operators, the multiplication, division, and remainder operators have a higher precedence than the addition and subtraction operators. For example, the expression

```
2 + 3 * 10
```

yields 32, not 50. Because of the higher precedence of the multiplication operator, 3 * 10 is calculated first, and then 2 is added into the result of the multiplication.

As you might know, you can put parentheses around an addition (or subtraction) to force the addition (or subtraction) to be performed before a multiplication, division, or modulus computation. For instance, the expression

```
(2 + 3) * 10
```

performs the addition of 2 and 3 first before it does the multiplication of 10.

You'll learn more operators of the C language in Hours 6, "Manipulating Data with Operators," and 8, "More Operators."

Statements

In the C language, a statement is a complete instruction, ending with a semicolon. In many cases, you can turn an expression into a statement by simply adding a semicolon at the end of the expression.

For instance, the following

```
i = 1;
```

is a statement. You may have already figured out that the statement consists of an expression of i = 1 and a semicolon (;).

Here are some other examples of statements:

```
i = (2 + 3) * 10;
i = 2 + 3 * 10;
j = 6 % 4;
k = i + j;
```

Also, in the first lesson of this book you learned statements such as

```
return 0;
exit(0);
printf ("Howdy, neighbor! This is my first C program.\n");
```

Statement Blocks

A group of statements can form a statement block that starts with an opening brace ({) and ends with a closing brace (}). A statement block is treated as a single statement by the C compiler.

For instance, the following

```
for(. . .) {
   s3 = s1 + s2;
   mul = s3 * c;
   remainder = sum % c;
}
```

is a statement block that starts with { and ends with }. Here for is a keyword in C that determines the statement block. The for keyword is discussed in Hour 7, "Doing the Same Thing Over and Over."

A statement block provides a way to group one or more statements together as a single statement. Many C keywords can only control one statement. If you want to put more than one statement under the control of a C keyword, you can add those statements into a statement block so that the block is considered one statement by the C keyword.

Anatomy of a C Function

Functions are the building blocks of C programs. Besides the standard C library functions, you can also use some other functions made by you or another programmer in your C program. In Hour 2 you saw the main() function, as well as two C library functions, printf() and exit(). Now, let's have a closer look at functions.

As shown in Figure 3.1, a function consists of six parts: the function type, the function name, arguments to the function, the opening brace, the function body, and the closing

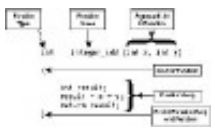

Figure 3.1. *The anatomy of a function in the C language. brace.*

The six parts of a function are explained in the following sections.

Determining a Function's Type

The function type is used to signify what type of value a function is going to return after its execution. In Hour 2, for instance, you learned that the default function type of main() is integer. You also learned how to change the function type of main() to void so that the main() function does need to return any value.

In C, int is used as the keyword for the integer data type. In the next hour, you'll learn more about data types.

Giving a Function a Valid Name

A function name is given in such a way that it reflects what the function can do. For

instance, the name of the printf() function means "print formatted data."

There are certain rules you have to follow to make a valid function name. The following are examples of illegal function names in C:

Illegal Name	The Rule
2 (digit)	A function name cannot start with a digit.
* (Asterisk)	A function name cannot start with an asterisk.
+ (Addition)	A function name cannot start with one of the arithmetic signs that are reserved C keywords.
. (dot)	A function name cannot start with ..
total-number	A function name cannot contain a minus sign.
account'97	A function name cannot contain an apostrophe.

Some samples of valid function names are as follows:

- print2copy
- total_number
- _quick_add
- Method3

Arguments to C Functions

You often need to pass a function some information before executing it. For example, in Listing 2.1 in Hour 2, a character string, "Howdy, neighbor! This is my first C program.\n", is passed to the printf() function, and then printf() prints the string on the screen.

Pieces of information passed to functions are known as arguments. The argument of a function is placed between the parentheses that immediately follow the function name.

The number of arguments to a function is determined by the task of the function. If a function needs more than one argument, arguments passed to the function must be separated by commas; these arguments are considered an argument list.

If no information needs to be passed to a function, you just leave the argument field between the parentheses blank. For instance, the main() function in Listing 2.1 of Hour 2 has no argument, so the field between the parentheses following the function name is empty.

The Beginning and End of a Function

As you may have already figured out, braces are used to mark the beginning and end of a function. The opening brace ({) signifies the start of a function body, while the closing brace (}) marks the end of the function body.

As mentioned earlier, the braces are also used to mark the beginning and end of a statement block. You can think of it as a natural extension to use braces with functions because a function body can contain several statements.

The Function Body

The function body in a function is the place that contains variable declarations and C statements. The task of a function is accomplished by executing the statements inside the function body one at a time.

Listing 3.1 demonstrates a function that adds two integers specified by its argument and returns the result of the addition.

TYPE
Listing 3.1. A function that adds two integers.

```
1:  /* This function adds two integers and returns the result */
2:  int integer_add( int x, int y )
3:  {
4:     int result;
5:     result = x + y;
6:     return result;
7:  }
```

ANALYSIS

As you learned in Hour 2, line 1 of Listing 3.1 is a comment that tells the program-mer what the function can do.

In line 2, you see that the int data type is prefixed prior to the function name. Here int is used as the function type, which signifies that an integer should be returned by the function. The function name shown in line 2 is integer_add. The argument list contains two arguments, int x and int y, in line 2, where the int data type specifies that the two arguments are both integers.

Line 4 contains the opening brace ({) that marks the start of the function.

The function body is in lines 4_6 in Listing 3.1. Line 4 gives the variable declaration of result, whose value is specified by the int data type as an integer. The statement in line 5 adds the two integers represented by x and y and assigns the computation result to the result variable. The return statement in line 6 then returns the computation result represented by result.

Last, but not least, the closing brace (}) in line 7 is used to close the function.

TIP

When you create a function in your C program, don't assign the function too much work. If a function has too much to do, it will be very difficult to write and debug. If you have a complex programming project, break it into smaller pieces. And try your best to make sure that each function has just one task to do.

Making Function Calls

Based on what you've learned so far, you can write a C program that calls the integer_add() function to calculate an addition and then print out the result on the screen. An example of such a program is demonstrated in Listing 3.2.

TYPE

Listing 3.2. A C program that calculates an addition and prints the result to the screen.

```
1:  /* 03L02.c: Calculate an addition and print out the result */
2:  #include <stdio.h>
3:  /* This function adds two integers and returns the result */
4:  int integer_add( int x, int y )
5:  {
6:     int result;
7:     result = x + y;
8:     return result;
9:  }
10:
11: int main()
12: {
13:    int sum;
14:
15:    sum = integer_add( 5, 12);
16:    printf("The addition of 5 and 12 is %d.\n", sum);
17:    return 0;
18: }
```

OUTPUT

The program in Listing 3.2 is saved as a source file called 03L02.c. After this program is compiled and linked, an executable file for 03L02.c is created. On my machine, the executable file is named 03L02.exe. The following is the output printed on the screen after I run the executable from a DOS prompt on my machine:

```
C:\app> 03L02
The addition of 5 and 12 is 17.
C:\app>
```

ANALYSIS

Line 1 in Listing 3.2 is a comment about the program. As you learned in Hour 2, the include directive in line 2 includes the stdio.h header file because of the printf() function in the program.

Lines 3_9 represent the integer_add() function that adds two integers, as discussed in the previous section.

The main() function, prefixed with the int data type, starts in line 11. Lines 12 and 18 contain the opening brace and closing brace for the main() function, respectively. An integer variable, sum, is declared in line 13.

The statement in line 15 calls the integer_add() function that we examined in the previous section. Note that two integer constants, 5 and 12, are passed to the integer_add() function, and that the sum variable is assigned the result returned from the integer_add() function.

You first saw the C standard library function printf() in Hour 2. Here you may find something new added to the function in line 16. You're right. This time, there are two arguments that are passed to the printf() function. They are the string "The addition of 5 and 12 is %d.\n" and the variable sum.

Note that a new symbol, %d, is added into the first argument. The second argument is the integer variable sum. Because the value of sum is going to be printed out on the screen, you might think that the %d has something to do with the integer variable sum. You're right again. %d tells the computer the format in which sum should be printed on the screen.

More details on %d are covered in Hour 4, "Data Types and Names in C." The relationship between %d and sum is discussed in Hour 5, "Reading from and Writing to Standard I/O."

More importantly, you should focus on the program in Listing 3.2 and pay attention to how to call either a user-generated function or a standard C library function from the main() function.

Summary

In this lesson you've learned the following:

- A constant in C is a value that never changes. A variable, on the other hand, can present different values.
- A combination of constants, variables, and operators is called an expression in the C language. An expression is used to denote different computations.
- The arithmetic operators include +, -, *, /, and %.
- A statement consists of a complete expression suffixed with a semicolon.
- The C compiler treats a statement block as a single statement, although the statement block may contain more than one statement.
- The function type of a function determines the type of the return value made by the function.
- You have to follow certain rules to make a valid function name.
- An argument contains information that you want to pass to a function. An argument list contains two or more arguments that are separated by commas.
- The opening brace ({) and closing brace (}) are used to mark the start and end of a C function.
- A function body contains variable declarations and statements. Usually, a function should accomplish just one task.

In the next lesson you'll learn more about data types in the C language.

Q&A

Q What is the difference between a constant and a variable?

A The major difference is that the value of a constant cannot be changed, while the value of a variable can. You can assign different values to a variable whenever it's necessary in your C program.

Q Why do you need a statement block?

A Many C keywords can only control one statement. A statement block provides a way to put more than one statement together and put the statement block under the control of a C keyword. Then, the statement block is treated as a single statement.

Q Which arithmetic operators have a higher precedence?

A Among the five arithmetic operators, the multiplication, division, and remainder operators have a higher precedence than the addition and subtraction operators.

Q How many parts does a function normally have?

A A function normally has six parts: the function type, the function name, the arguments, the opening brace, the function body, and the closing brace.

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. In the C language, is 74 a constant? How about 571?
2. Is x = 570 + 1 an expression? How about x = 12 + y?
3. Are the following function names valid?

```
2methods
m2_algorithm
*start_function
Room_Size
.End_Exe
_turbo_add
```

4. Is 2 + 5 * 2 equal to (2 + 5) * 2?
5. Does 7 % 2 produce the same result as 4 % 3?

Exercises

1. Given two statements, x = 3; and y = 5 + x;, how can you build a statement block with the two statements?
2. What is wrong with the following function?

```
int 3integer_add( int x, int y, int z)
{
   int sum;
   sum = x + y + z;
   return sum;
}
```

3. What is wrong with the following function?

```
int integer_add( int x, int y, int z)
{
   int sum;
   sum = x + y + z
   return sum;
}
```

4. Write a C function that can multiply two integers and return the calculated result.
5. Write a C program that calls the C function you just wrote in exercise 4 to calculate the multiplication of 3 times 5 and then print out the return value from the function on the screen.