Sams Teach Yourself C in 24 Hours

Hour 10 - Getting Controls

It is harder to command than to obey.

—F. Nietzsche

In Hour 7, "Doing the Same Thing Over and Over," you learned to use the for, while, and do-while statements to do the same things over and over. These three statements can be grouped into the looping category that is a part of the control flow statements in C.

In this lesson you'll learn about the statements that belong to another group of control flow statements—conditional branching (or jumping), such as

- The if statement
- The if-else statement
- The switch statement
- The break statement
- The continue statement
- The goto statement

Always Saying "if…"

If life were a straight line, it would be very boring. The same thing is true for programming. It would be too dull if the statements in your program could only be executed in the order in which they appear.

In fact, an important task of a program is to instruct the computer to branch (that is, jump) to different portions of the code and work on different jobs whenever the specified conditions are met.

However, in most cases, you don't know in advance what will come next. What you do know is that something is bound to happen if certain conditions are met. Therefore, you can just write down tasks and conditions in the program. The decisions of when to perform the tasks are made by the conditional branching statements.

In C, the if statement is the most popular conditional branching statement; it can be used to evaluate the conditions as well as to make the decision whether the block of code controlled by the statement is going to be executed.

The general form of the if statement is

```
if (expression) {
   statement1;
   statement2;
   .
   .
   .
}
```

Here expression is the conditional criterion. If expression is logical TRUE (that is, nonzero), the statements inside the braces ({ and }), such as statement1 and statement2, are executed. If expression is logical FALSE (zero), then the statements are skipped.

Note that the braces ({ and }) form a block of statements that is under the control of the if statement. If there is only one statement inside the block, the braces can be ignored. The parentheses ((and)), however, must always be used to enclose the conditional expression.

For instance, the following expression

```
if (x > 0)
  printf("The square root of x is: %f\n", sqrt(x));
```

tells the computer that if the value of x is greater than zero (that is, positive), it should calculate the square root of x by calling the sqrt() function, and then print out the result. Here the conditional criterion is the relational expression x > 0, which returns 1 for true and 0 for false.

Listing 10.1 gives you another example of using the if statement.

TYPE
Listing 10.1. Using the if statement in decision making.

```
1:  /* 10L01.c  Using the if statement */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int i;
7:
8:     printf("Integers that can be divided by both 2 and 3\n");
9:     printf("(within the range of 0 to 100):\n");
10:    for (i=0; i<=100; i++){
11:       if ((i%2 == 0) && (i%3 == 0))
12:         printf("   %d\n", i);
13:   }
14:   return 0;
15: }
```

OUTPUT

After 10L01.exe, the executable of the program in Listing 10.1, is created and run from a DOS prompt, the following output is printed on the screen:

```
C:\app> 10L01
Integers that can be divided by both 2 and 3
(within the range of 0 to 100):
   0
   6
   12
   18
   24
   30
   36
   42
   48
   54
   60
   66
   72
   78
   84
   90
   96
C:\app>
```

ANALYSIS

As you see in Listing 10.1, line 6 declares an integer variable, i. Lines 8 and 9 print out two headlines. Starting in line 10, the for statement keeps looping 101 times.

Within the for loop, the if statement in lines 11 and 12 evaluates the logical expression (i%2 == 0) && (i%3 == 0). If the expression returns 1 (that is, the value of i can be divided by both 2 and 3 completely), the value of i is displayed on the screen by calling the printf() function in line 12. Otherwise, the statement in line 12 is skipped.

Note that the braces ({ and }) are discarded in the if statement because there is only one statement under the control of the statement.

The result shown on the screen gives all integers within the range of 0 to 100 that can be divided by both 2 and 3.

The if-else Statement

In the if statement, when the conditional expression is logical TRUE, the computer will jump to the statements controlled by the if statement and execute them right away. If the expression is false, the computer will ignore those statements controlled by the if statement.

From time to time, you will want the computer to execute some other specified statements when the conditional expression of the if statement is logical FALSE. By doing so, you can use another conditional branching statement in C—the if-else statement.

As an expansion of the if statement, the if-else statement has the following form:

```
if (expression) {
   statement1;
   statement2;
   .
   .
   .
}
else {
   statement_A;
   statement_B;
   .
   .
   .
}
```

Here if expression is logical TRUE, the statements controlled by if, including statement1 and statement2, are executed. The statements, such as statement_A and statement_B, inside the statement block and following the else keyword are executed if expression is not logical TRUE.

The program in Listing 10.2 shows how to use the if-else statement.

TYPE
Listing 10.2. Using the if-else statement.

```
1:  /* 10L02.c  Using the if-else statement */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int i;
7:
8:     printf("Even Number   Odd Number\n");
9:     for (i=0; i<10; i++)

10:       if (i%2 == 0)
11:          printf("%d", i);
12:       else
13:          printf("%14d\n", i);
14:
15:    return 0;
16: }
```

OUTPUT

The following result is obtained by running the executable file 10L02.exe:

```
C:\app>10L02
Even Number   Odd Number
0             1
2             3
4             5
6             7
8             9
C:\app>
```

ANALYSIS
Line 6 of Listing 10.2 declares an integer variable, i. The printf() function in line 8 displays a headline on the screen.

The integer variable i is initialized in the first expression field of the for statement in line 9. Controlled by the for statement, the if-else statement in lines 10_13 is executed 10 times. According to the if-else statement, the printf() function in line 11 prints out even numbers if the relational expression i%2 == 0 in line 10 returns 1 (that is, TRUE). If the relational expression returns 0 (that is, FALSE), the printf() function controlled by the else keyword in line 12 outputs odd numbers to the standard output.

Because the if-else statement is treated as a single statement, the braces { and } are not needed to form a block of statement in the for statement. Likewise, there are no braces used in the if-else statement because the if and else keywords each control a single statement, respectively, in lines 11 and 13.

Note that the minimum width of 14 is specified in the printf() function in line 13, so the output of the odd numbers is listed to the right side of the even numbers, as you can see in the output section. Because the program in Listing 10.2 checks numbers in a range of 0 to 9, the output shows that 0, 2, 4, 6, and 8 are even numbers, and 1, 3, 5, 7, and 9 are odd ones.

Nested if Statements

As you saw in the previous sections, one if statement enables a program to make one decision. In many cases, a program has to make a series of decisions. To enable it to do so, you can use nested if statements.

Listing 10.3 demonstrates the usage of nested if statements.

TYPE
Listing 10.3. Using nested if statements.

```
1:  /* 10L03.c  Using nested if statements */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int i;
7:
8:     for (i=-5; i<=5; i++){
9:        if (i > 0)
10:           if (i%2 == 0)
11:              printf("%d is an even number.\n", i);
12:           else
13:              printf("%d is an odd number.\n", i);
14:       else if (i == 0)
15:           printf("The number is zero.\n");
16:       else
17:           printf("Negative number: %d\n", i);
18:    }
19:    return 0;
20: }
```

OUTPUT
After running the executable file 10L03.exe, I obtain the following output:

```
C:\app>10L03
Negative number: -5
Negative number: -4
Negative number: -3
Negative number: -2
Negative number: -1
The number is zero.
1 is an odd number.
2 is an even number.
3 is an odd number.
4 is an even number.
5 is an odd number.
C:\app>
```

ANALYSIS
Listing 10.3 contains a for loop, starting in line 8 and ending in line 18. According to the expressions of the for statement in line 8, any tasks controlled by the for statement are executed up to 11 times.

First, a decision has to be made based on the return value of the relational expression i > 0 in the if statement of line 9. The i > 0 expression is used to test whether the value of i is positive or negative (including zero.) If the return value is 1, the computer jumps to the second (that is, nested) if statement in line 10.

Note that line 10 contains another relational expression, i%2 == 0, which tests whether the integer variable i is even or odd. Therefore, the second decision of displaying even numbers or odd numbers has to be made according to the return value of the second relational expression, i%2 == 0. The printf() function in line 11 prints out an even number if the return value is 1. Otherwise, the statement in line 13 is executed, and an odd number is shown on the screen.

The computer branches to line 14 if the i > 0 expression returns 0; that is, if the value of i is not greater than 0. In line 14, another if statement is nested within an else phrase, and the relational expression i == 0 is evaluated. If i == 0 is true, which means i contains the value of zero, the string of The number is zero. is displayed on the screen. Otherwise, the value of i is negative, according to the value returned by the i > 0 expression. The statement in line 17 then outputs the negative number to the standard output.

As you can see in the example, the value of i is within the range of 5 to -5. Thus, -5, -4, -3, -2, and -1 are printed out as negative numbers. In addition, the odd numbers 1, 3, and 5, as well as the even numbers 2 and 4, are also printed out.

The switch Statement

In the last section, you saw that nested if statements are used when there is more than one decision to be made. The nested if statements will become very complex if there are many decisions that need to be made, however. Sometimes, the programmer will have problems following the complex if statements.

Fortunately there is another statement in C, the switch statement, that you can use to make unlimited decisions or choices based on the value of a conditional expression and specified cases.

The general form of the switch statement is

```
switch (expression) {
   case expression1:
        statement1;
   case expression2:
        statement2;
   .
   .
   .
   default:
        statement-default;
}
```

Here the conditional expression, expression, is evaluated first. If the return value of expression is equal to the constant expression expression1, the statement statement1 is executed. If the value of expression is the same as the value of expression2, statement2 is executed. If, however, the value of expression is not equal to any values of the constant expressions labeled by the case keyword, the statement (statement-default) following the default keyword is executed.

You have to use the case keyword to label each case. The default keyword is recommended to be used for the default case. Note that no constant expressions are identical in the switch statement.

The program in Listing 10.4 gives you an example of using the switch statement. The program also demonstrates an important feature of the switch statement.

TYPE
Listing 10.4. Using the switch statement.

```
1:  /* 10L04.c  Using the switch statement */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int day;
7:
8:     printf("Please enter a single digit for a day\n");
9:     printf("(within the range of 1 to 3):\n");
10:    day = getchar();
11:    switch (day){
12:       case `1':
13:          printf("Day 1\n");
14:       case `2':
15:          printf("Day 2\n");
16:       case `3':
17:          printf("Day 3\n");
18:       default:
19:          ;
20:    }
21:    return 0;
22: }
```

OUTPUT
If I run the executable file 10L04.exe and enter 3, I obtain the following output:

```
C:\app>10L04
Please enter a single digit for a day
(within the range of 1 to 3):
3
Day 3
C:\app>
```

ANALYSIS
As you can see in line 6, an int variable, day, is declared; it is assigned the input entered by the user in line 10.

In line 11, the value of the integer variable day is evaluated in the switch statement. If the value is equal to one of the values of the constant expressions, the computer starts to execute statements from there. The constant expressions are labeled by prefixing case in front of them.

For instance, I entered 3 and then pressed the Enter key. The numeric value of 3 is assigned to day in line 10. Then, after finding a case in which the value of the constant expression matches the value contained by day, the computer jumps to line 17 to execute the printf() function and display Day 3 on the screen.

Note that under the default label in Listing 10.4, there is an empty (that is, null) statement ending with semicolon (;) in line 19. The computer does nothing with the empty statement.

However, if I enter 1 from my keyboard and then press the Enter key, I get the following output:

```
C:\app>10L04
Please enter a single digit for a day
(within the range of 1 to 3):
1
Day 1
Day 2
Day 3
C:\app>
```

OUTPUT
From the output, you can see that the statement controlled by the selected case, case 1, and the statements controlled by the rest of the cases are executed, because Day 1, Day 2, and Day 3 are displayed on the screen. Likewise, if I enter 2 from my keyboard, I have Day2 and Day3 shown on the screen.

This is an important feature of the switch statement: The computer continues to execute the statements following the selected case until the end of the switch statement.

You're going to learn how to exit from the switch construct in the next section.

The break Statement

You can add a break statement at the end of the statement list following every case label, if you want to exit the switch construct after the statements within a selected case are executed.

The program in Listing 10.5 does a similar job as the one in Listing 10.4, but this time, the break statement is used.

TYPE
Listing 10.5. Adding the break statement.

```
1:  /* 10L05.c  Adding the break statement */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int day;
7:
8:     printf("Please enter a single digit for a day\n");
9:     printf("(within the range of 1 to 7):\n");
10:    day = getchar();
11:    switch (day){
12:       case `1':
13:          printf("Day 1 is Sunday.\n");
14:          break;
15:       case `2':
16:          printf("Day 2 is Monday.\n");
17:          break;
18:       case `3':
19:          printf("Day 3 is Tuesday.\n");
20:          break;
21:       case `4':
22:          printf("Day 4 is Wednesday.\n");
23:          break;
24:       case `5':
25:          printf("Day 5 is Thursday.\n");
26:          break;
27:       case `6':
28:          printf("Day 6 is Friday.\n");
29:          break;
30:       case `7':
31:          printf("Day 7 is Saturday.\n");
32:          break;
33:       default:
34:          printf("The digit is not within the range of 1 to 7.\n");
35:          break;
36:    }
37:    return 0;
38: }
```

OUTPUT
With the help from the break statement, I can run the executable file 10L05.exe and only obtain the output of the selected case:

```
C:\app>10L05
Please enter a single digit for a day
(within the range of 1 to 7):
1
Day 1 is Sunday.
C:\app>
```

ANALYSIS
This program has seven case labels followed by the constant expressions of `1', `2', `3', `4', `5', `6', and `7', respectively. (See lines 12, 15, 18, 21, 24, 27, and 30.)

In each case, there is a statement followed by a break statement. As mentioned, the break statements help to exit the switch construct after the statement in a selected case is executed.

For example, after the int variable day is assigned the value of 1 and evaluated in the switch statement, the case with `1' is selected, and the statement in line 13 is executed. Then, the break statement in line 14 is executed, which breaks the control of the switch statement and returns the control to the next statement outside the switch construct. In Listing 10.5, the next statement is the return statement in line 37, which ends the main function.

The printf() function in line 13 outputs a string of Day 1 is Sunday. on the screen.

Note that in a switch statement, braces are not needed to group the statements within an individual case into a statement block.

Breaking an Infinite Loop

You can also use the break statement to break an infinite loop. As you saw in Hour 7, the following for and while loops are all infinite loops:

```
for (;;){
   statement1;
   statement2;
   .
   .
   .
}

while (1){
   statement1;
   statement2;
   .
   .
   .
}
```

The program in Listing 10.6 shows an example of using the break statement in an infinite while loop.

TYPE
Listing 10.6. Breaking an infinite loop.

```
1:  /* 10L06.c: Breaking an infinite loop */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int c;
7:
8:     printf("Enter a character:\n(enter x to exit)\n");
9:     while (1) {
10:       c = getc(stdin);
11:       if (c == `x')
12:          break;
13:    }
14:    printf("Break the infinite while loop. Bye!\n");
15:    return 0;
16: }
```

OUTPUT
The following is the result I got after running the executable file (10L06.exe) on my machine:

```
C:\app>10L06
Enter a character:
(enter x to exit)
H
I
x
Break the infinite while loop. Bye!
C:\app>
```

ANALYSIS
There is an infinite while loop in Listing 10.6, which starts in line 9 and ends in line 13. Within the infinite loop, the characters the user entered are assigned, one at a time, to the integer variable c. (See line 10.)

The relational expression c == `x' in the if statement (see line 11) is evaluated each time during the looping. If the expression returns 0 (that is, the user does not enter the letter x), the looping continues. Otherwise, the break statement in line 12 is executed, which causes the computer to jump out of the infinite loop and start executing the next statement, which is shown in line 14.

You can see in the sample output, the while loop continues until I have entered the letter x, which causes the infinite loop to be broken and a piece of message, Break the infinite while loop. Bye!, to be displayed on the screen.

The continue Statement

Instead of breaking a loop, there are times when you want to stay in a loop but skip over some statements within the loop. To do this, you can use the continue statement provided by C. The continue statement causes execution to jump to the top of the loop immediately.

You should be aware that using the continue statement, as well as the break statement, may make your program hard to debug.

For example, Listing 10.7 demonstrates how to use the continue statement in a loop doing sums.

TYPE
Listing 10.7. Using the continue statement.

```
1:  /* 10L07.c: Using the continue statement */
```

```
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int i, sum;
7:
8:     sum = 0;
9:     for (i=1; i<8; i++){
10:       if ((i==3) || (i==5))
11:          continue;
12:       sum += i;
13:    }
14:    printf("The sum of 1, 2, 4, 6, and 7 is: %d\n", sum);
15:    return 0;
16: }
```

OUTPUT

After the executable file 10L07.exe is run from a DOS prompt, the output is shown on the screen:

```
C:\app>10L07
The sum of 1, 2, 4, 6, and 7 is: 20
C:\app>
```

ANALYSIS

In Listing 10.7, we want to calculate the sum of the integer values of 1, 2, 4, 6, and 7. Because the integers are almost consecutive, a for loop is built in lines 9_13. The statement in line 12 sums all consecutive integers from 1 to 7 (except for 3 and 5, which aren't in the listing and are skipped in the for loop).

By doing so, the expression (i==3) || (i==5) is evaluated in the if statement of line 10. If the expression returns 1 (that is, the value of i is equal to either 3 or 5), the continue statement in line 11 is executed, which causes the sum operation in line 12 to be skipped, and another iteration to be started at the beginning of the for loop. In this way, we obtain the sum of the integer values of 1, 2, 4, 6, and 7, but skip 3 and 5, automatically by using one for loop.

After the for loop, the value of sum, 20, is displayed on the screen by the printf() function in the statement of line 14.

The goto Statement

I feel that this book would not be not complete without mentioning the goto statement, although I do not recommend that you use the goto statement unless it's absolutely necessary. The main reason that the goto statement is discouraged is because its usage may make the C program unreliable and hard to debug.

The following is the general form of the goto statement:

```
labelname:
   statement1;
   statement2;
   .
   .
   .
   goto  labelname;
```

Here labelname is a label name that tells the goto statement where to jump. You have to place labelname in two places: One is at the place where the goto statement is going to jump (note that a colon must follow the label name), and the other is the place following the goto keyword. You have to follow the same rules to make a label name as you name a variable or function.

Also, the place for the goto statement to jump to can appear either before or after the statement.

Summary

In this lesson you've learned the following:

- An important task of a program is to instruct the computer to jump to a different portion of the code according to the specified branch conditions.
- The if statement is a very important statement for conditional branching in C.
- The if statement can be nested for making a series of decisions in your program.
- The if-else statement is an expansion of the if statement.
- The switch statement helps you to keep your program more readable when there are more than just a couple decisions to be made in your code.
- The case and default keywords, followed by a colon (:) and an integral value, are used in the switch statement as labels.
- The break statement can be used to exit the switch construct or a loop (usually, an infinite loop).
- The continue statement is used to let you stay within a loop while skipping over some statements.
- The goto statement enables the computer to jump to some other spot in your computer. Using this statement is not recommended because it may cause your program to be unreliable and hard to debug.

In the next lesson you'll learn about a very important concept—pointers.

Q&A

Q How many expressions are there in the if statement?

A The if statement takes only one expression to hold the conditional criteria. When the expression is true (that is, the conditions are met), the statements controlled by the if statement are executed. Otherwise, the next statement following the if statement block is executed.

Q Why is the if-else statement an expansion of the if statement?

A When the conditional expression in the if statement is false, the program control flow is returned back to the original track. However, when the conditional expression in the if-else statement is false, the program control flow branches to the statement block under the else keyword and returns to its original track after the statements controlled by else are executed. In other words, the if statement allows a single statement block to be executed or skipped entirely, whereas the if-else statement executes one of the two statement blocks under the control of the if-else statement.

Q Why do you normally need to add the break statement into the switch statement?

A When one of the cases within the switch statement is selected, the program control will branch to the case and execute all statements within the selected case and the rest of the cases that follow it. Therefore, you might get more results than you expected. To tell the computer to execute only the statements inside a selected case, you can put a break statement at the end of the case so that the program control flow will exit the switch construct after the statements within the case are executed.

Q What can the continue statement do inside a loop?

A When the continue statement inside a loop is executed, the program control is branched back to the beginning of the loop so that another iteration can be started. Inside the loop, any statements following the continue statement will be skipped over each time if the continue statement is executed.

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 x = 0, will the arithmetic operations inside the following if statement be performed?

```
if (x != 0)
   y = 123 / x + 456;
```

2. Given x = 4, y = 2, and operator = `-', what is the final value of x after the following switch statement is executed?

```
switch (operator){
   case `+':  x += y;
   case `-':  x -= y;
   case `*':  x *= y;
   case `/':  x /= y;
   default:   break;
}
```

3. Similarly to in question 2, using x = 4, y = 2, and operator = `-', what is the final value of x after the following switch statement is executed?

```
switch (operator){
   case `+':  x += y; break;
   case `-':  x -= y; break;
   case `*':  x *= y; break;
   case `/':  x /= y; break;
   default:   break;
}
```

4. What is the value of the integer variable x after the following code is executed?

```
x = 1;
for (i=2; i<10; i++){
   if (i%3 == 0)
      continue;
   x += i;
}
```

Exercises

1. Rewrite the program in Listing 10.1. This time, use the logical expression i%6 == 0 in the if statement.
2. Rewrite the program in Listing 10.1 by using nested if statements.
3. Write a program to read characters from the standard I/O. If the characters are A, B, and C, display their numeric values on the screen. (The switch statement is required.)
4. Write a program that keeps reading characters from the standard input until the character q is entered.
5. Rewrite the program in Listing 10.7. This time, instead of skipping 3 and 5, skip the integer that can be divided evenly by both 2 and 3.