Stephen Prata

Sixth Edition

C Primer Plus

C Primer Plus

Sixth Edition

Key Concepts

One aspect of intelligence is the ability to adjust one's responses to the circumstances. Therefore, selection statements are the foundation for developing programs that behave intelligently. In C, the `if`, `if else`, and `switch` statements, along with the conditional operator (`?:`), implement selection.

The `if` and `if else` statements use a test condition to determine which statements are executed. Any nonzero value is treated as `true`, whereas zero is treated as `false`. Typically, tests involve relational expressions, which compare two values, and logical expressions, which use logical operators to combine or modify other expressions.

One general principle to keep in mind is that if you want to test for two conditions, you should use a logical operator together with two complete test expressions. For instance, the following two attempts are faulty:

```
if (a < x < z)           // wrong --no logical operator
...
if (ch != 'q' && != 'Q') // wrong -- missing a complete test
...
```

Remember, the correct way is to join two relational expressions with a logical operator:

```
if (a < x && x < z)          // use && to combine two expressions
...
if (ch != 'q' && ch != 'Q')  // use && to combine two expressions
...
```

The control statements presented in these last two chapters will enable you to tackle programs that are much more powerful and ambitious than those you worked with before. For evidence, just compare some of the examples in these chapters to those of the earlier chapters.

Summary

This chapter has presented quite a few topics to review, so let's get to it. The `if` statement uses a test condition to control whether a program executes the single simple statement or block following the test condition. Execution occurs if the test expression has a nonzero value and doesn't occur if the value is zero. The `if else` statement enables you to select from two alternatives. If the test condition is nonzero, the statement before the `else` is executed. If the test expression evaluates to zero, the statement following the `else` is executed. By using another `if` statement to immediately follow the `else`, you can set up a structure that chooses between a series of alternatives.

The test condition is often a *relational expression*—that is, an expression formed by using one of the relational operators, such as < or ==. By using C's logical operators, you can combine relational expressions to create more complex tests.

The *conditional operator* (`? :`) creates an expression that, in many cases, provides a more compact alternative to an `if else` statement.

The `ctype.h` family of character functions, such as `isspace()` and `isalpha()`, offers convenient tools for creating test expressions based on classifying characters.

The `switch` statement enables you to select from a series of statements labeled with integer values. If the integer value of the test condition following the `switch` keyword matches a label, execution goes to the statement bearing that label. Execution then proceeds through the statements following the labeled statement unless you use a `break` statement.

Finally, `break`, `continue`, and `goto` are jump statements that cause program flow to jump to another location in the program. A `break` statement causes the program to jump to the next statement following the end of the loop or `switch` containing the `break`. The `continue` statement causes the program to skip the rest of the containing loop and to start the next cycle.

Review Questions

You'll find answers to the review questions in Appendix A, "Answers to the Review Questions."

1. Determine which expressions are `true` and which are `false`.

   a. `100 > 3 && 'a'>'c'`

   b. `100 > 3 || 'a'>'c'`

   c. `!(100>3)`

2. Construct an expression to express the following conditions:

   a. `number` is equal to or greater than 90 but smaller than 100.

   b. `ch` is not a `q` or a `k` character.

   c. `number` is between 1 and 9 (including the end values) but is not a 5.

   d. `number` is not between 1 and 9.

3. The following program has unnecessarily complex relational expressions as well as some outright errors. Simplify and correct it.

```
#include <stdio.h>
int main(void)                                      /* 1  */
{                                                   /* 2  */
  int weight, height;  /* weight in lbs, height in inches */
                                                    /* 4  */
  scanf("%d , weight, height);                      /* 5  */
  if (weight < 100 && height > 64)                  /* 6  */
     if (height >= 72)                              /* 7  */
        printf("You are very tall for your weight.\n");
     else if (height < 72 &&  > 64)                   /* 9  */
        printf("You are tall for your weight.\n");  /* 10 */
  else if (weight > 300 && ! (weight <= 300)        /* 11 */
           && height < 48)                          /* 12 */
     if (!(height >= 48) )                          /* 13 */
         printf(" You are quite short for your weight.\n");
  else                                              /* 15 */
     printf("Your weight is ideal.\n");             /* 16 */
                                                    /* 17 */
  return 0;
}
```

4. What is the numerical value of each of the following expressions?

   a. `5 > 2`

   b. `3 + 4 > 2 && 3 < 2`

   c. `x >= y || y > x`

   d. `d = 5 + ( 6 > 2 )`

   e. `'X' > 'T' ? 10 : 5`

   f. `x > y ? y > x : x > y`

5. What will the following program print?

```
#include <stdio.h>
int main(void)
{
  int num;
  for (num = 1; num <= 11; num++)
  {
       if (num % 3 == 0)
            putchar('$');
       else
            putchar('*');
            putchar('#');
       putchar('%');
  }
  putchar('\n');
  return 0;
}
```

6. What will the following program print?

```
#include <stdio.h>
int main(void)
```

```
{
    int i = 0;
    while ( i < 3) {
       switch(i++) {
           case 0 : printf("fat ");
           case 1 : printf("hat ");
           case 2 : printf("cat ");
           default: printf("Oh no!");
       }
       putchar('\n');
    }
    return 0;
}
```

7. What's wrong with this program?

```
#include <stdio.h>
int main(void)
{
  char ch;
  int lc = 0;    /* lowercase char count
  int uc = 0;    /* uppercase char count
  int oc = 0;    /* other char count

  while ((ch = getchar()) != '#')
  {
       if ('a' <= ch >= 'z')
            lc++;
       else if (!(ch < 'A') || !(ch > 'Z')
            uc++;
       oc++;
  }
  printf(%d lowercase, %d uppercase, %d other, lc, uc, oc);
  return 0;
}
```

8. What will the following program print?

```
/* retire.c   */
#include <stdio.h>
int main(void)
{
  int age = 20;
  while (age++ <= 65)
  {
     if (( age % 20) == 0) /* is age divisible by 20? */
         printf("You are %d. Here is a raise.\n", age);
     if (age = 65)
         printf("You are %d. Here is your gold watch.\n", age);
  }
  return 0;
}
```

9. What will the following program print when given this input?

```
q
c
h
b
#include <stdio.h>
int main(void)
{
  char ch;

  while ((ch = getchar()) != '#')
  {
       if (ch == '\n')
            continue;
       printf("Step 1\n");
       if (ch == 'c')
            continue;
       else if (ch == 'b')
            break;
       else if (ch == 'h')
            goto laststep;
       printf("Step 2\n");
  laststep:  printf("Step 3\n");
  }
  printf("Done\n");
  return 0;
}
```

10. Rewrite the program in Review Question 9 so that it exhibits the same behavior but does not use a `continue` or a `goto`.

Programming Exercises

1. Write a program that reads input until encountering the # character and then reports the number of spaces read, the number of newline characters read, and the number of all other characters read.

2. Write a program that reads input until encountering #. Have the program print each input character and its ASCII decimal code. Print eight character-code pairs per line. Suggestion: Use a character count and the modulus operator (`%`) to print a newline character for every eight cycles of the loop.

3. Write a program that reads integers until 0 is entered. After input terminates, the program should report the total number of even integers (excluding the 0) entered, the average value of the even integers, the total number of odd integers entered, and the average value of the odd integers.

4. Using `if else` statements, write a program that reads input up to #, replaces each period with an exclamation mark, replaces each exclamation mark initially present with two exclamation marks, and reports at the end the number of substitutions it has made.

5. Redo exercise 4 using a `switch`.

6. Write a program that reads input up to # and reports the number of times that the sequence `ei` occurs.

Note

The program will have to "remember" the preceding character as well as the current character. Test it with input such as "Receive your eieio award."

7. Write a program that requests the hours worked in a week and then prints the gross pay, the taxes, and the net pay. Assume the following:

   a. Basic pay rate = $10.00/hr

   b. Overtime (in excess of 40 hours) = time and a half

   c. Tax rate: #15% of the first $300

   20% of the next $150

   25% of the rest

   Use `#define` constants, and don't worry if the example does not conform to current tax law.