

4

C Program Control



OBJECTIVES

In this chapter you will learn:

- To understand multiple selection using the switch selection statement.
- To use the logical operators to form complex conditional expressions in control statements.
- To avoid the consequences of confusing the equality and assignment operators.



4.7 **switch Multiple-Selection Statement**

4.10 **Logical Operators**

4.11 **Confusing Equality (==) and Assignment (=) Operators**



4.7 switch Multiple-Selection Statement

- **switch**

- Useful when a variable or expression is tested for all the values it can assume and different actions are taken

- **Format**

- Series of case labels and an optional default case

```
switch ( value ){  
    case '1':  
        actions  
    case '2':  
        actions  
    default:  
        actions  
}
```

- **break;** exits from statement



Outline

fig04_07.c

(1 of 4)

```

1  /* Fig. 4.7: fig04_07.c
2     Counting letter grades */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8      int grade;      /* one grade */
9      int aCount = 0; /* number of As */
10     int bCount = 0; /* number of Bs */
11     int cCount = 0; /* number of Cs */
12     int dCount = 0; /* number of Ds */
13     int fCount = 0; /* number of Fs */
14
15     printf( "Enter the letter grades.\n" );
16     printf( "Enter the EOF character to end input.\n" );
17
18     /* loop until user types end-of-file key sequence */
19     while ( ( grade = getchar() ) != EOF ) {
20
21         /* determine which grade was input */
22         switch ( grade ) { /* switch nested in while */
23
24             case 'A': /* grade was uppercase A */
25             case 'a': /* or lowercase a */
26                 ++aCount; /* increment aCount */
27                 break; /* necessary to exit switch */
28

```

EOF stands for “end of file;” this character varies from system to system

switch statement checks each of its nested **cases** for a match

break statement makes program skip to end of **switch**



Outline

fig04_07.c

(2 of 4)

```
29 case 'B': /* grade was uppercase B */
30 case 'b': /* or lowercase b */
31     ++bCount; /* increment bCount */
32     break; /* exit switch */
33
34 case 'C': /* grade was uppercase C */
35 case 'c': /* or lowercase c */
36     ++cCount; /* increment cCount */
37     break; /* exit switch */
38
39 case 'D': /* grade was uppercase D */
40 case 'd': /* or lowercase d */
41     ++dCount; /* increment dCount */
42     break; /* exit switch */
43
44 case 'F': /* grade was uppercase F */
45 case 'f': /* or lowercase f */
46     ++fCount; /* increment fCount */
47     break; /* exit switch */
48
49 case '\n': /* ignore newlines, */
50 case '\t': /* tabs, */
51 case ' ': /* and spaces in input */
52     break; /* exit switch */
53
```



```
54     default: /* catch all other characters */
55         printf( "Incorrect letter grade entered." );
56         printf( " Enter a new grade.\n" );
57         break; /* optional; will exit switch anyway */
58     } /* end switch */
59
60 } /* end while */
61
62 /* output summary of results */
63 printf( "\nTotals for each letter grade are:\n" );
64 printf( "A: %d\n", aCount ); /* display number of A grades */
65 printf( "B: %d\n", bCount ); /* display number of B grades */
66 printf( "C: %d\n", cCount ); /* display number of C grades */
67 printf( "D: %d\n", dCount ); /* display number of D grades */
68 printf( "F: %d\n", fCount ); /* display number of F grades */
69
70 return 0; /* indicate program ended successfully */
71
72 } /* end function main */
```

Outline

default case occurs if none of the
cases are matched

fig04_07.c

(3 of 4)



Outline

fig04_07.c

(4 of 4)

```
Enter the letter grades.  
Enter the EOF character to end input.  
a  
b  
C  
C  
A  
d  
f  
C  
E  
Incorrect letter grade entered. Enter a new grade.  
D  
A  
b  
^Z  
  
Totals for each letter grade are:  
A: 3  
B: 2  
C: 3  
D: 2  
F: 1
```



Portability Tip 4.1

The keystroke combinations for entering EOF (end of file) are system dependent.



Portability Tip 4.2

Testing for the symbolic constant EOF rather than -1 makes programs more portable. The C standard states that EOF is a negative integral value (but not necessarily -1). Thus, EOF could have different values on different systems.



Common Programming Error 4.5

Forgetting a `break` statement when one is needed in a `switch` statement is a logic error.



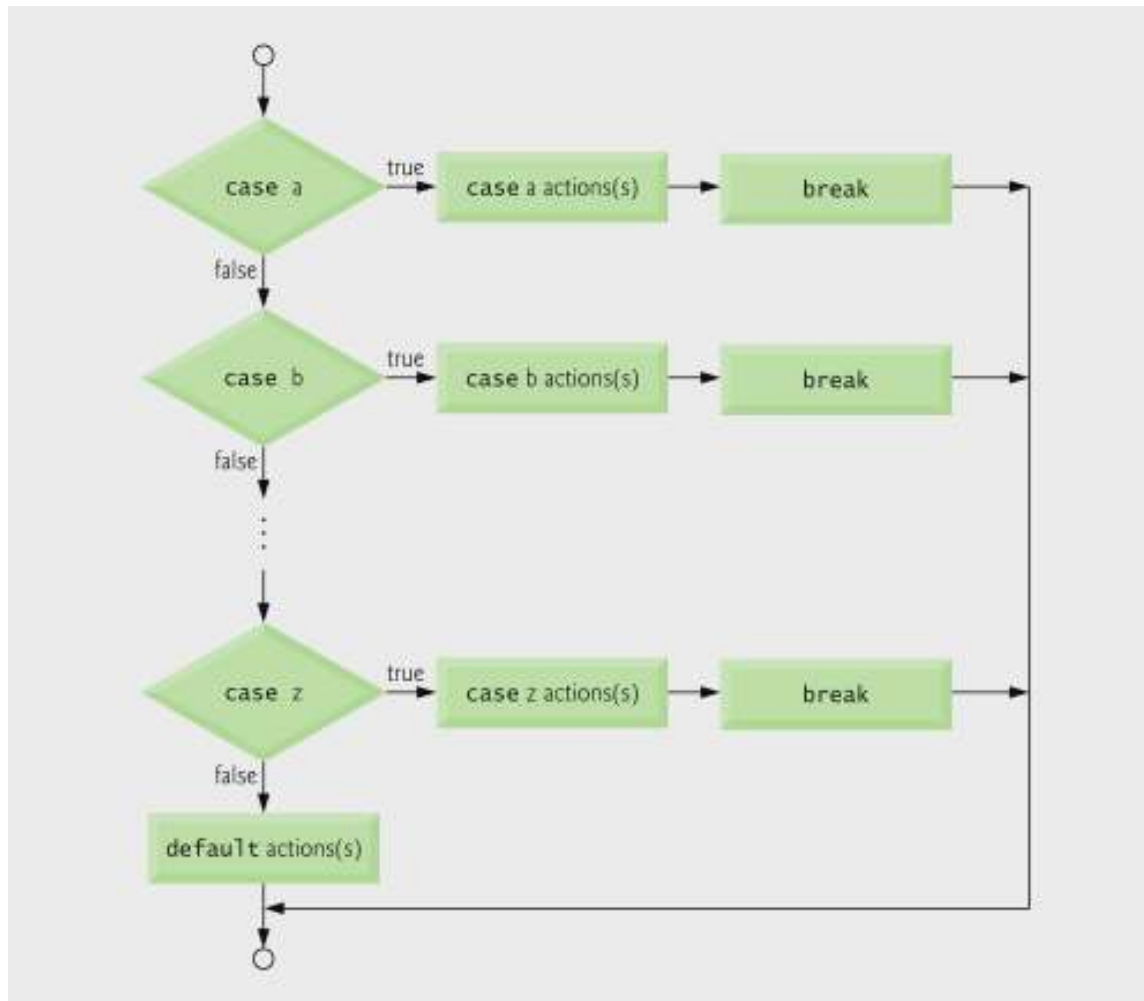


Fig. 4.8 | switch multiple-selection statement with **breaks**.



Good Programming Practice 4.7

Provide a default case in switch statements. Cases not explicitly tested in a switch are ignored. The default case helps prevent this by focusing the programmer on the need to process exceptional conditions. There are situations in which no default processing is needed.



Good Programming Practice 4.8

Although the `case` clauses and the `default` case clause in a `switch` statement can occur in any order, it is considered good programming practice to place the `default` clause last.



Good Programming Practice 4.9

In a `switch` statement when the `default` clause is listed last, the `break` statement is not required. But some programmers include this `break` for clarity and symmetry with other cases.



Common Programming Error 4.6

Not processing newline characters in the input when reading characters one at a time can cause logic errors.



Error-Prevention Tip 4.5

Remember to provide processing capabilities for newline (and possibly other white-space) characters in the input when processing characters one at a time.



4.10 Logical Operators

- **&& (logical AND)**
 - Returns true if both conditions are true
- **|| (logical OR)**
 - Returns true if either of its conditions are true
- **! (logical NOT, logical negation)**
 - Reverses the truth/falsity of its condition
 - Unary operator, has one operand
- **Useful as conditions in loops**

<u>Expression</u>	<u>Result</u>
true && false	false
true false	true
!false	true



expression1	expression2	expression1 && expression2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Fig. 4.13 | Truth table for the **&&** (logical AND) operator.



expression1	expression2	expression1 expression2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Fig. 4.14 | Truth table for the logical **OR** (||) operator.



expression	!expression
0	1
nonzero	0

Fig. 4.15 | Truth table for operator ! (logical negation).



Performance Tip 4.2

In expressions using operator `&&`, make the condition that is most likely to be false the leftmost condition. In expressions using operator `||`, make the condition that is most likely to be true the leftmost condition. This can reduce a program's execution time.



Operators	Associativity	Type
++ (<i>postfix</i>) -- (<i>postfix</i>)	right to left	postfix
+ - ! ++ (<i>prefix</i>) -- (<i>prefix</i>) (<i>type</i>)	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
 	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

Fig. 4.16 | Operator precedence and associativity.



4.11 Confusing Equality (==) and Assignment (=) Operators

■ Dangerous error

- Does not ordinarily cause syntax errors
- Any expression that produces a value can be used in control structures
- Nonzero values are true, zero values are false
- Example using ==:

```
if ( payCode == 4 )  
    printf( "You get a bonus!\n" );
```

- Checks payCode, if it is 4 then a bonus is awarded



4.11 Confusing Equality (==) and Assignment (=) Operators

- Example, replacing == with =:

```
if ( payCode = 4 )  
    printf( "You get a bonus!\n" );
```

This sets payCode to 4

4 is nonzero, so expression is true, and
bonus awarded no matter what the
payCode was

- Logic error, not a syntax error



Common Programming Error 4.8

Using operator `==` for assignment or using operator `=` for equality is a logic error.



4.11 Confusing Equality (==) and Assignment (=) Operators

■ lvalues

- Expressions that can appear on the left side of an equation
- Their values can be changed, such as variable names
 - `x = 4;`

■ rvalues

- Expressions that can only appear on the right side of an equation
- Constants, such as numbers
 - Cannot write `4 = x;`
 - Must write `x = 4;`
- lvalues can be used as rvalues, but not vice versa
 - `y = x;`



Good Programming Practice 4.11

When an equality expression has a variable and a constant, as in `x == 1`, some programmers prefer to write the expression with the constant on the left and the variable name on the right (e.g. `1 == x` as protection against the logic error that occurs when you accidentally replace operator `==` with `=`).



Error-Prevention Tip 4.6

After you write a program, text search it for every = and check that it is being used properly.

