2

Introduction to C Programming

2.6 Decision Making: Equality and Relational Operators

- **Executable statements**
  - **Perform actions (calculations, input/output of data)**
  - **Perform decisions**
    - **May want to print `"pass"` or `"fail"` given the value of a test grade**
- **`if` control statement**
  - **Simple version in this section, more detail later**
  - **If a condition is `true`, then the body of the `if` statement executed**
    - **`0` is `false`, non-zero is `true`**
  - **Control always resumes after the `if` structure**
- **Keywords**
  - **Special words reserved for C**
  - **Cannot be used as identifiers or variable names**

Standard algebraic equality operator or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.12 | Equality and relational operators.

Outline

fig02_13.c

(1 of 3)

```
1  /* Fig. 2.13: fig02_13.c
2     Using if statements, relational
3     operators, and equality operators */
4  #include <stdio.h>
5
6  /* function main begins program execution */
7  int main( void )
8  {
9     int num1; /* first number to be read from user  */
10    int num2; /* second number to be read from user */
11
12    printf( "Enter two integers, and I will tell you\n" );
13    printf( "the relationships they satisfy: " );
14
15    scanf( "%d%d", &num1, &num2 ); /* read two integers */
16
17    if ( num1 == num2 ) {
18       printf( "%d is equal to %d\n", num1, num2 );
19    } /* end if */
20
21    if ( num1 != num2 ) {
22       printf( "%d is not equal to %d\n", num1, num2 );
23    } /* end if */
24
25    if ( num1 < num2 ) {
26       printf( "%d is less than %d\n", num1, num2 );
27    } /* end if */
28
```

Checks if **num1** is equal to **num2**

Checks if **num1** is not equal to **num2**

Checks if **num1** is less than **num2**

```
29    if ( num1 > num2 ) {
30       printf( "%d is greater than %d\n", num1, num2 );
31    } /* end if */
32
33    if ( num1 <= num2 ) {
34       printf( "%d is less than or equal to %d\n", num1, num2 );
35    } /* end if */
36
37    if ( num1 >= num2 ) {
38       printf( "%d is greater than or equal to %d\n", num1, num2 );
39    } /* end if */
40
41    return 0;   /* indicate that program ended successfully */
42
43 } /* end function main */
43 } /* end function main */
```

Checks if **num1** is greater than **num2**

Checks if **num1** is less than or equal to **num2**

Checks if **num1** is greater than equal to **num2**

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

(continued on next slide…)

Outline

fig02_13.c

(2 of 3)

(continued from previous slide…)

```
Enter two integers, and I will tell you
the relationships they satisfy:
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12


Enter two integers, and I will tell you
the relationships they satisfy:
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Outline

fig02_13.c

(3 of 3)

Common Programming Errors

2.16: A syntax error will occur if the two symbols in any of the operators ==, !=, >= and <= are separated by spaces.

2.17: A syntax error will occur if the two symbols in any of the operators !=, >= and <= are reversed as in =!, => and =<, respectively.

2.18: Confusing the equality operator == with the assignment operator =.

Standard algebraic equality operator or relational operator	C equality or relational operator
Equality operators	
=	==
≠	!=
Relational operators	
>	>
<	<
≥	>=
≤	<=

Common Programming Error 2.19

Placing a semicolon immediately to the right of the right parenthesis after the condition in an `if` statement.

```
7  int main( void )
8  {
9     int num1; /* first number to be read from user  */
10    int num2; /* second number to be read from user */
11
12    printf( "Enter two integers, and I will tell you\n" );
13    printf( "the relationships they satisfy: " );
14
15    scanf( "%d%d", &num1, &num2 ); /* read two integers */
16
17    if ( num1 == num2 ) {
18       printf( "%d is equal to %d\n", num1, num2 );
19    } /* end if */
20
21    if ( num1 != num2 ) {
22       printf( "%d is not equal to %d\n", num1, num2 );
23    } /* end if */
```

Good Programming Practice 2.13-14

Indent the statement(s) in the body of an `if` statement.

Place a blank line before and after every `if` statement in a program for readability.

```
7  int main( void )
8  {
9     int num1; /* first number to be read from user  */
10    int num2; /* second number to be read from user */
11
12    printf( "Enter two integers, and I will tell you\n" );
13    printf( "the relationships they satisfy: " );
14
15    scanf( "%d%d", &num1, &num2 ); /* read two integers */
16
17    if ( num1 == num2 ) {
18       printf( "%d is equal to %d\n", num1, num2 );
19    } /* end if */
20
21    if ( num1 != num2 ) {
22       printf( "%d is not equal to %d\n", num1, num2 );
23    } /* end if */
```

Good Programming Practice 2.15

Although it is allowed, there should be no more than one statement per line in a program.

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**

Expression	Result
`true && false`	`false`
`true || false`	`true`
`!false`	`true`

Good Programming Practice 2.17

Refer to the operator precedence chart when writing expressions containing many operators. Confirm that the operators in the expression are applied in the proper order. If you are uncertain about the order of evaluation in a complex expression, use parentheses to group expressions or break the statement into several simpler statements. Be sure to observe that some of C's operators such as the assignment operator (=) associate from right to left rather than from left to right.

Operators	Associativity	Type
`++` *(postfix)* `--` *(postfix)*	right to left	postfix
`+` `-` `!` `++` *(prefix)* `--` *(prefix)* (*type*)	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

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 =.

3

Structured Program Development in C

3.5 The `if` selection statement

- **Selection structure:**
  - **Used to choose among alternative courses of action**
  - **Pseudocode:**

    If student's grade is greater than or equal to 60
    Print "Passed"
- **If condition `true`**
  - **Print statement executed and program goes on to next statement**
  - **If `false`, print statement is ignored and the program goes onto the next statement**
  - **Indenting makes programs easier to read**
    - **C ignores whitespace characters**

3.5 The if selection statement

- **Pseudocode statement in C:**

```
if ( grade >= 60 )
   printf( "Passed\n" );
```

  - **C code corresponds closely to the pseudocode**
- **Diamond symbol (decision symbol)**
  - **Indicates decision is to be made**
  - **Contains an expression that can be `true` or `false`**
  - **Test the condition, follow appropriate path**

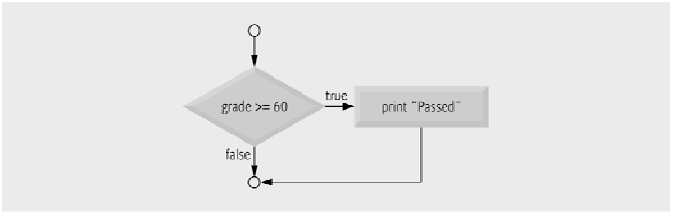


Fig. 3.2 | Flowcharting the single-selection **`if`** statement.

Common Programming Error 3.2

Placing a semicolon after the condition in an `if` statement as in `if ( grade >= 60 );` leads to a logic error in single-selection `if` statements and a syntax error in double-selection `if` statements.

3.6 The `if…else` selection statement

- `if`
  - Only performs an action if the condition is `true`
- `if…else`
  - Specifies an action to be performed both when the condition is `true` and when it is `false`
- Psuedocode:

  If student's grade is greater than or equal to 60
  Print "Passed"
  else
  Print "Failed"
  - Note spacing/indentation conventions

3.6 The `if…else` selection statement

- **C code:**

```
if ( grade >= 60 )
   printf( "Passed\n");
else
   printf( "Failed\n");
```

- **Ternary conditional operator (`?:`)**
  - **Takes three arguments (condition, value if `true`, value if `false`)**
  - **Our pseudocode could be written:**

```
printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );
```

  - **Or it could have been written:**

```
grade >= 60 ? printf( “Passed\n” ) :
  printf( “Failed\n” );
```

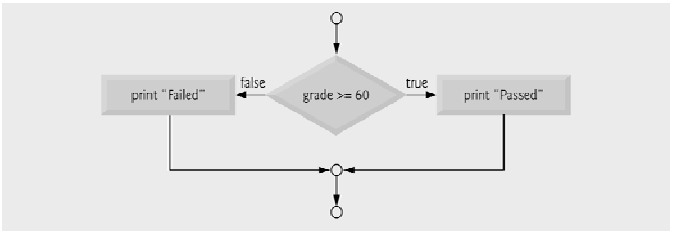


Fig. 3.3 | Flowcharting the double-selection **`if...else`** statement.

3.6 The `if…else` selection statement

- **Nested `if…else` statements**
  - **Test for multiple cases by placing `if…else` selection statements inside `if…else` selection statement**
  - **Once condition is met, rest of statements skipped**
  - **Deep indentation usually not used in practice**

3.6 The `if…else` selection statement

- **Pseudocode for a nested `if…else` statement**

```
If student's grade is greater than or equal to 90
  Print "A"
else
  If student's grade is greater than or equal to 80
    Print "B"
  else
    If student's grade is greater than or equal to 70
      Print "C"
    else
      If student's grade is greater than or equal to 60
        Print "D"
      else
        Print "F"
```

3.6 The `if…else` selection statement

- **Compound statement:**
  - **Set of statements within a pair of braces**
  - **Example:**

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else {
   printf( "Failed.\n" );
   printf( "You must take this course
      again.\n" );
 }
```

  - **Without the braces, the statement**

```
printf( "You must take this course
   again.\n" );
```

  would be executed automatically

3.6 The `if…else` selection statement

- **Block:**
  - **Compound statements with declarations**
- **Syntax errors**
  - **Caught by compiler**
- **Logic errors:**
  - **Have their effect at execution time**
  - **Non-fatal: program runs, but has incorrect output**
  - **Fatal: program exits prematurely**

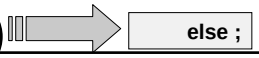
Error-Prevention Tip 3.1

Typing the beginning and ending braces of compound statements before typing the individual statements within the braces helps avoid omitting one or both of the braces, preventing syntax errors and logic errors (where both braces are indeed required).

Software Engineering Observation 3.2

Just as a compound statement can be placed anywhere a single statement can be placed, it is also possible to have no statement at all, i.e., the empty statement. The empty statement is represented by placing a semicolon (;) where a statement would normally be.

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else {
   printf( "Failed.\n" );
   printf( "You must take this course
    again.\n" );
}
```



4

C Program Control

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

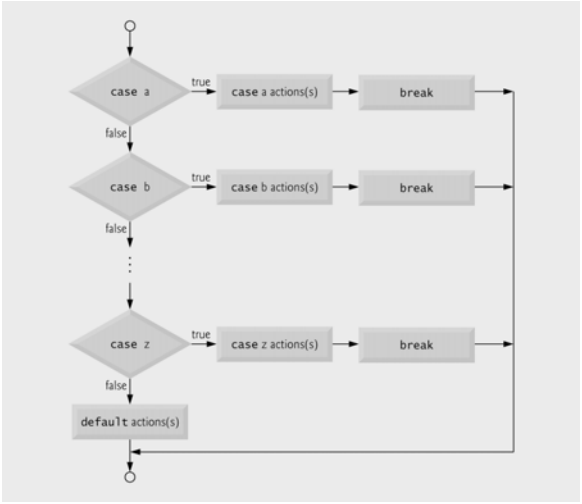


Fig. 4.8 | `switch` multiple-selection statement with `breaks`.

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 **`case`**s for a match

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

Outline

```
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
```

fig04_07.c

(2 of 4)

Outline

```
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 */
```

`default` case occurs if none of the **`case`**s are matched

fig04_07.c

(3 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
```

Outline

fig04_07.c

(4 of 4)

Common Programming Error 4.5

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

```
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 */
```

Keep going to next actions

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.

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.

```
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 */
```