

Chapter 7 - Structured Program Development

Outline

1. Introduction
2. Structured Programming Summary
3. The **if** Selection Statement
4. The **if...Else** Selection Statement
5. The **While** Repetition Statement
6. Formulating Algorithms with Top-down, Stepwise Refinement
7. Assignment Operators
8. Increment and Decrement Operators
9. The **for** Repetition Statement
10. The **switch** Multiple-Selection Statement
11. The **do...while** Repetition Statement
12. The **break** and **continue** Statements
13. Logical Operators
14. Confusing Equality (**==**) and Assignment (**=**) Operators



Structured-Programming Summary

- All programs can be broken down into 3 controls
 - Sequence – handled automatically by compiler
 - Selection – `if`, `if...else` or `switch`
 - Repetition – `while`, `do...while` or `for`
 - Can only be combined in two ways
 - Nesting (rule 3)
 - Stacking (rule 2)
 - Any selection can be rewritten as an `if` statement, and any repetition can be rewritten as a `while` statement



The if Selection Statement

- Pseudocode statement in C:

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

- C code corresponds closely to the pseudocode



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



The `if...else` Selection Statement

- C code:

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



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"



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



The while Repetition Statement

- Repetition structure
 - Programmer specifies an action to be repeated while some condition remains `true`
 - Psuedocode:
 - While there are more items on my shopping list*
 - Purchase next item and cross it off my list*
 - `while` loop repeated until condition becomes `false`



The while Repetition Statement

- Example:

```
int product = 2;  
while ( product <= 1000 )  
    product = 2 * product;
```





fig03_06.c (Part 1 of 2)

```
1  /* Fig. 3.6: fig03_06.c
2     Class average program with counter-controlled repetition */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter; /* number of grade to be entered next */
9     int grade;   /* grade value */
10    int total;   /* sum of grades input by user */
11    int average; /* average of grades */
12
13    /* initialization phase */
14    total = 0;   /* initialize total */
15    counter = 1; /* initialize loop counter */
16
17    /* processing phase */
18    while ( counter <= 10 ) { /* loop 10 times */
19        printf( "Enter grade: " ); /* prompt for input */
20        scanf( "%d", &grade ); /* read grade from user */
21        total = total + grade; /* add grade to total */
22        counter = counter + 1; /* increment counter */
23    } /* end while */
24
```

```
25  /* termination phase */
26  average = total / 10;          /* integer division */
27
28  /* display result */
29  printf( "Class average is %d\n", average );
30
31  return 0; /* indicate program ended successfully */
32
33 } /* end function main */
```

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```



Outline

11

fig03_06.c (Part 2 of 2)

Program Output

Formulating Algorithms with Top-Down, Stepwise Refinement

Initialize total to zero

Initialize counter to zero

Input the first grade

While the user has not as yet entered the sentinel

Add this grade into the running total

Add one to the grade counter

Input the next grade (possibly the sentinel)

If the counter is not equal to zero

Set the average to the total divided by the counter

Print the average

else

Print “No grades were entered”





Outline

fig03_08.c (Part 1 of 2)

```
1  /* Fig. 3.8: fig03_08.c
2     Class average program with sentinel-controlled repetition */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter;    /* number of grades entered */
9     int grade;     /* grade value */
10    int total;     /* sum of grades */
11
12    float average; /* number with decimal point for average */
13
14    /* initialization phase */
15    total = 0;     /* initialize total */
16    counter = 0;  /* initialize loop counter */
17
18    /* processing phase */
19    /* get first grade from user */
20    printf( "Enter grade, -1 to end: " );    /* prompt for input */
21    scanf( "%d", &grade );                /* read grade from user */
22
23    /* loop while sentinel value not yet read from user */
24    while ( grade != -1 ) {
25        total = total + grade;            /* add grade to total */
26        counter = counter + 1;          /* increment counter */
27
```

```
28     printf( "Enter grade, -1 to end: " ); /* prompt for input */
29     scanf("%d", &grade); /* read next grade */
30 } /* end while */
31
32 /* termination phase */
33 /* if user entered at least one grade */
34 if ( counter != 0 ) {
35
36     /* calculate average of all grades entered */
37     average = ( float ) total / counter;
38
39     /* display average with two digits of precision */
40     printf( "Class average is %.2f\n", average );
41 } /* end if */
42 else { /* if no grades were entered, output message */
43     printf( "No grades were entered\n" );
44 } /* end else */
45
46 return 0; /* indicate program ended successfully */
47
48 } /* end function main */
```



Outline

14

fig03_08.c (Part 2 of 2)



Outline



Program Output

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

```
Enter grade, -1 to end: -1
No grades were entered
```

Assignment Operators

- Assignment operators abbreviate assignment expressions

`c = c + 3;`

can be abbreviated as `c += 3;` using the addition assignment operator

- Statements of the form

variable = variable operator expression;

can be rewritten as

variable operator= expression;

- Examples of other assignment operators:

`d -= 4` (`d = d - 4`)

`e *= 5` (`e = e * 5`)

`f /= 3` (`f = f / 3`)

`g %= 9` (`g = g % 9`)



Assignment Operators

Assume: <code>int c = 3, d = 5, e = 4, f = 6, g = 12;</code>			
Assignment operator	Sample expression	Explanation	Assigns
<code>+=</code>	<code>c += 7</code>	<code>c = c + 7</code>	10 to c
<code>-=</code>	<code>d -= 4</code>	<code>d = d - 4</code>	1 to d
<code>*=</code>	<code>e *= 5</code>	<code>e = e * 5</code>	20 to e
<code>/=</code>	<code>f /= 3</code>	<code>f = f / 3</code>	2 to f
<code>%=</code>	<code>g %= 9</code>	<code>g = g % 9</code>	3 to g

Fig. 3.11 Arithmetic assignment operators.



Increment and Decrement Operators

- Increment operator (`++`)
 - Can be used instead of `c+=1`
- Decrement operator (`--`)
 - Can be used instead of `c-=1`
- Preincrement
 - Operator is used before the variable (`++c` or `--c`)
 - Variable is changed before the expression it is in is evaluated
- Postincrement
 - Operator is used after the variable (`c++` or `c--`)
 - Expression executes before the variable is changed



Increment and Decrement Operators

- If `c` equals 5, then
 - `printf("%d", ++c);`
 - Prints 6
 - `printf("%d", c++);`
 - Prints 5
 - In either case, `c` now has the value of 6
- When variable not in an expression
 - Preincrementing and postincrementing have the same effect
 - `++c;`
 - `printf("%d", c);`
 - Has the same effect as
 - `c++;`
 - `printf("%d", c);`



Increment and Decrement Operators

Operator	Sample expression	Explanation
++	++a	Increment a by 1 then use the new value of a in the expression in which a resides.
++	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	Decrement b by 1 then use the new value of b in the expression in which b resides.
--	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Fig. 3.12 The increment and decrement operators





Outline

fig03_13.c

```
1  /* Fig. 3.13: fig03_13.c
2     Preincrementing and postincrementing */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int c;                /* define variable */
9
10    /* demonstrate postincrement */
11    c = 5;                /* assign 5 to c */
12    printf( "%d\n", c ); /* print 5 */
13    printf( "%d\n", c++ ); /* print 5 then postincrement */
14    printf( "%d\n\n", c ); /* print 6 */
15
16    /* demonstrate preincrement */
17    c = 5;                /* assign 5 to c */
18    printf( "%d\n", c ); /* print 5 */
19    printf( "%d\n", ++c ); /* preincrement then print 6 */
20    printf( "%d\n", c ); /* print 6 */
21
22    return 0; /* indicate program ended successfully */
23
24 }
```

5
5
6

5
6
6



Outline



Program Output

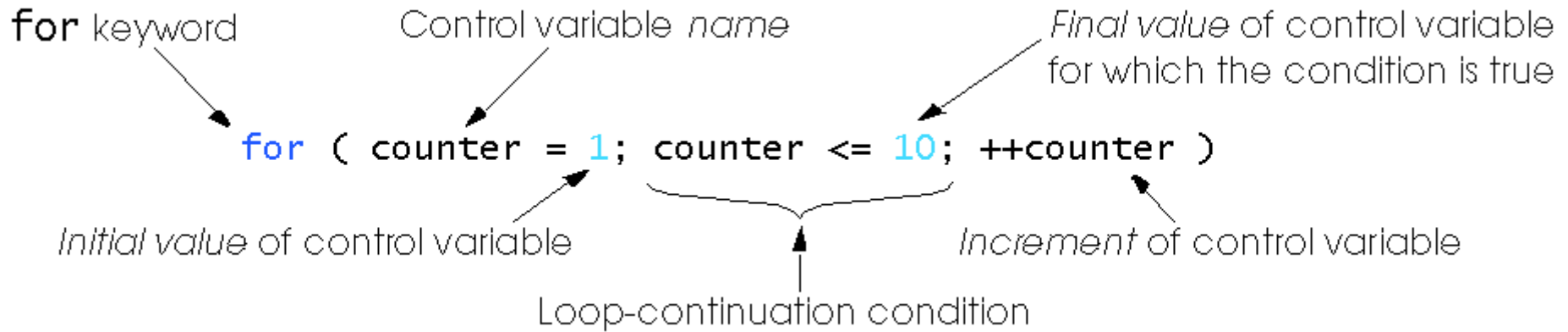
Increment and Decrement Operators

Operators					Associativity	Type
++	--	+	-	(type)	right to left	unary
*	/	%			left to right	multiplicative
+	-				left to right	additive
<	<=	>	>=		left to right	relational
==	!=				left to right	equality
?:					right to left	conditional
=	+=	-=	*=	/=	right to left	assignment

Fig. 3.14 Precedence of the operators encountered so far in the text.



The for Repetition Statement



The for Repetition Statement

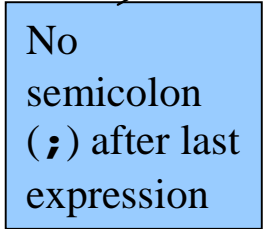
- Format when using for loops

```
for ( initialization; loopContinuationTest; increment )  
    statement
```

- Example:

```
for( int counter = 1; counter <= 10; counter++ )  
    printf( "%d\n", counter );
```

- Prints the integers from one to ten



No
semicolon
(;) after last
expression



[Outline](#)

fig04_05.c

```
1  /* Fig. 4.5: fig04_05.c
2     Summation with for */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int sum = 0; /* initialize sum */
9     int number; /* number to be added to sum */
10
11    for ( number = 2; number <= 100; number += 2 ) {
12        sum += number; /* add number to sum */
13    } /* end for */
14
15    printf( "Sum is %d\n", sum ); /* output sum */
16
17    return 0; /* indicate program ended successfully */
18
19 } /* end function main */
```

Sum is 2550

Program Output



```
1  /* Fig. 4.6: fig04_06.c
2     calculating compound interest */
3  #include <stdio.h>
4  #include <math.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     double amount;           /* amount on deposit */
10    double principal = 1000.0; /* starting principal */
11    double rate = .05;       /* interest rate */
12    int year;                /* year counter */
13
14    /* output table column head */
15    printf( "%4s%21s\n", "Year", "Amount on deposit" );
16
17    /* calculate amount on deposit for each of ten years */
18    for ( year = 1; year <= 10; year++ ) {
19
20        /* calculate new amount for specified year */
21        amount = principal * pow( 1.0 + rate, year );
22
23        /* output one table row */
24        printf( "%4d%21.2f\n", year, amount );
25    } /* end for */
26
```

```
27     return 0; /* indicate program ended successfully */
28
29 } /* end function main */
```

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89



Outline

fig04_06.c (Part 2 of 2)

Program Output

The switch Multiple-Selection Statement

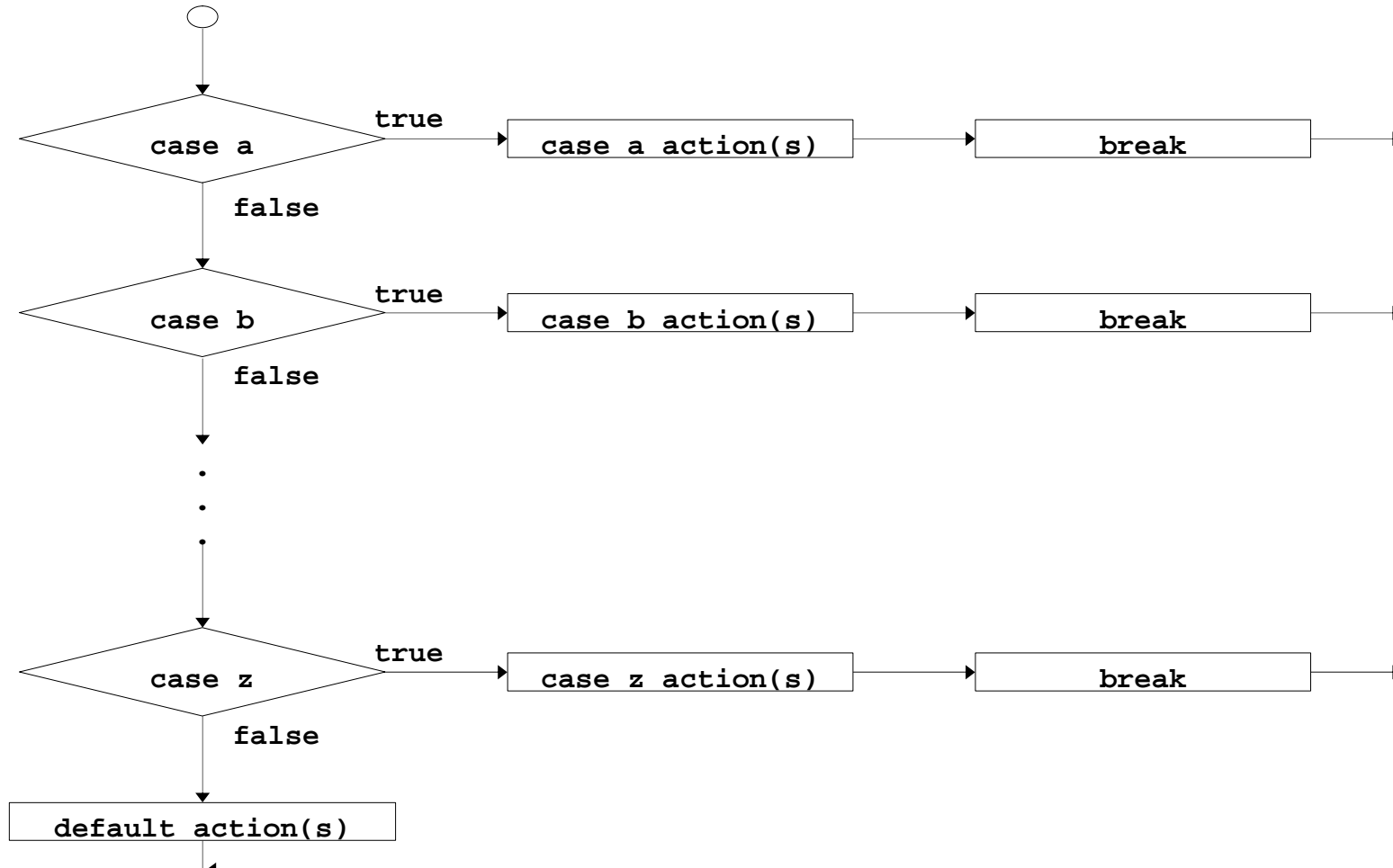
- **switch**
 - Useful when a variable or expression is tested for all the values it can assume and different actions are taken
- **Format**
 - Series of case labels and an optional default case

```
switch ( value ){
    case '1':
        actions
    case '2':
        actions
    default:
        actions
}
```
 - **break;** exits from statement



The switch Multiple-Selection Statement

- Flowchart of the switch statement





```
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()
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
```



```
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 */
```



```
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
```

The do...while Repetition Statement

- The do...while repetition statement
 - Similar to the while structure
 - Condition for repetition tested after the body of the loop is performed
 - All actions are performed at least once
 - Format:

```
do {  
    statement;  
} while ( condition );
```



The do...while Repetition Statement

- Example (letting counter = 1):

```
do {  
    printf( "%d  ", counter );  
} while (++counter <= 10);
```

 - Prints the integers from 1 to 10





Outline

```
1  /* Fig. 4.9: fig04_09.c
2     Using the do/while repetition statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter = 1; /* initialize counter */
9
10    do {
11        printf( "%d ", counter ); /* display counter */
12    } while ( ++counter <= 10 ); /* end do...while */
13
14    return 0; /* indicate program ended successfully */
15
16 } /* end function main */
```

1 2 3 4 5 6 7 8 9 10

The break and continue Statements

- **break**
 - Causes immediate exit from a `while`, `for`, `do...while` or `switch` statement
 - Program execution continues with the first statement after the structure
 - Common uses of the `break` statement
 - Escape early from a loop
 - Skip the remainder of a `switch` statement





```
1  /* Fig. 4.11: fig04_11.c
2     Using the break statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13        /* if x is 5, terminate loop */
14        if ( x == 5 ) {
15            break; /* break loop only if x is 5 */
16        } /* end if */
17
18        printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nBroke out of loop at x == %d\n", x );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

1 2 3 4

Broke out of loop at x == 5

The break and continue Statements

- `continue`
 - Skips the remaining statements in the body of a `while`, `for` or `do...while` statement
 - Proceeds with the next iteration of the loop
 - `while` and `do...while`
 - Loop-continuation test is evaluated immediately after the `continue` statement is executed
 - `for`
 - Increment expression is executed, then the loop-continuation test is evaluated





```
1  /* Fig. 4.12: fig04_12.c
2     Using the continue statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13        /* if x is 5, continue with next iteration of loop */
14        if ( x == 5 ) {
15            continue; /* skip remaining code in loop body */
16        } /* end if */
17
18        printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nUsed continue to skip printing the value 5\n" );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

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>
<code>true && false</code>	<code>false</code>
<code>true false</code>	<code>true</code>
<code>!false</code>	<code>true</code>



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



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

