Chapter 3 - Structured Program Development

Outline

Objectives

- In this chapter, you will learn:
  - To understand basic problem solving techniques.
  - To be able to develop algorithms through the process of top-down, stepwise refinement.
  - To be able to use the `if` selection statement and `if…else` selection statement to select actions.
  - To be able to use the `while` repetition statement to execute statements in a program repeatedly.
  - To understand counter-controlled repetition and sentinel-controlled repetition.
  - To understand structured programming.
  - To be able to use the increment, decrement and assignment operators.

3.1 Introduction

- Before writing a program:
  - Have a thorough understanding of the problem
  - Carefully plan an approach for solving it
- While writing a program:
  - Know what “building blocks” are available
  - Use good programming principles

3.2 Algorithms

- Computing problems
  - All can be solved by executing a series of actions in a specific order
- Algorithm: procedure in terms of
  - Actions to be executed
  - The order in which these actions are to be executed
- Program control
  - Specify order in which statements are to be executed

3.3 Pseudocode

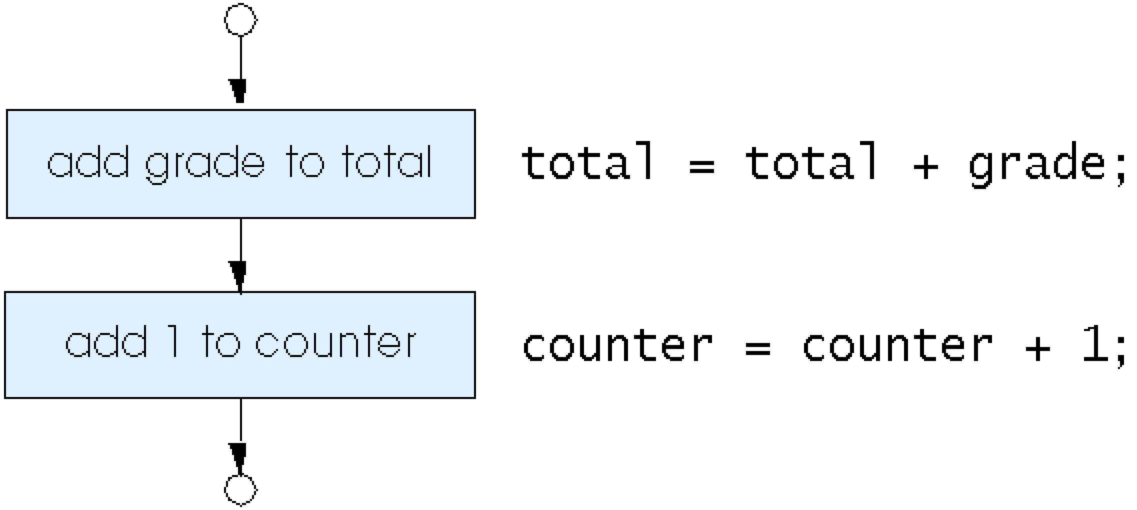
- Pseudocode
  - Artificial, informal language that helps us develop algorithms
  - Similar to everyday English
  - Not actually executed on computers
  - Helps us “think out” a program before writing it
    - Easy to convert into a corresponding C++ program
    - Consists only of executable statements

3.4 Control Structures

- Sequential execution
  - Statements executed one after the other in the order written
- Transfer of control
  - When the next statement executed is not the next one in sequence
  - Overuse of `goto` statements led to many problems
- Bohm and Jacopini
  - All programs written in terms of 3 control structures
    - Sequence structures: Built into C. Programs executed sequentially by default
    - Selection structures: C has three types: `if`, `if`…`else`, and `switch`
    - Repetition structures: C has three types: `while`, `do`…`while` and `for`

3.4 Control Structures

Figure 3.1 Flowcharting C's sequence structure.



3.4 Control Structures

- Flowchart
  - Graphical representation of an algorithm
  - Drawn using certain special-purpose symbols connected by arrows called flowlines
  - Rectangle symbol (action symbol):
    - Indicates any type of action
  - Oval symbol:
    - Indicates the beginning or end of a program or a section of code
- Single-entry/single-exit control structures
  - Connect exit point of one control structure to entry point of the next (control-structure stacking)
  - Makes programs easy to build

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

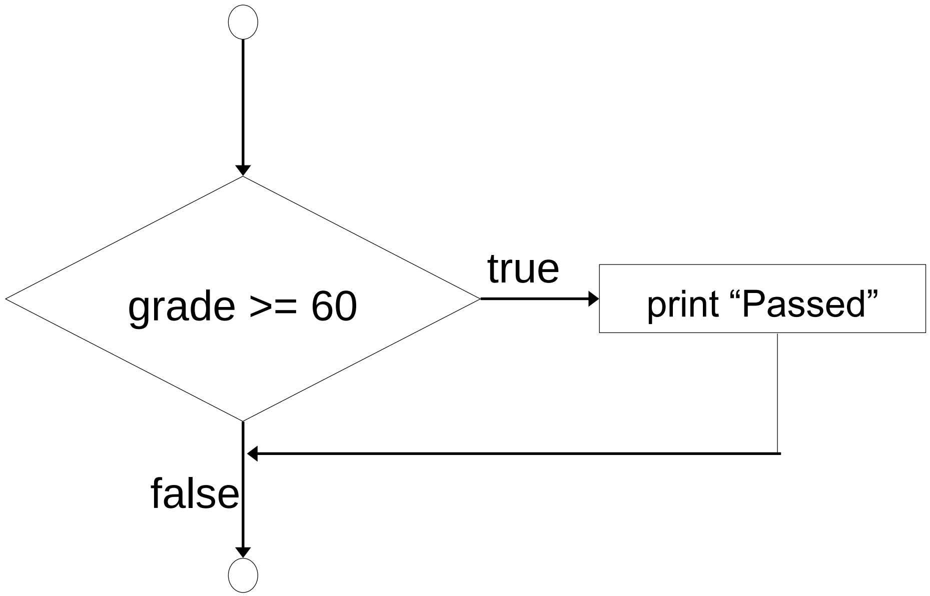
- Pseudocode statement in C:

```
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

3.5 The `if` Selection Statement

- `if` statement is a single-entry/single-exit structure



A decision can be made on any expression.

zero - `false`

nonzero - `true`

Example:

`3 - 4` is `true`

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:

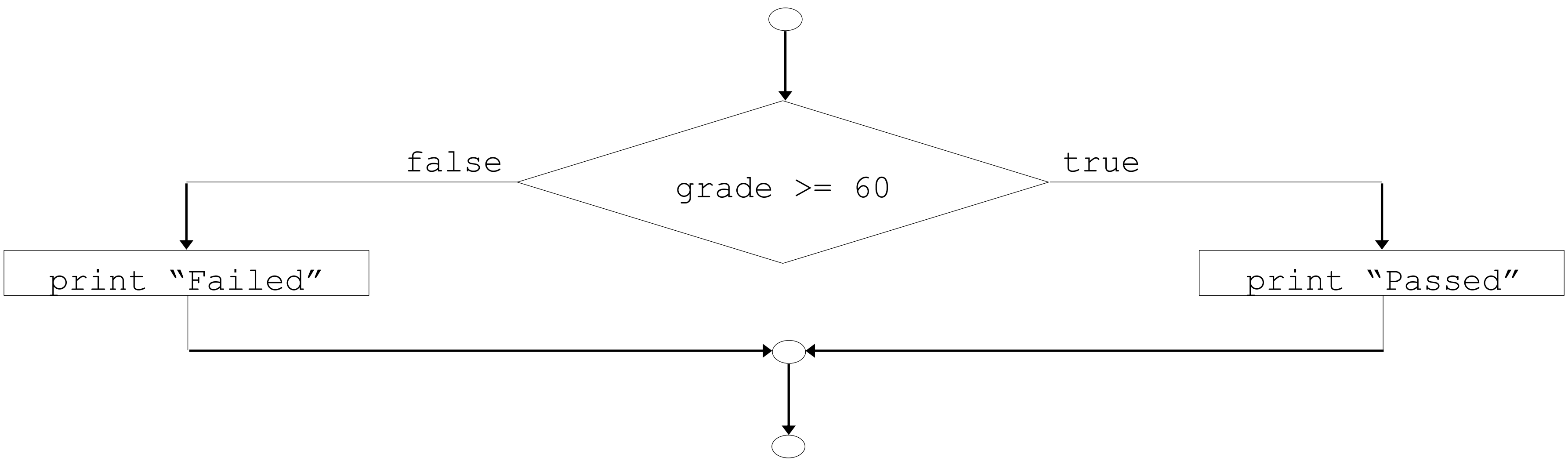
```
printf( "%s\n", grade >= 60 ? "Passed" :
  "Failed" );
```

  - Or it could have been written:

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

3.6 The `if...else` Selection Statement

- Flow chart of 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

3.7 The `while` Repetition Statement

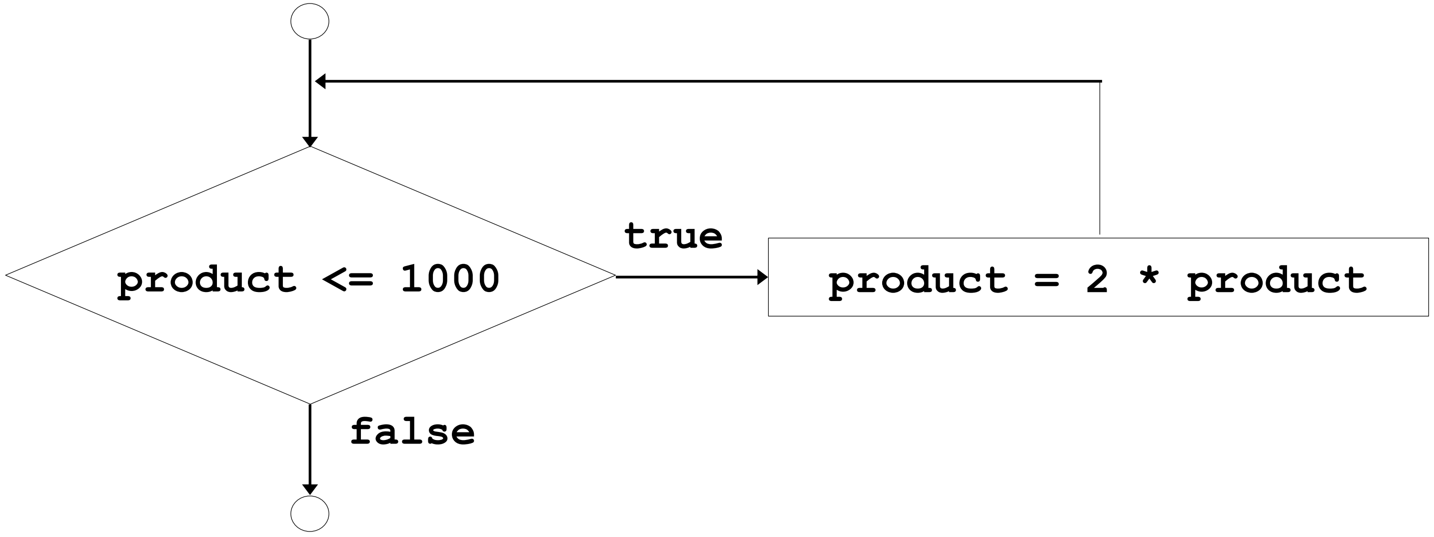
- Repetition structure
  - Programmer specifies an action to be repeated while some condition remains `true`
  - Psuedocode:

    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`

3.7 The while Repetition Statement

- Example:

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



3.8 Formulating Algorithms (Counter-Controlled Repetition)

- Counter-controlled repetition
  - Loop repeated until counter reaches a certain value
  - Definite repetition: number of repetitions is known
  - Example: A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz
  - Pseudocode:

Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
Input the next grade
Add the grade into the total
Add one to the grade counter

Set the class average to the total divided by ten
Print the class average

```c
/* Fig. 3.6: fig03_06.c
   Class average program with counter-controlled repetition */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   int counter; /* number of grade to be entered next */
   int grade;   /* grade value */
    int total;   /* sum of grades input by user */
    int average; /* average of grades */

    /* initialization phase */
    total = 0;    /* initialize total */
    counter = 1; /* initialize loop counter */

    /* processing phase */
    while ( counter <= 10 ) {     /* loop 10 times */
       printf( "Enter grade: " ); /* prompt for input */
       scanf( "%d", &grade );     /* read grade from user */
       total = total + grade;     /* add grade to total */
       counter = counter + 1;    /* increment counter */
    } /* end while */

```


fig03_06.c (Part 1 of 2)

```
   /* termination phase */
   average = total / 10;          /* integer division */

   /* display result */
   printf( "Class average is %d\n", average );

   return 0; /* indicate program ended successfully */

} /* 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
```

fig03_06.c (Part 2 of 2)

Program Output

3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

- Problem becomes:

  Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

  - Unknown number of students
  - How will the program know to end?
- Use sentinel value
  - Also called signal value, dummy value, or flag value
  - Indicates "end of data entry."
  - Loop ends when user inputs the sentinel value
  - Sentinel value chosen so it cannot be confused with a regular input (such as `-1` in this case)

3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

- Top-down, stepwise refinement
  - Begin with a pseudocode representation of the *top*:

    Determine the class average for the quiz
  - Divide *top* into smaller tasks and list them in order:

    Initialize variables
    Input, sum and count the quiz grades
    Calculate and print the class average
- Many programs have three phases:
  - Initialization: initializes the program variables
  - Processing: inputs data values and adjusts program variables accordingly
  - Termination: calculates and prints the final results

3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

- Refine the initialization phase from *Initialize variables* to:

  Initialize total to zero
  Initialize counter to zero

- Refine *Input, sum and count the quiz grades* to

  Input the first grade (possibly the sentinel)
  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)

3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

- Refine *Calculate and print the class average* to

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"

3.9 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"

fig03_08.c (Part 1 of 2)

```c
/* Fig. 3.8: fig03_08.c
   Class average program with sentinel-controlled repetition */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   int counter;   /* number of grades entered */
   int grade;     /* grade value */
   int total;     /* sum of grades */

   float average; /* number with decimal point for average */

   /* initialization phase */
   total = 0;     /* initialize total */
   counter = 0;   /* initialize loop counter */

   /* processing phase */
   /* get first grade from user */
   printf( "Enter grade, -1 to end: " );    /* prompt for input */
   scanf( "%d", &grade );                   /* read grade from user */

   /* loop while sentinel value not yet read from user */
   while ( grade != -1 ) {
      total = total + grade;                /* add grade to total */
      counter = counter + 1;                /* increment counter */

```

```
      printf( "Enter grade, -1 to end: " ); /* prompt for input */
      scanf("%d", &grade);                  /* read next grade */
   } /* end while */

   /* termination phase */
   /* if user entered at least one grade */
   if ( counter != 0 ) {

      /* calculate average of all grades entered */
      average = ( float ) total / counter;

      /* display average with two digits of precision */
      printf( "Class average is %.2f\n", average );
   } /* end if */
   else { /* if no grades were entered, output message */
      printf( "No grades were entered\n" );
   } /* end else */

   return 0; /* indicate program ended successfully */

} /* end function main */
```


fig03_08.c (Part 2 of 2)

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
```

3.10 Nested control structures

- Problem
  - A college has a list of test results (1 = pass, 2 = fail) for 10 students
  - Write a program that analyzes the results
    - If more than 8 students pass, print "Raise Tuition"
- Notice that
  - The program must process 10 test results
    - Counter-controlled loop will be used
  - Two counters can be used
    - One for number of passes, one for number of fails
  - Each test result is a number—either a 1 or a 2
    - If the number is not a 1, we assume that it is a 2

3.10 Nested control structures

- Top level outline

  Analyze exam results and decide if tuition should be raised

- First Refinement

  Initialize variables

  Input the ten quiz grades and count passes and failures

  Print a summary of the exam results and decide if tuition should be raised

- Refine *Initialize variables* to

  Initialize passes to zero

  Initialize failures to zero

  Initialize student counter to one

3.10 Nested control structures

- Refine *Input the ten quiz grades and count passes and failures* to

  While student counter is less than or equal to ten
  Input the next exam result
  If the student passed
  Add one to passes
  else
  Add one to failures
  Add one to student counter

- Refine *Print a summary of the exam results and decide if tuition should be raised* to

  Print the number of passes
  Print the number of failures
  If more than eight students passed
  Print “Raise tuition”

3.10 Nested control structures

```
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
        Input the next exam result

        If the student passed
                Add one to passes
        else
                Add one to failures

        Add one to student counter

Print the number of passes
Print the number of failures
If more than eight students passed
        Print "Raise tuition"
```

Outline

fig03_10.c (Part 1 of 2)

```
/* Fig. 3.10: fig03_10.c
   Analysis of examination results */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   /* initialize variables in definitions */
   int passes = 0;   /* number of passes */
    int failures = 0; /* number of failures */
    int student = 1;  /* student counter */
    int result;       /* one exam result */

    /* process 10 students using counter-controlled loop */
    while ( student <= 10 ) {

       /* prompt user for input and obtain value from user */
       printf( "Enter result ( 1=pass,2=fail ): " );
       scanf( "%d", &result );

       /* if result 1, increment passes */
       if ( result == 1 ) {
          passes = passes + 1;
       } /* end if */
```

```c
      else { /* otherwise, increment failures */
         failures = failures + 1;
      } /* end else */

      student = student + 1; /* increment student counter */
   } /* end while */

   /* termination phase; display number of passes and failures */
   printf( "Passed %d\n", passes );
   printf( "Failed %d\n", failures );

   /* if more than eight students passed, print "raise tuition" */
   if ( passes > 8 ) {
      printf( "Raise tuition\n" );
   } /* end if */

   return 0; /* indicate program ended successfully */

} /* end function main */
```

fig03_10.c (Part 2 of 2)

Outline

Program Output

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Passed 6
Failed 4
```

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Passed 9
Failed 1
Raise tuition
```

3.11 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)
```

3.11 Assignment Operators

Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`

Assignment operator	Sample expression	Explanation	Assigns
`+=`	`c += 7`	`c = c + 7`	10 to `c`
`-=`	`d -= 4`	`d = d - 4`	1 to `d`
`*=`	`e *= 5`	`e = e * 5`	20 to `e`
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to `g`

Fig. 3.11 Arithmetic assignment operators.

3.12 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

3.12 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 );
```

3.12 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

```
/* Fig. 3.13: fig03_13.c
   Preincrementing and postincrementing */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   int c;                    /* define variable */

   /* demonstrate postincrement */
   c = 5;                    /* assign 5 to c */
   printf( "%d\n", c );      /* print 5 */
   printf( "%d\n", c++ ); /* print 5 then postincrement */
   printf( "%d\n\n", c ); /* print 6 */

   /* demonstrate preincrement */
   c = 5;                    /* assign 5 to c */
   printf( "%d\n", c );      /* print 5 */
   printf( "%d\n", ++c ); /* preincrement then print 6 */
   printf( "%d\n", c );      /* print 6 */

   return 0; /* indicate program ended successfully */

} /* end function main */
```

```
5
5
6

5
6
6
```

Outline

Program Output

3.12 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.