Programming for Engineers

Structured Program

UNIVERSITY AT ALBANY
State University of New York

ICEN 360– Spring 2018

Prof. Dola Saha

Steps to writing a program

- Understand the problem
- Plan a solution
  - Step by step procedure

Algorithm

- The solution to any computing problem involves executing a series of actions in a specific order.
- A procedure for solving a problem in terms of
  - the actions to be executed, and
  - the order in which these actions are to be executed
- is called an algorithm.
- Correctly specifying the order in which the actions are to be executed is important.

Order matters

- Example “rise-and-shine” algorithm

In-Order	
1. Get out of bed	
2. Take of pajamas	
3. Take a shower	
4. Get dressed	
5. Eat breakfast	
6. Carpool to work	

Order matters

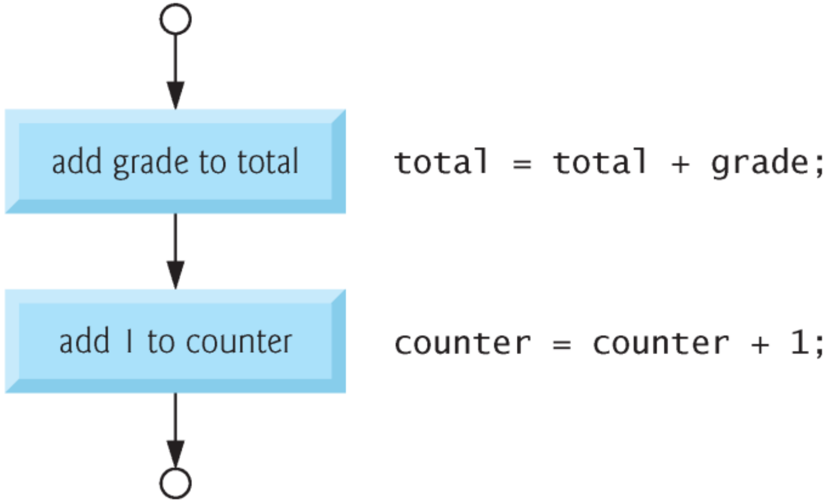
- Example “rise-and-shine” algorithm

In-Order	Out-of-Order
1. Get out of bed	1. Get out of bed
2. Take off pajamas	2. Take off pajamas
3. Take a shower	3. Get dressed
4. Get dressed	4. Take a shower
5. Eat breakfast	5. Eat breakfast
6. Go to school	6. Go to school

- Specifying the order in which statements are to be executed in a computer program is called program control.

Flow Chart

- Graphical representation of an algorithm
- Uses certain special-purpose symbols such as rectangles, diamonds, rounded rectangles, and small circles
- Symbols are connected by arrows called flowlines



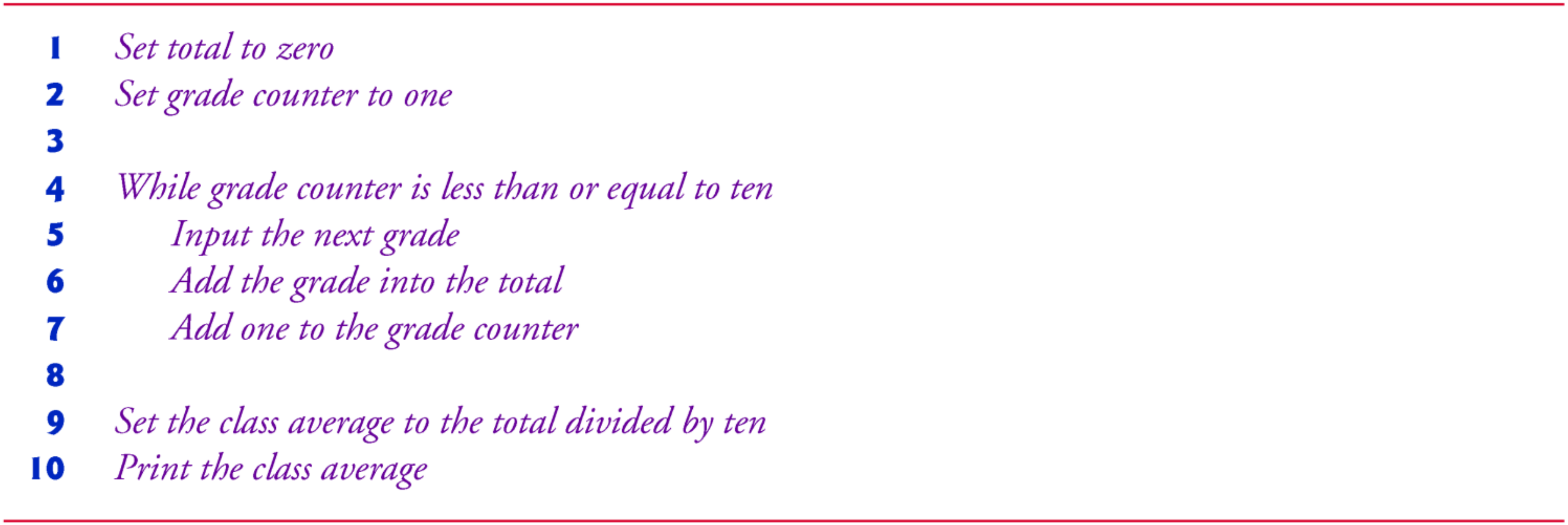
```
total = total + grade;
counter = counter + 1;
```

Flow Chart

- Rectangle symbol or action symbol indicate any type of action including a calculation or an input/output operation.
- The flowlines indicate the order in which the actions are performed.

Pseudocode

- Artificial, informal, user-friendly, convenient, English-like
- They are NOT executed on computers
- Can be easily converted into ANY programming language
- Consists of actions and decision statements



```
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
```

Control Structures

- Sequential execution: statements are executed one after another
- Transfer of control: Some C statements can specify that next statement to be executed MAY NOT be the next statement

Decision Making - Example

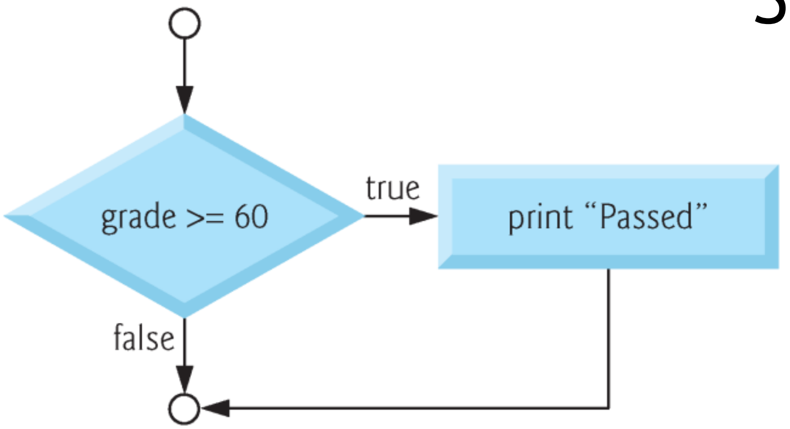
- Check condition
  - Is the distance between Albany to NYC more than Albany to Buffalo?
  - Is John’s grade greater than 60 ?
- Perform Tasks based on decision
  - If Albany to NYC is shorter, then I will drive to NYC
  - If Amy’s grade is greater than 60, then she passes
- Otherwise
  - I will drive to Buffalo
  - She fails

Selection Statement

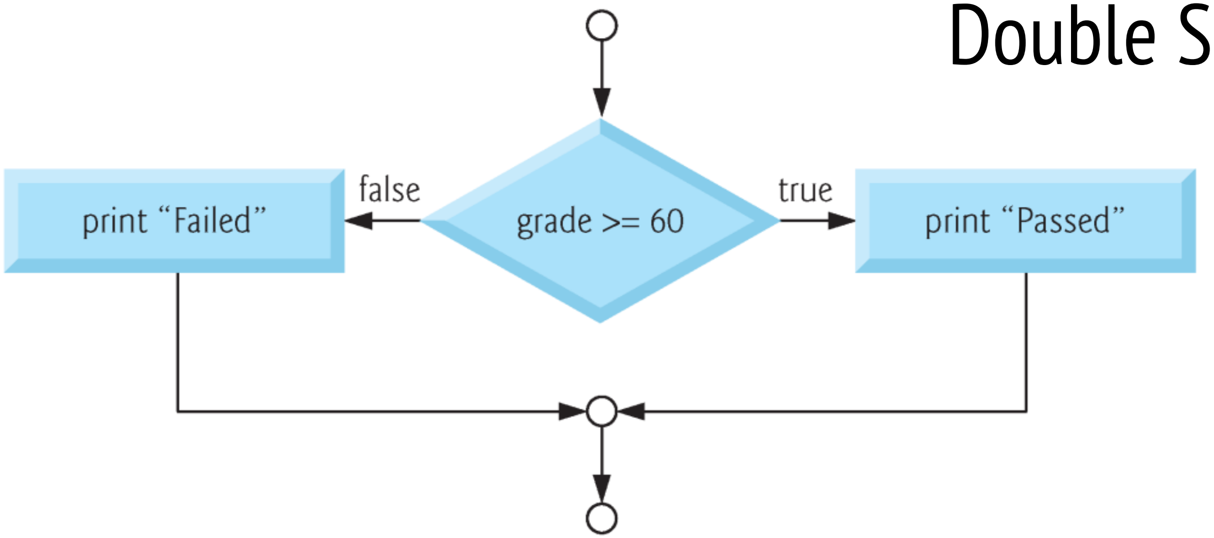
- `If` Statement
  - `If`:
    - Performs a set of actions if condition is TRUE,
    - otherwise skip
  - `If … else`:
    - Performs a set of actions if condition is TRUE,
    - otherwise performs a different set of actions
- `Switch` Statement:
  - Performs one of many different set of actions
- Used to choose among alternative courses of action.

Selection Statement in Flow Chart

Single Selection



Double Selection



Selection Statement in Pseudocode

```
If student's grade is greater than or equal to 60
    Print "Passed"
```

```
If student's grade is greater than or equal to 60
    Print "Passed"
else
    Print "Failed"
```

Selection Statement in C

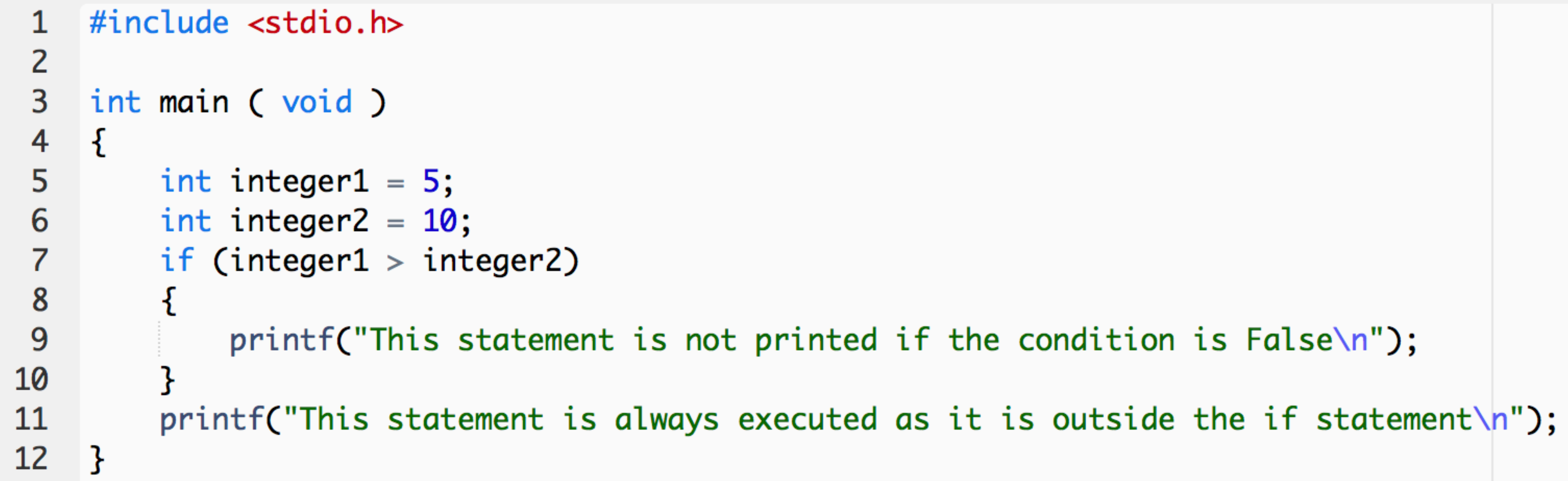
-
  ```c
  if ( grade >= 60 ) {
     printf( "Passed\n" );
  } // end if
  ```

-
  ```c
  if ( grade >= 60 ) {
     printf( "Passed\n" );
  } // end if
  else {
     printf( "Failed\n" );
  } // end else
  ```

`if` Statement

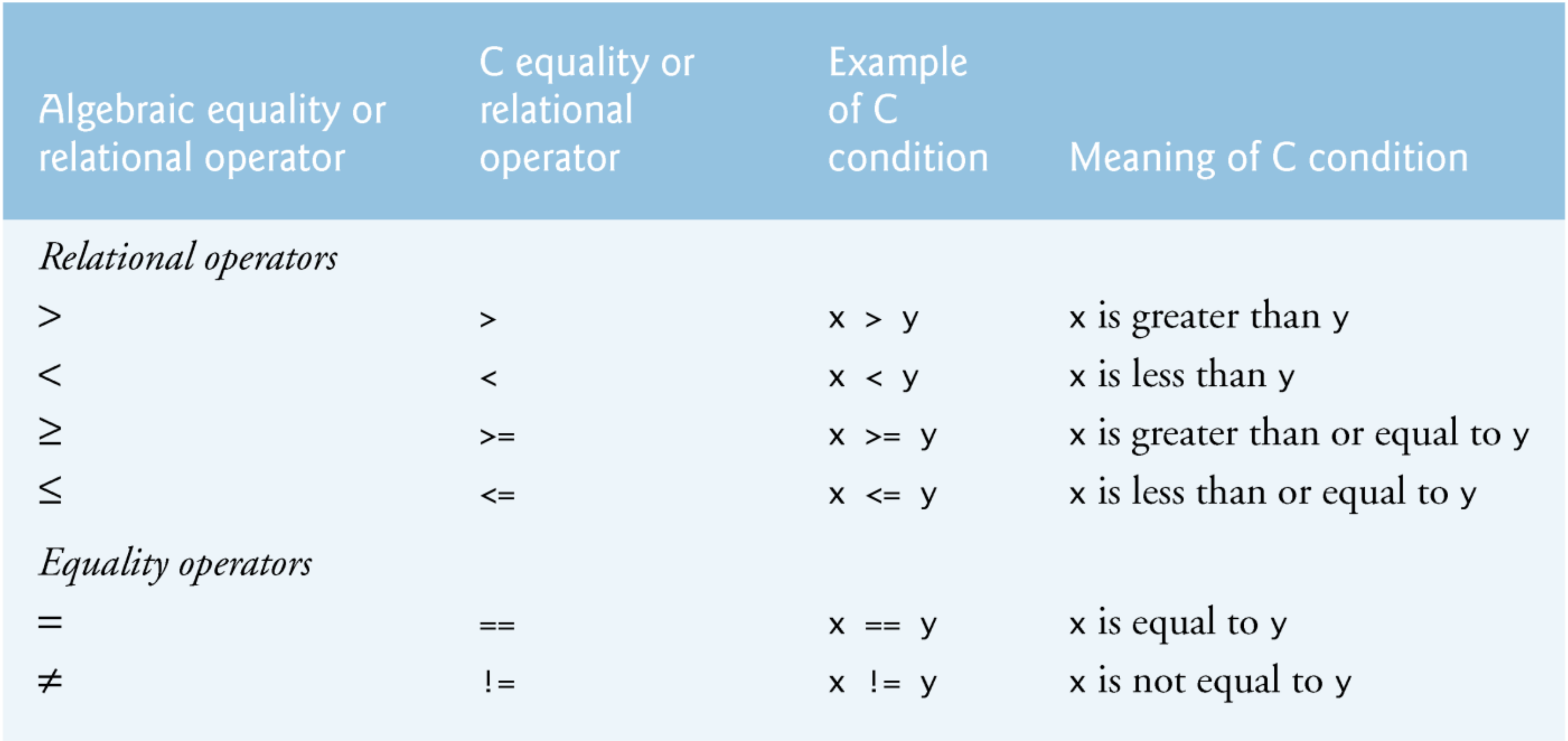
- If the condition is true (i.e., the condition is met) the statement in the body of the `if` statement is executed.
- If the condition is false (i.e., the condition isn’t met) the body statement is not executed.
- Whether the body statement is executed or not, after the `if` statement completes, execution proceeds with the next statement after the `if` statement.
- Conditions in `if` statements are formed by using the equality operators and relational operators.

If Statement



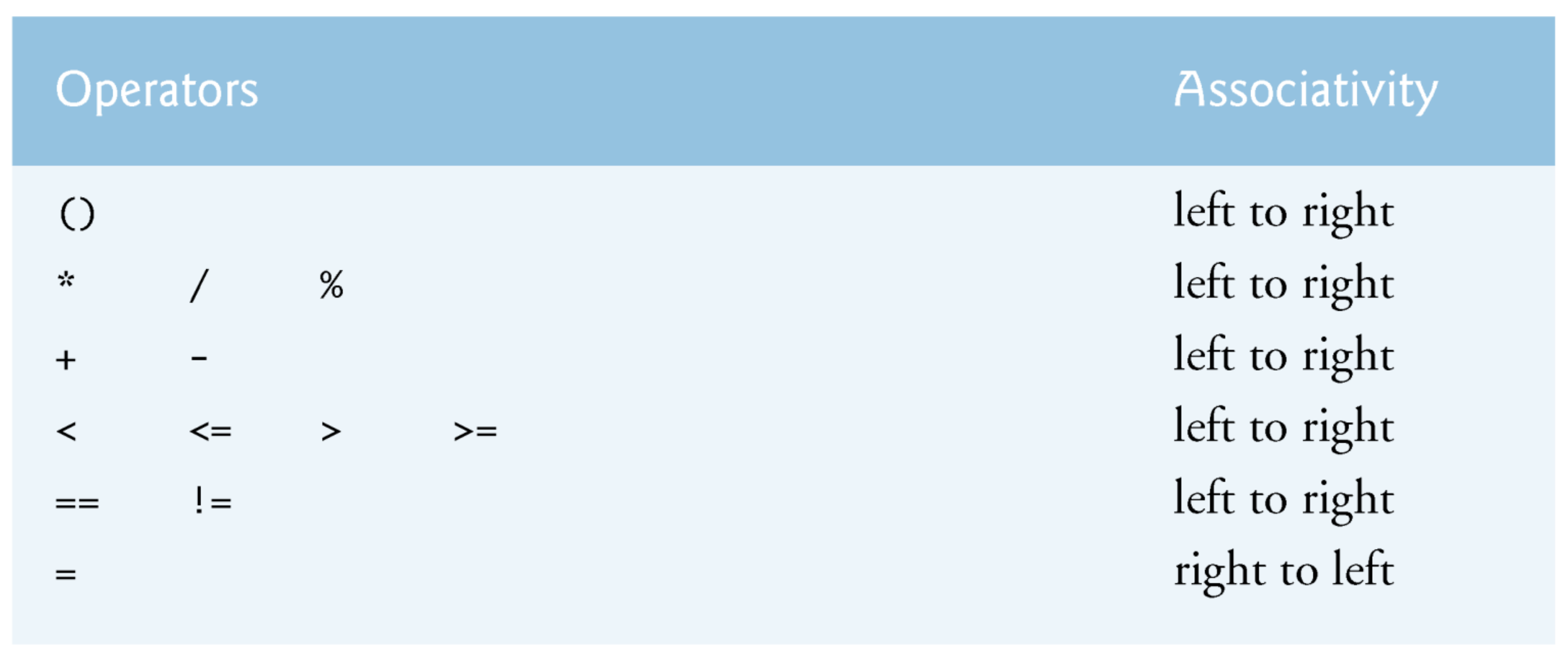
```c
#include <stdio.h>

int main ( void )
{
    int integer1 = 5;
    int integer2 = 10;
    if (integer1 > integer2)
    {
        printf("This statement is not printed if the condition is False\n");
    }
    printf("This statement is always executed as it is outside the if statement\n");
}
```

Relational & Equality Operators



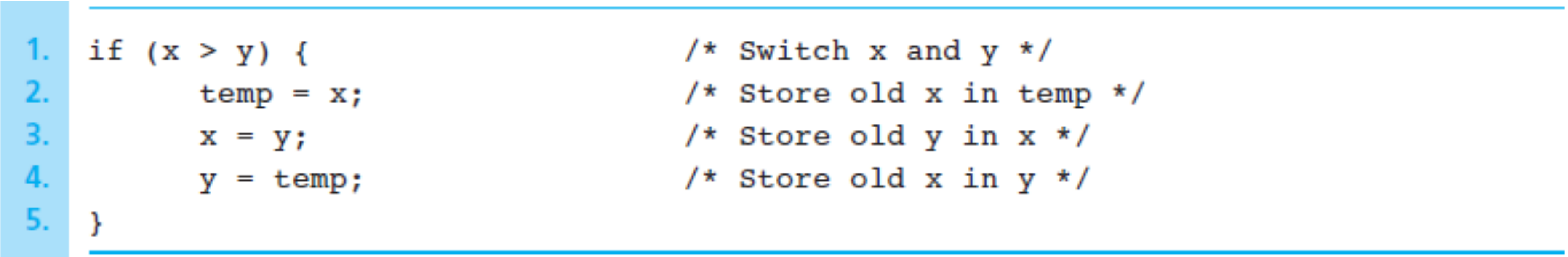
Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Relational operators			
>	`>`	`x > y`	x is greater than y
<	`<`	`x < y`	x is less than y
≥	`>=`	`x >= y`	x is greater than or equal to y
≤	`<=`	`x <= y`	x is less than or equal to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

Precedence of Operators



Operators	Associativity
()	left to right
* / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
=	right to left

Example: Swap values of two variables



```
if (x > y) {                    /* Switch x and y */
      temp = x;                 /* Store old x in temp */
      x = y;                    /* Store old y in x */
      y = temp;                 /* Store old x in y */
}
```

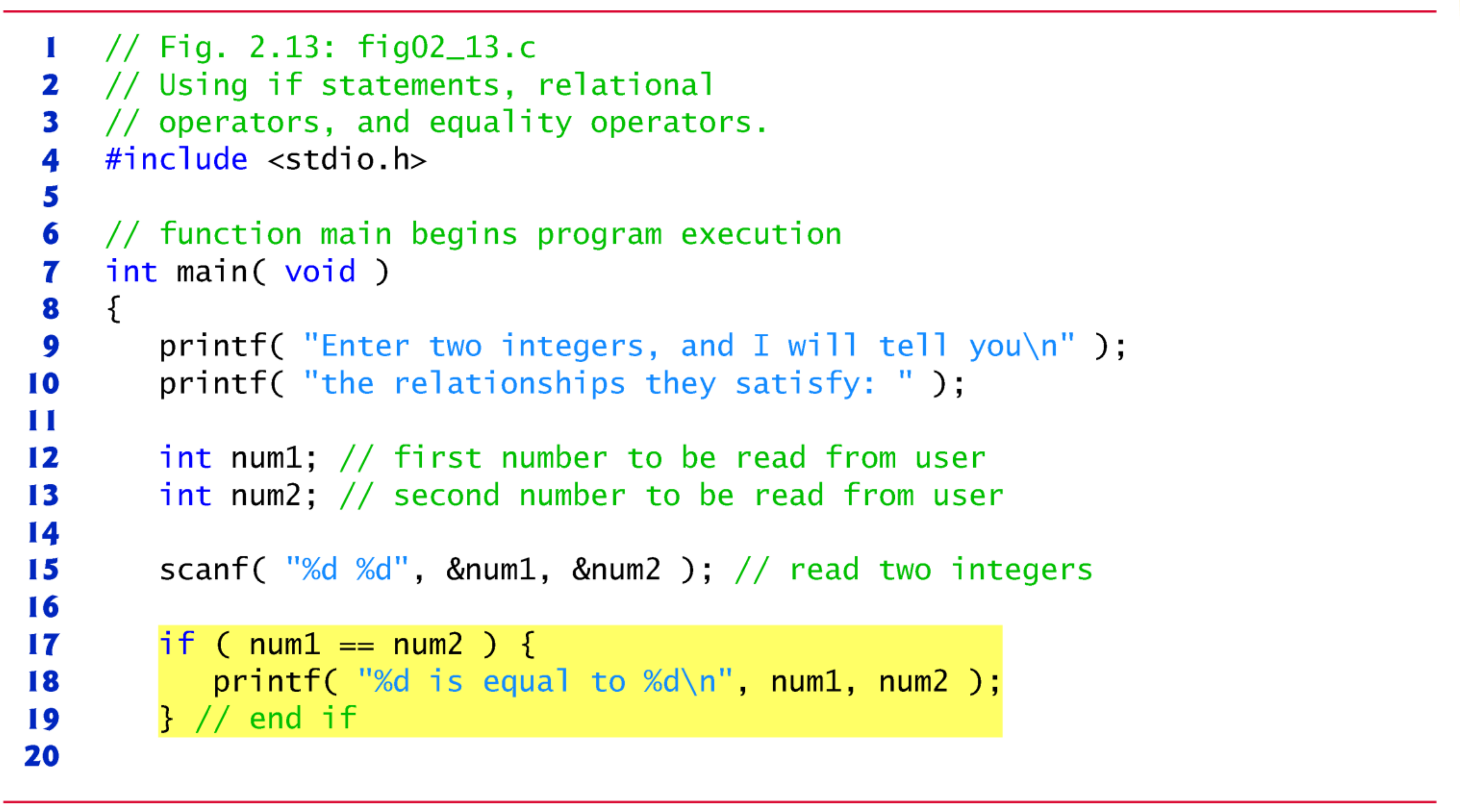
Conditional Operator (?)

- C’s only ternary operator—it takes *three* operands.
- The first operand is a *condition*.
- The second operand is the value for the entire conditional expression if the condition is *TRUE.*
- The third operand is the value for the entire conditional expression if the condition is *FALSE*.
- Example:
  - `printf( grade >= 60 ? "Passed" : "Failed" );`

Classroom Discussion

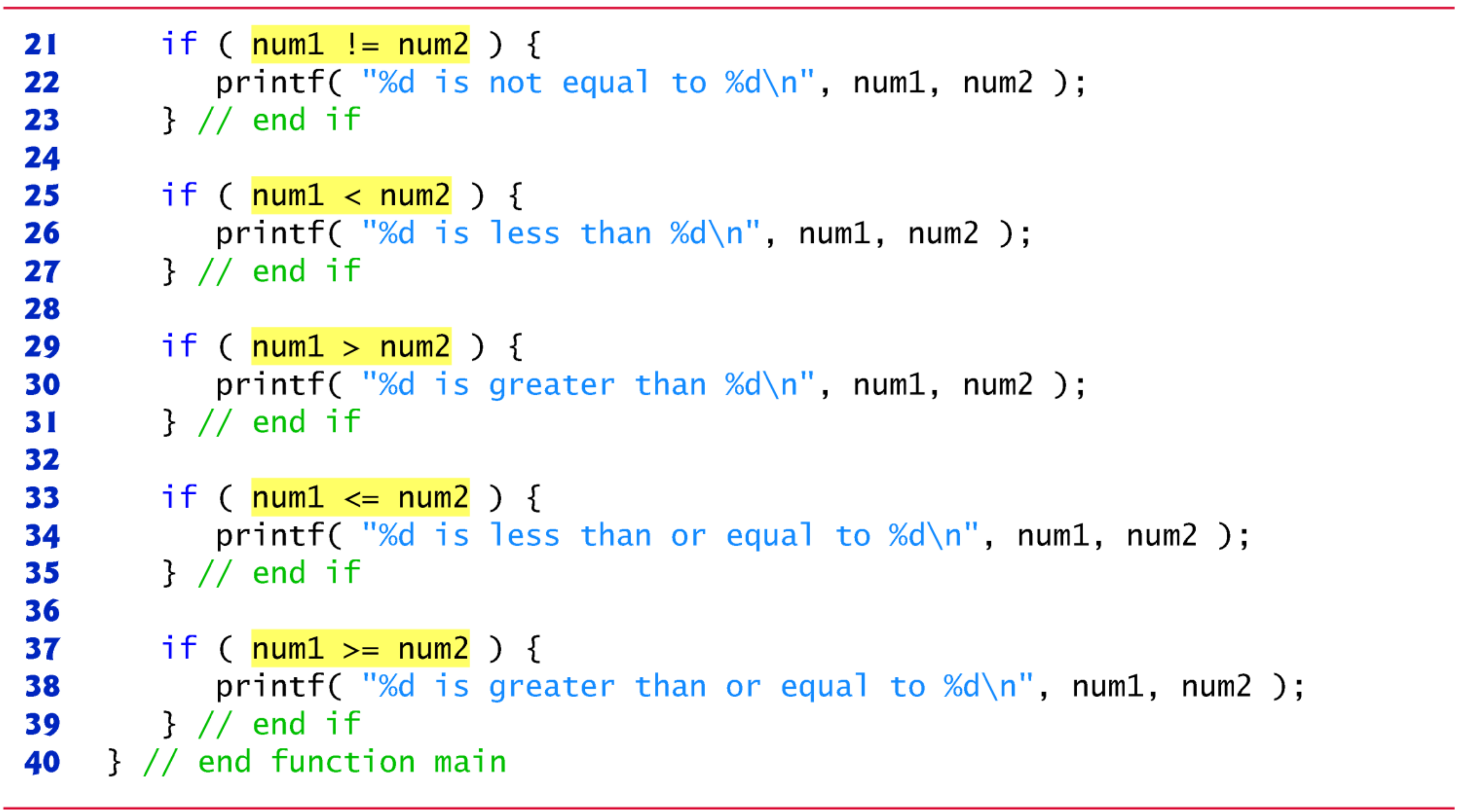
- Develop an algorithm to find a number is odd or even
- Write a pseudocode to check if a number is odd or even
- Write a C code that takes an integer as input from the user and prints out whether it is odd or even number

Example C Program



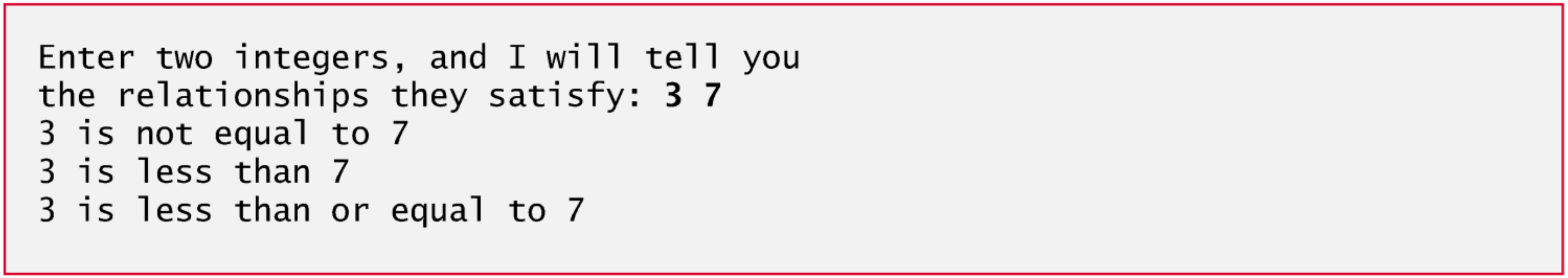
```c
// Fig. 2.13: fig02_13.c
// Using if statements, relational
// operators, and equality operators.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );

   int num1; // first number to be read from user
   int num2; // second number to be read from user

   scanf( "%d %d", &num1, &num2 ); // read two integers

   if ( num1 == num2 ) {
      printf( "%d is equal to %d\n", num1, num2 );
   } // end if

```

Example C Program... continued

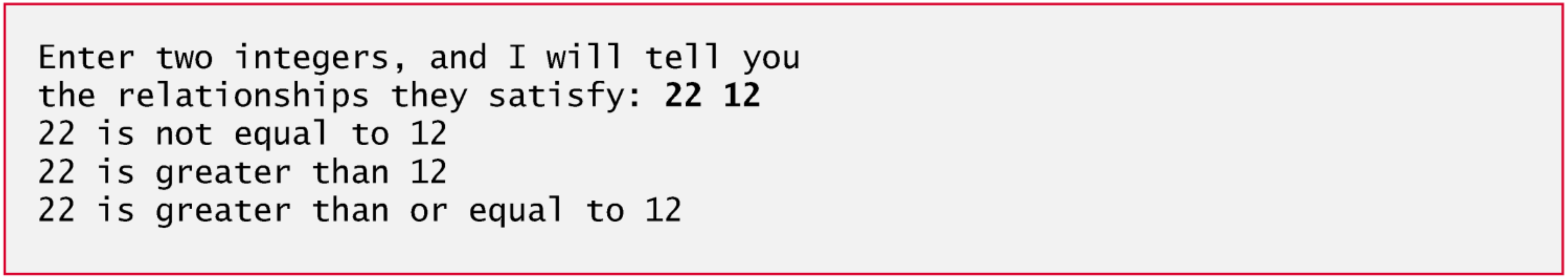


```c
    if ( num1 != num2 ) {
        printf( "%d is not equal to %d\n", num1, num2 );
    } // end if

    if ( num1 < num2 ) {
        printf( "%d is less than %d\n", num1, num2 );
    } // end if

    if ( num1 > num2 ) {
        printf( "%d is greater than %d\n", num1, num2 );
    } // end if

    if ( num1 <= num2 ) {
        printf( "%d is less than or equal to %d\n", num1, num2 );
    } // end if

    if ( num1 >= num2 ) {
        printf( "%d is greater than or equal to %d\n", num1, num2 );
    } // end if
} // end function main
```

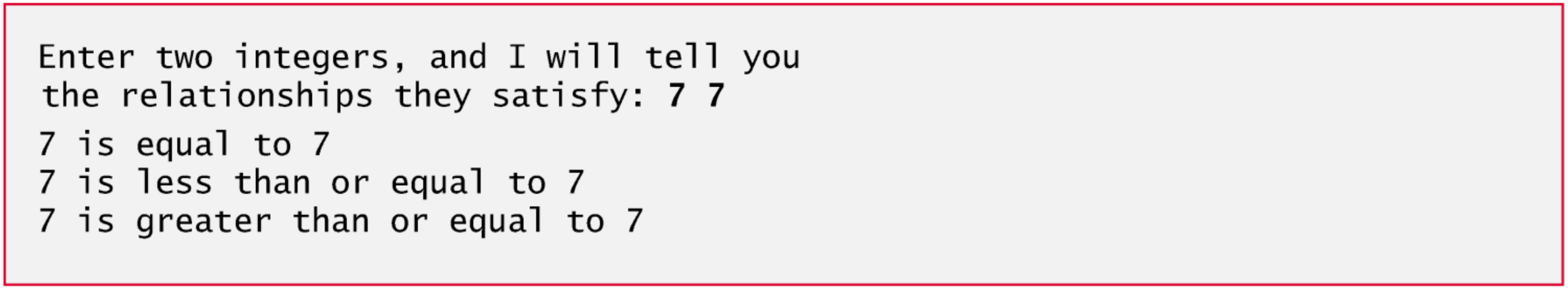
Example C Program …. Output



```
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
```



```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
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 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

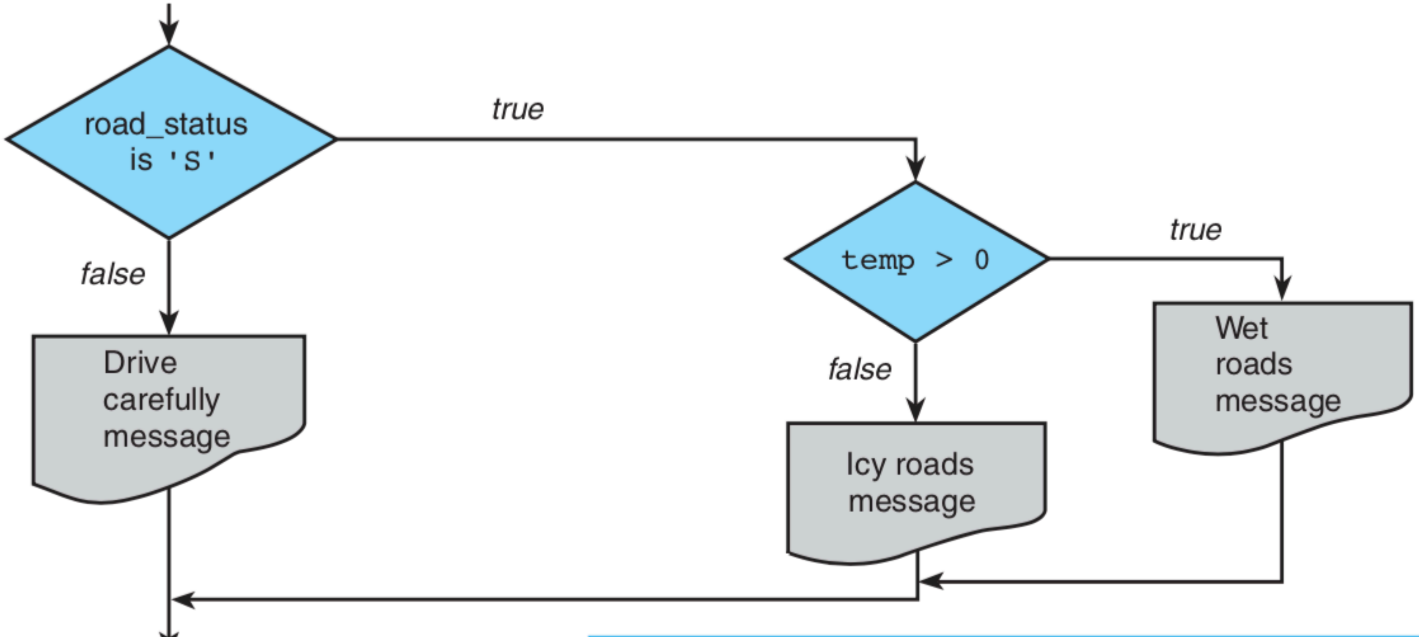
Nested if... else Statements

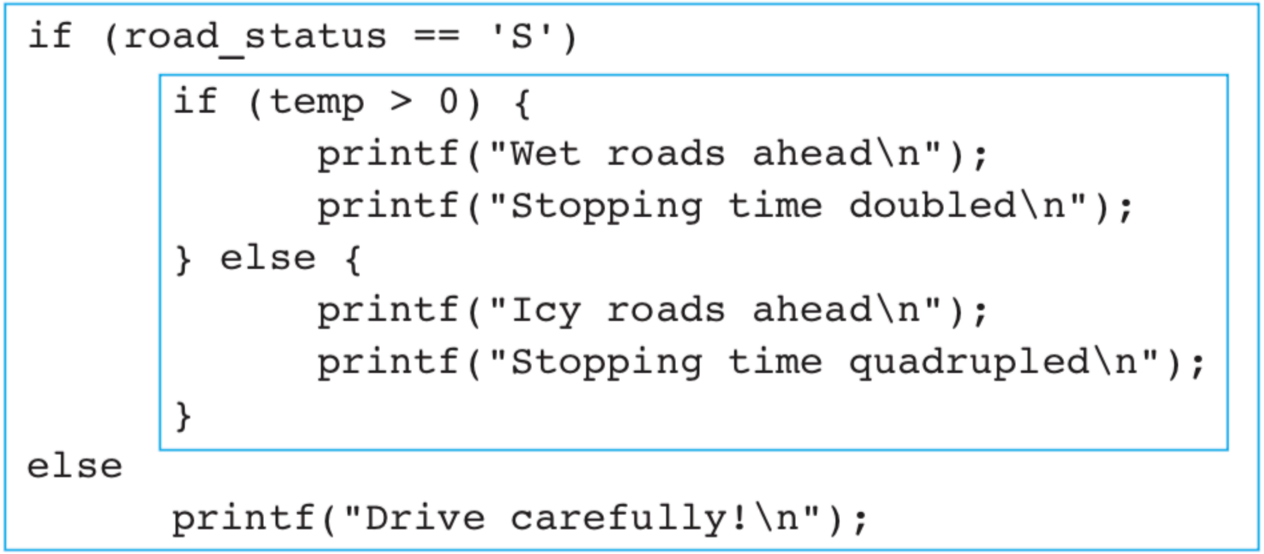
```
if ( grade >= 90 )
  puts( "A" );
else
  if ( grade >= 80 )
    puts("B");
  else
    if ( grade >= 70 )
      puts("C");
    else
      if ( grade >= 60 )
        puts( "D" );
      else
        puts( "F" );
```

If … else if Statement

```
if ( grade >= 90 )
   puts( "A" );
else if ( grade >= 80 )
   puts( "B" );
else if ( grade >= 70 )
   puts( "C" );
else if ( grade >= 60 )
   puts( "D" );
else
   puts( "F" );
```

Nested if - Example





```
if (road_status == 'S')
      if (temp > 0) {
            printf("Wet roads ahead\n");
            printf("Stopping time doubled\n");
      } else {
            printf("Icy roads ahead\n");
            printf("Stopping time quadrupled\n");
      }
else
      printf("Drive carefully!\n");
```

UNIVERSITY AT ALBANY
State University of New York

Compound Statement

- The `if` selection statement expects only one statement in its body
- To include several statements in the body of an `if`, the set of statements are included in braces

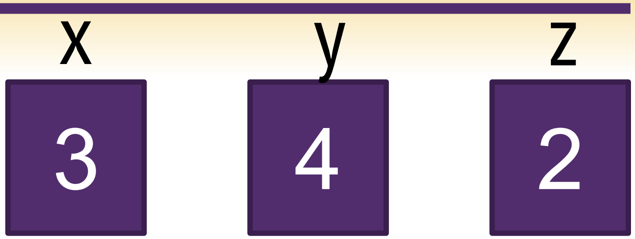
```
{
        statement;
        statement;
            .
            .
            .
        statement;
}
```

-
```
if ( grade >= 60 )
   puts( "Passed. " );
// end if
else {
   puts( "Failed. " );
   puts( ”Take this course again. " );
} // end else
```

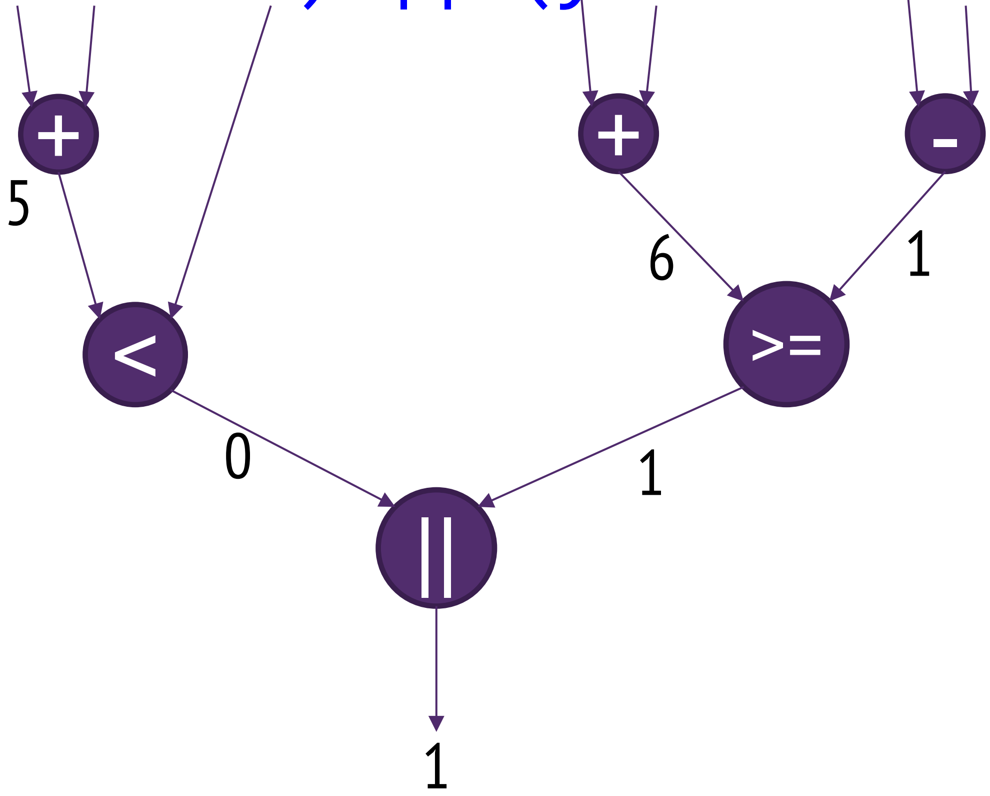
Precedence

Operator	Precedence
()	highest (evaluated first)
* / %	
+ -	
< <= >= >	
== !=	
&&	
||	
=	lowest (evaluated last)

Condition Statements

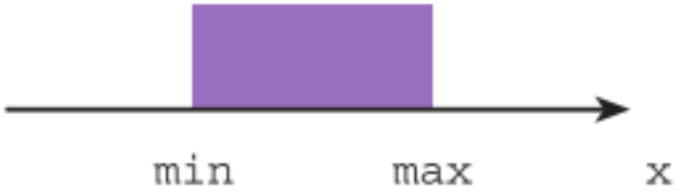


```
(z+x < 1) || (y+z >= x-z)
```

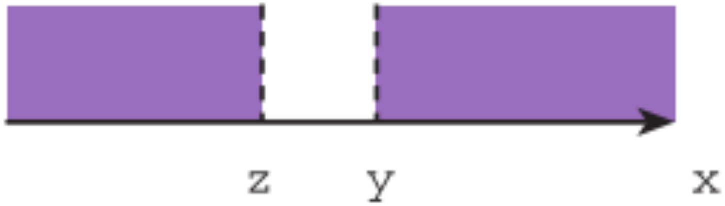


Common usage in program

- Check range of x



```
x >= min  &&  x <= max
```



```
x < z || x > y
```

English to C Logical Expression

English Condition
`x` and `y` are greater than `z`
`x` is equal to `1.0` or `3.0`
`x` is in the range `z` to `y`, inclusive
`x` is outside the range `z` to `y`

English to C Logical Expression

<table>
<tr><th>English Condition</th><th>Logical Expression</th><th>Evaluation</th></tr>
<tr><td>x and y are greater than z</td><td>x > z && y > z</td><td>1 && 1 is 1 (true)</td></tr>
<tr><td>x is equal to 1.0 or 3.0</td><td>x == 1.0 || x == 3.0</td><td>0 || 1 is 1 (true)</td></tr>
<tr><td>x is in the range z to y, inclusive</td><td>z <= x && x <= y</td><td>1 && 1 is 1 (true)</td></tr>
<tr><td>x is outside the range z to y</td><td>!(z <= x && x <= y)
z > x || x > y</td><td>!(1 && 1) is 0 (false)
0 || 0 is 0 (false)</td></tr>
</table>

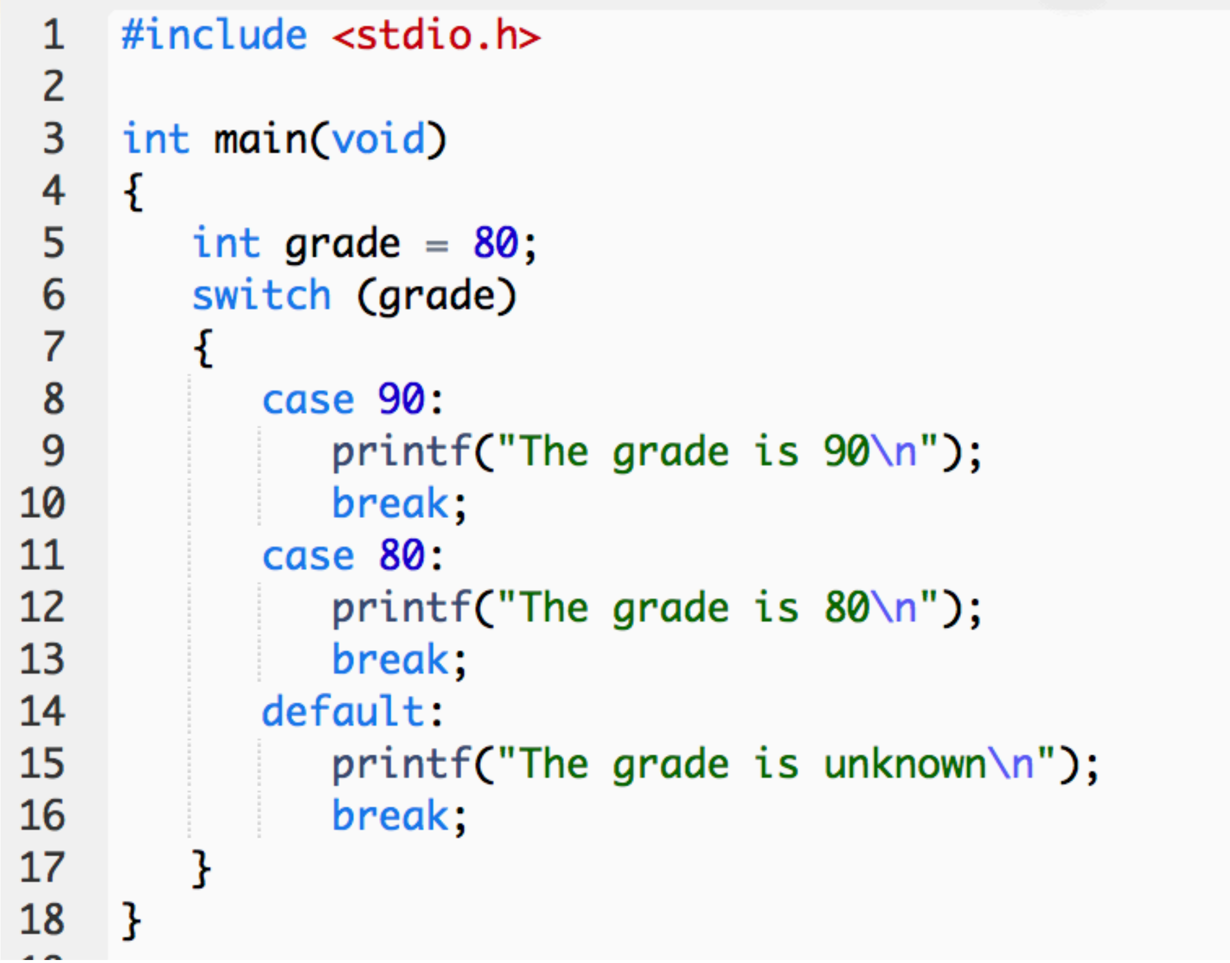
Switch Statement

- Used to select one of several alternatives
- useful when the selection is based on the value of
  - a single variable
  - or a simple expression
- values may be of type `int` or `char`
  - NOT double

switch Statement

```
switch (controlling expression) {
        label set$_1$
                statements$_1$
                break;
        label set$_2$
                statements$_2$
                break;
                .
                .
                .
        label set$_n$
                statements$_n$
                break;
```

switch Statement Example



```c
#include <stdio.h>

int main(void)
{
    int grade = 80;
    switch (grade)
    {
        case 90:
            printf("The grade is 90\n");
            break;
        case 80:
            printf("The grade is 80\n");
            break;
        default:
            printf("The grade is unknown\n");
            break;
    }
}
```

Iteration Statement

- Repeat a set of actions while some condition remains TRUE
- Example:
  - *While there are more students in class raising their hand* Condition
    Answer a question and let him/her lower the hand Action
  - *While there are more items on my shopping list*
    Purchase next item and cross it off my list
- Usually, action statements modify variables that affect the condition
- CAUTION: Check when the condition becomes FALSE
- Can be single statement or multiple (block)

While Statement in C

```
while (condition)
{
…
}
```

While Statement in C

```
while (condition)
{
…
}
```

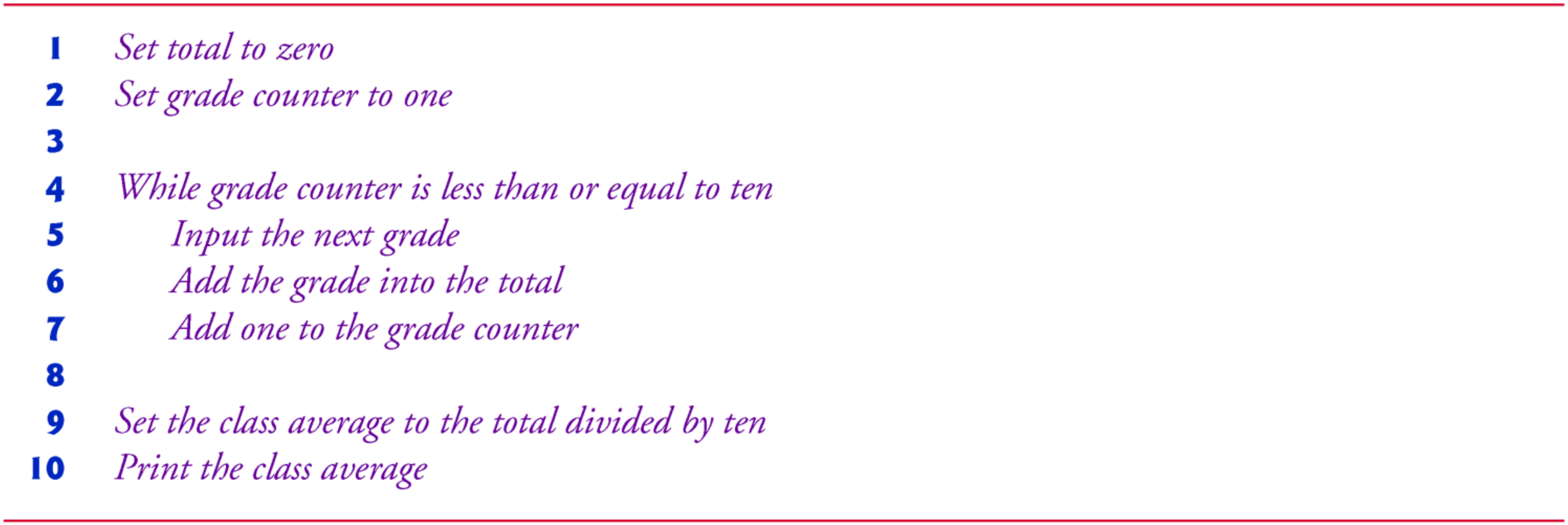
- Previously used Equation: ax^3+7
- Code to compute x^3

```
times=1; product =1;
while ( times <= 3 ) {
  product = x * product;
       times = times+1;
} // end while
```

Counter controlled iteration

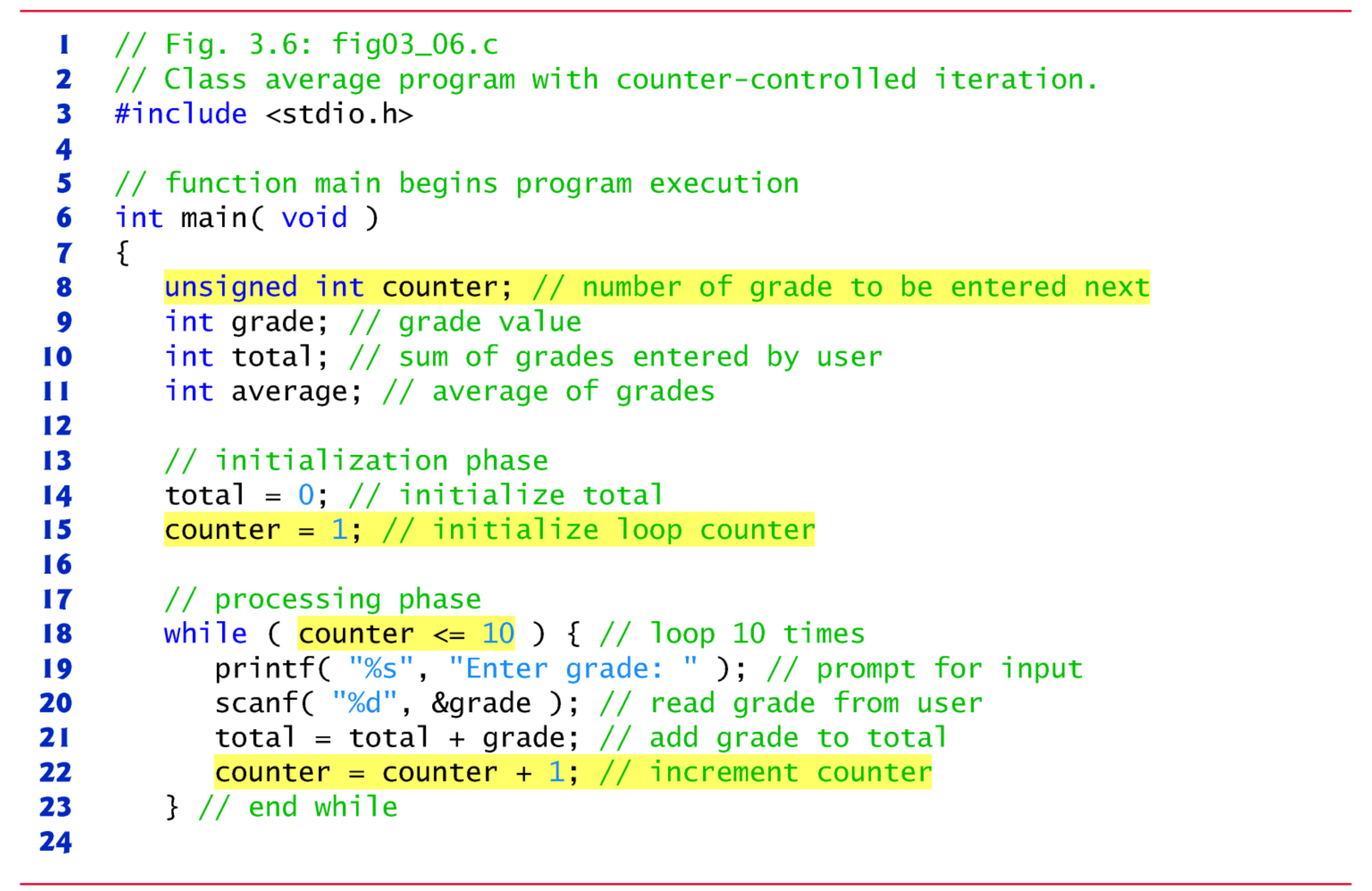
- Uses a variable called a counter to specify the number of times a set of statements should execute.
- Counter-controlled iteration is often called definite iteration because the number of iterations is known *before* the loop begins executing.

Counter controlled iteration - pseudocode



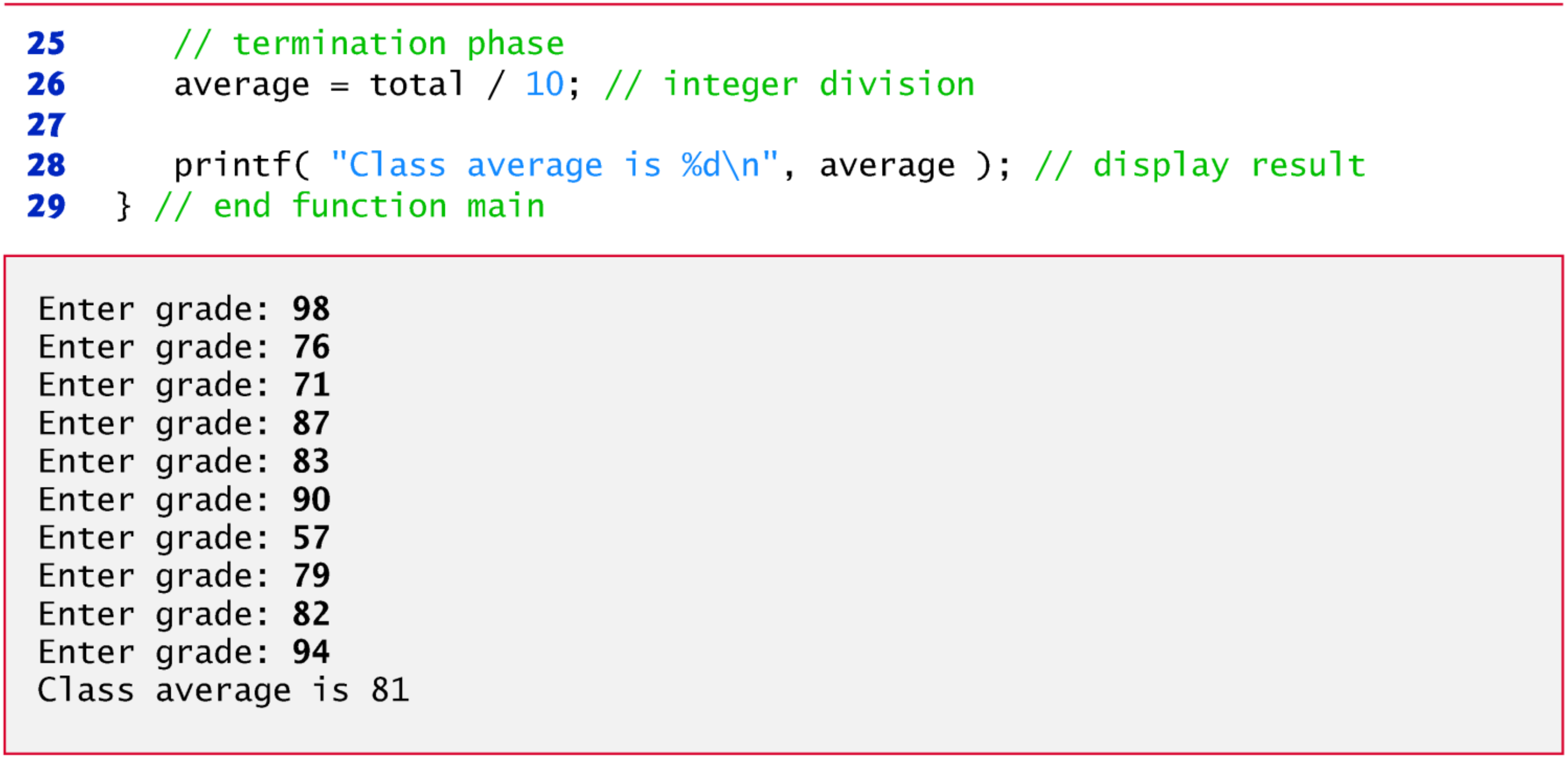
```
Set total to zero
Set grade counter to one

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
```

Counter controlled iteration – C code



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

// function main begins program execution
int main( void )
{
   unsigned int counter; // number of grade to be entered next
   int grade; // grade value
   int total; // sum of grades entered 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( "%s", "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

```

Counter controlled iteration – C code continued



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

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

Initialization phase

- A total is a variable used to accumulate the sum of a series of values.
- A counter is a variable used to count—in this case, to count the number of grades entered.
- An uninitialized variable contains a “garbage” value—the value last stored in the memory location reserved for that variable.

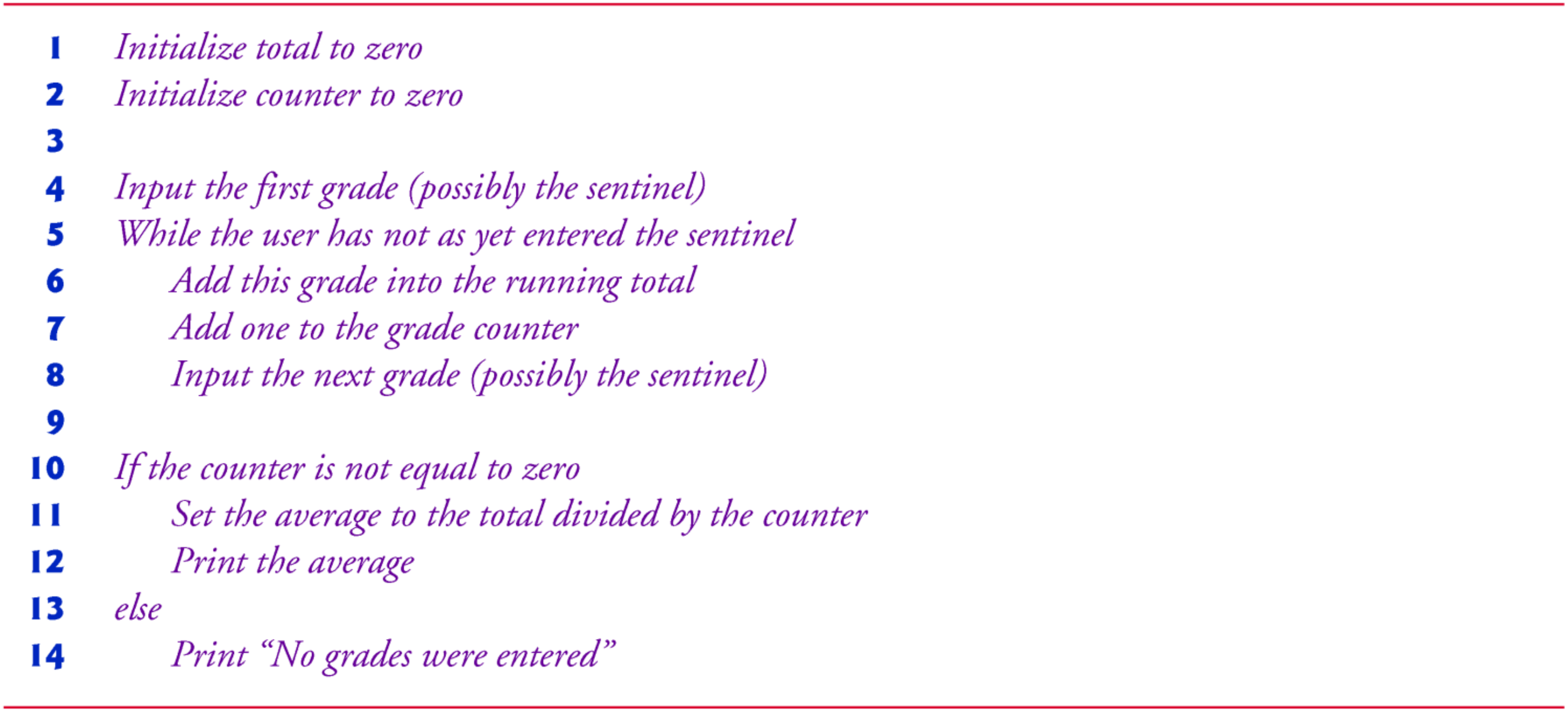
Formulating algorithm – Sentinel Controlled Iteration

- Consider the following problem:
  - *Develop a class-average program that will process an arbitrary number of grades each time the program is run.*
- In this example, the program must process an *arbitrary number* of grades.

Sentinel Controlled Iteration

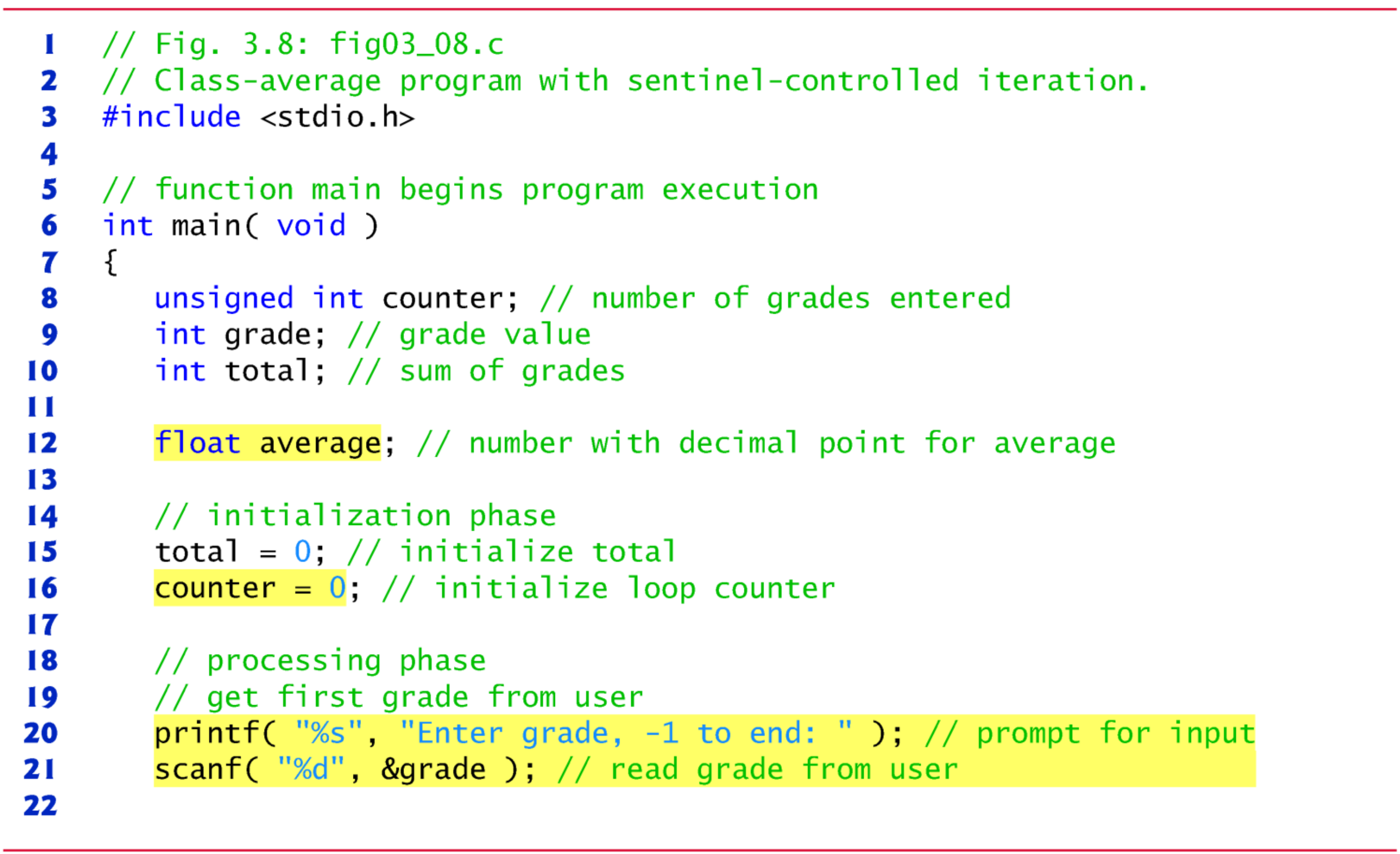
- Use a special value called a sentinel value (also called a signal value, a dummy value, or a flag value) to indicate “end of data entry.”
- Sentinel-controlled iteration is often called indefinite iteration because the number of iterations isn’t known before the loop begins executing.
- Sentinel value must be chosen so that it cannot be confused with an acceptable input value.

Sentinel controlled iteration - pseudocode



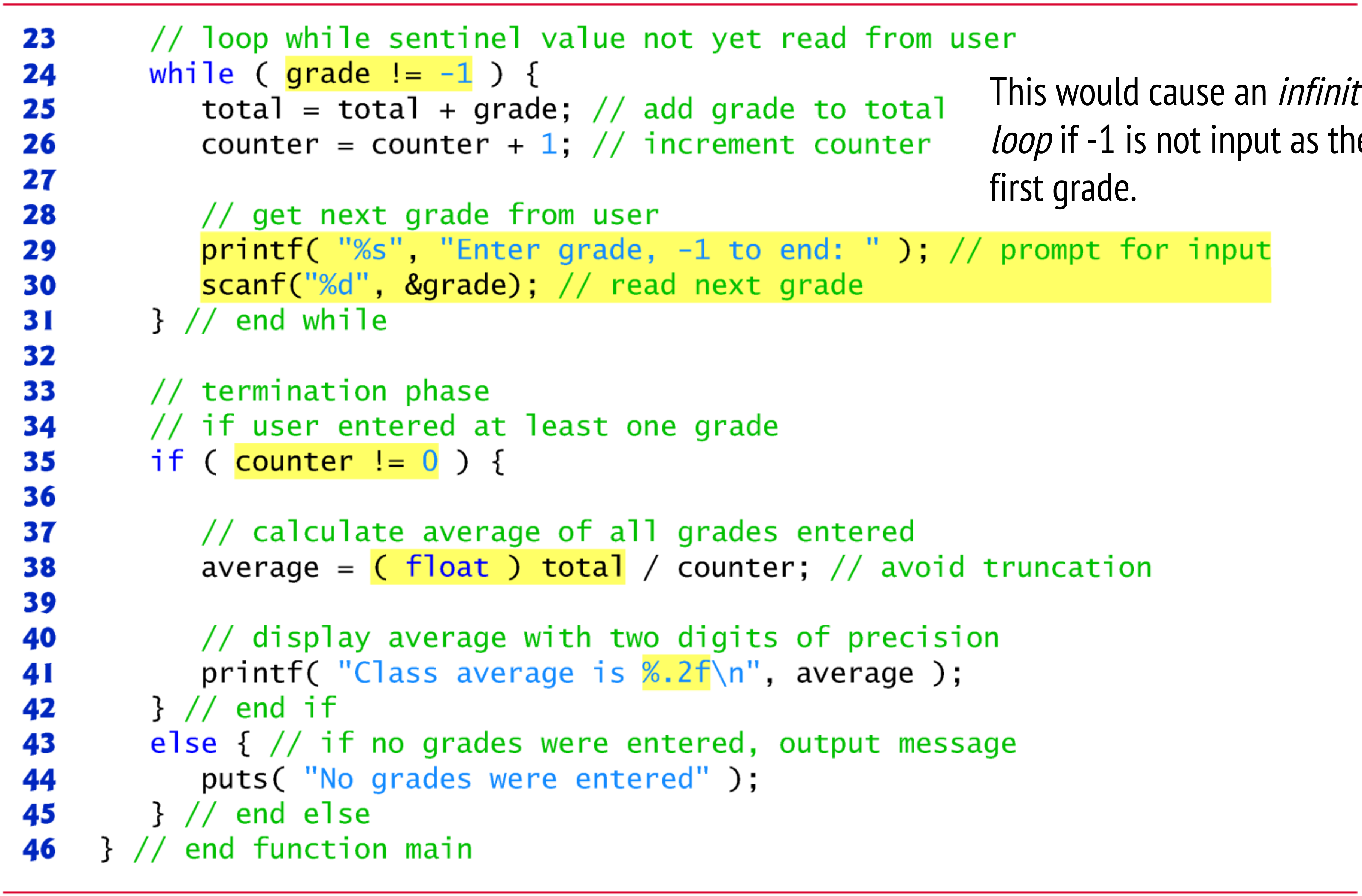
```
Initialize total to zero
Initialize counter to zero

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)

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"
```

Sentinel controlled iteration – C code



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

// function main begins program execution
int main( void )
{
   unsigned 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( "%s", "Enter grade, -1 to end: " ); // prompt for input
   scanf( "%d", &grade ); // read grade from user

```

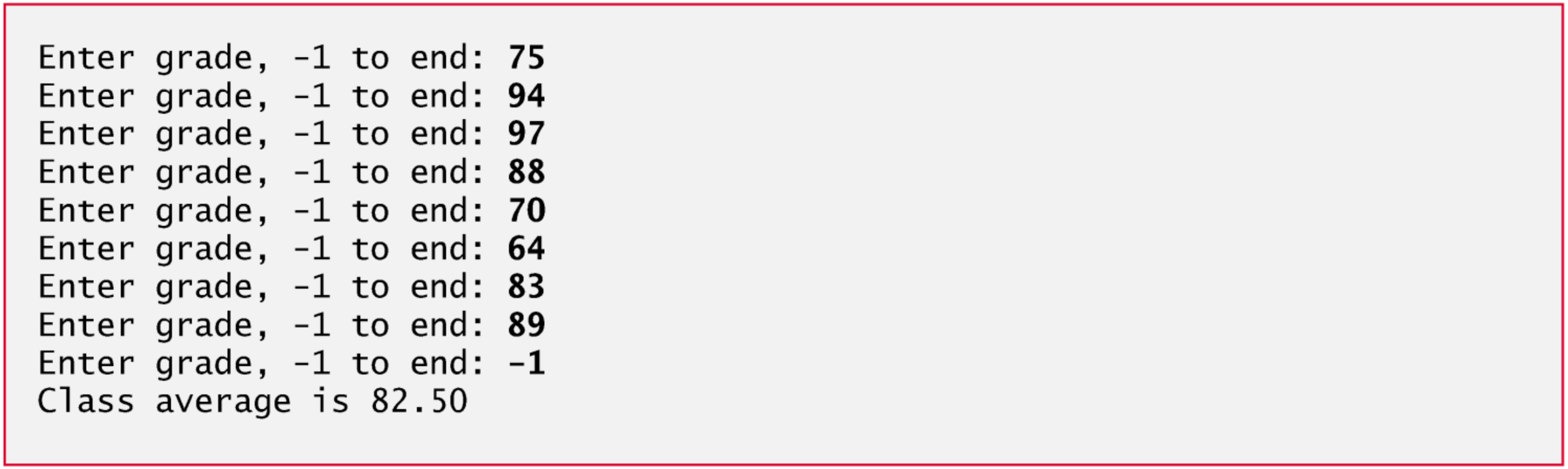
Sentinel controlled iteration – C code



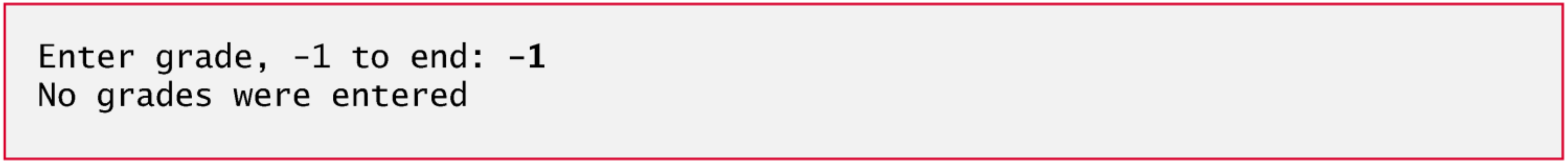
```c
   // loop while sentinel value not yet read from user
   while ( grade != -1 ) {
      total = total + grade; // add grade to total
      counter = counter + 1; // increment counter

      // get next grade from user
      printf( "%s", "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; // avoid truncation

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

This would cause an *infinite loop* if -1 is not input as the first grade.

Sentinel controlled iteration - 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
```

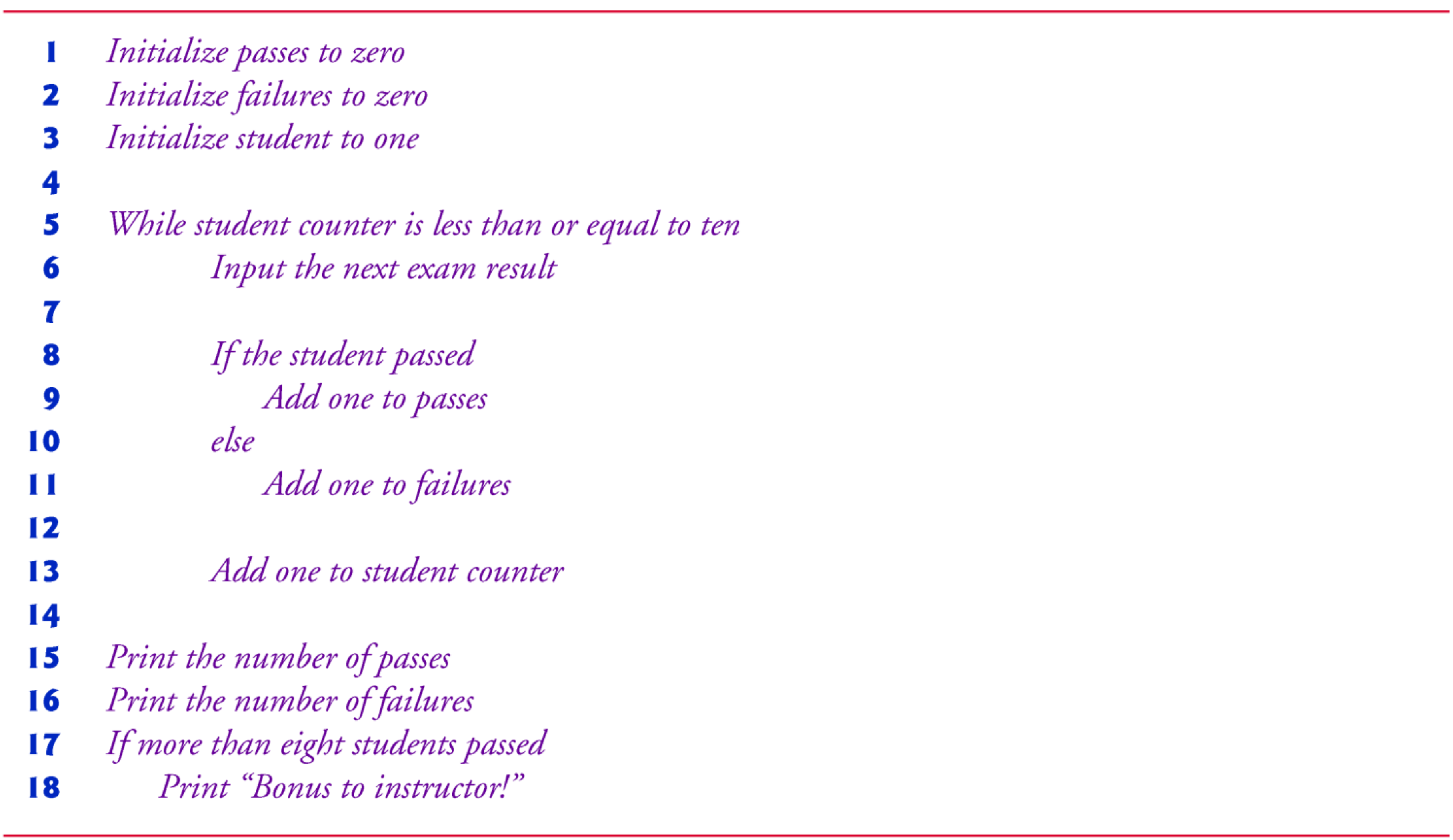


```
Enter grade, -1 to end: -1
No grades were entered
```

Nested Control Statement

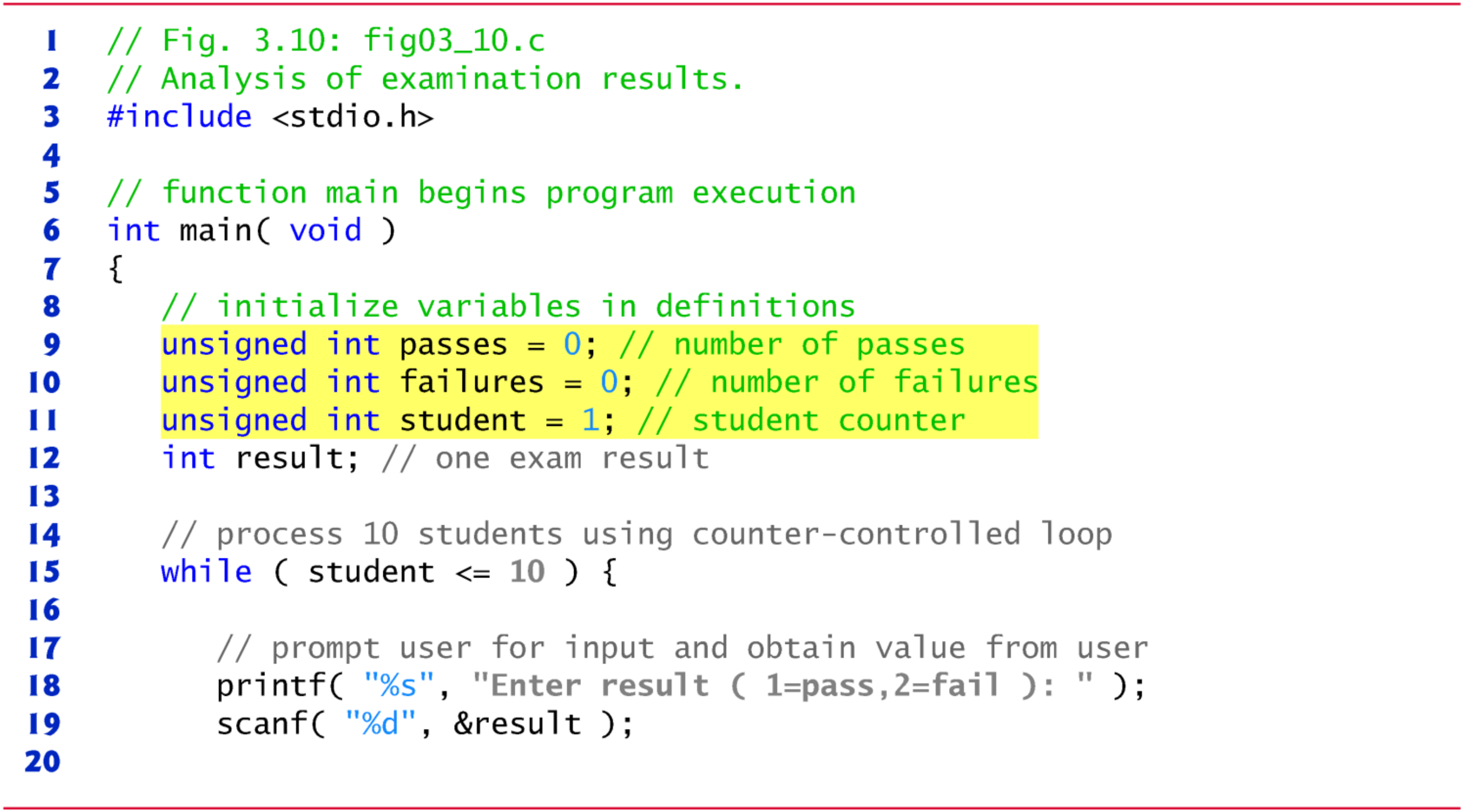
- One control statement within another
- Consider the following problem statement:
  - *In a class of 10 students, get the result of the student from the user (1=pass, 2=fail) and find the number of students who failed and who passed. If more than 8 students passed, print a statement for bonus to the instructor.*

Nested Control Statement – Pseudocode



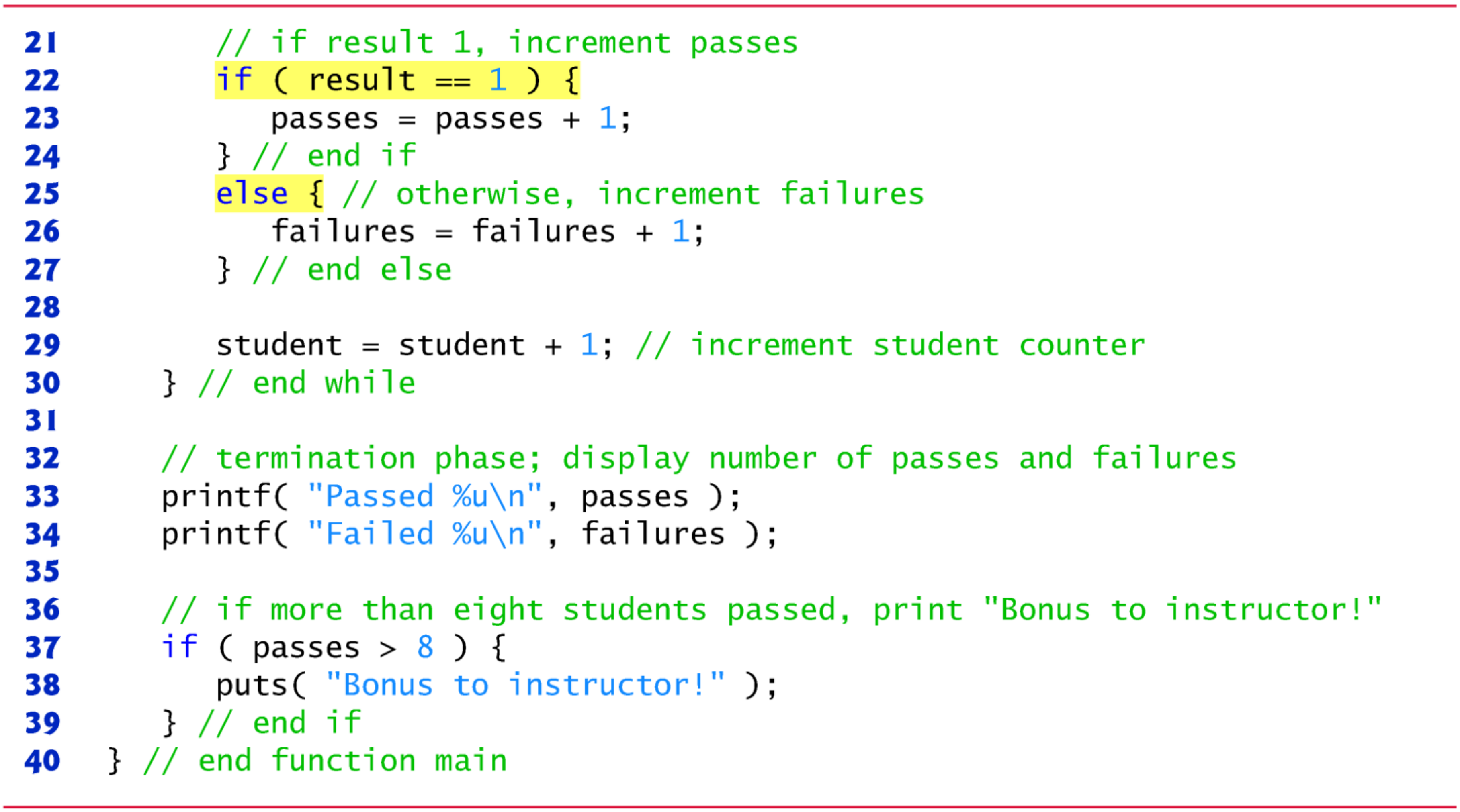
```
Initialize passes to zero
Initialize failures to zero
Initialize student to one

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 "Bonus to instructor!"
```

Nested Control Statement – C code



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

// function main begins program execution
int main( void )
{
   // initialize variables in definitions
   unsigned int passes = 0; // number of passes
   unsigned int failures = 0; // number of failures
   unsigned 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( "%s", "Enter result ( 1=pass,2=fail ): " );
      scanf( "%d", &result );

```

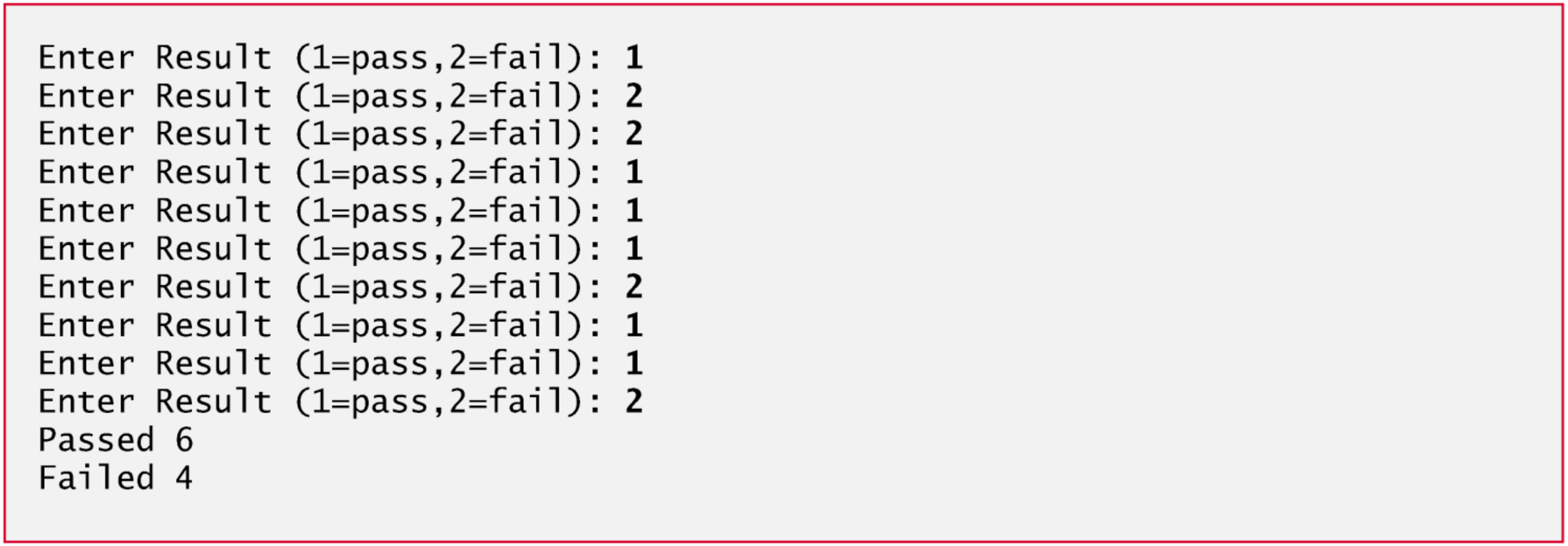
Nested Control Statement – C code



```c
         // if result 1, increment passes
         if ( result == 1 ) {
            passes = passes + 1;
         } // end if
         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 %u\n", passes );
      printf( "Failed %u\n", failures );

      // if more than eight students passed, print "Bonus to instructor!"
      if ( passes > 8 ) {
         puts( "Bonus to instructor!" );
      } // end if
   } // end function main
```

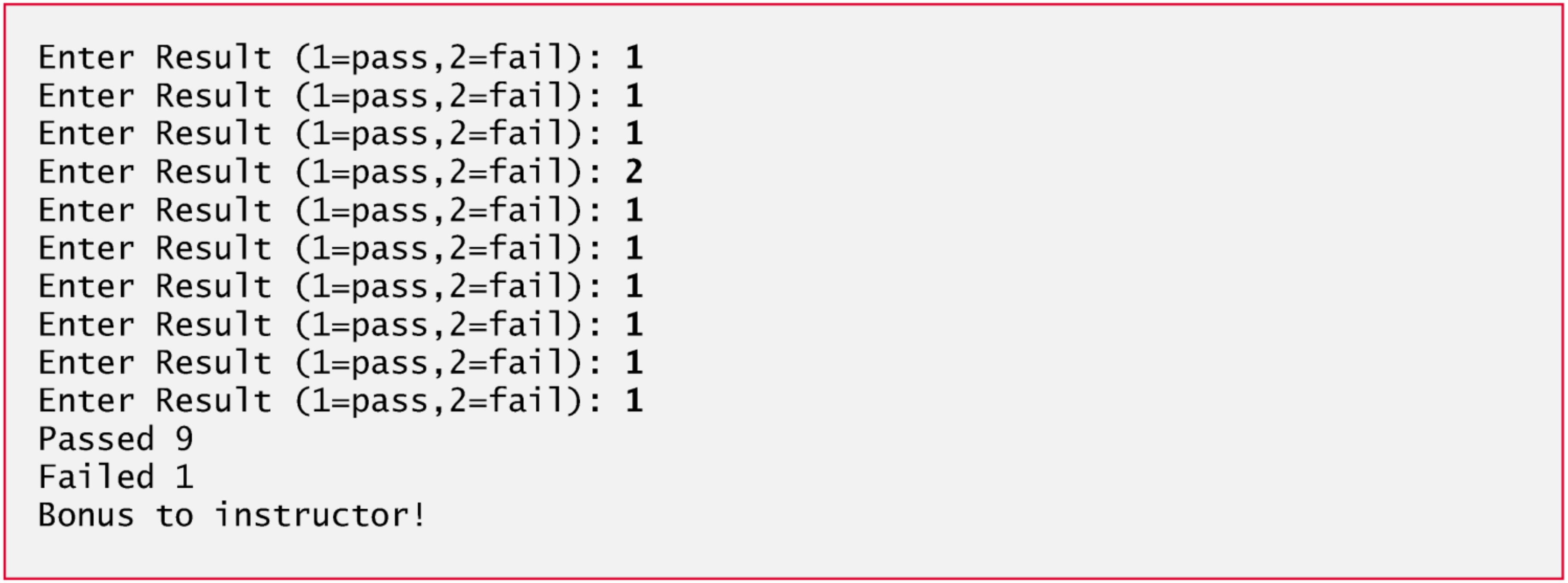
UNIVERSITY AT ALBANY
State University of New York

Nested Control Statement – Output 1



```
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
```

Nested Control Statement – Output 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): 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
Bonus to instructor!
```