

Chapter 3 - Structured Program Development

Outline

- 3.1 Introduction
- 3.2 Algorithms
- 3.3 Pseudocode
- 3.4 Control Structures
- 3.5 The If Selection Statement
- 3.6 The If...Else Selection Statement
- 3.7 The While Repetition Statement
- 3.8 Formulating Algorithms: Case Study 1 (Counter-Controlled Repetition)
- 3.9 Formulating Algorithms with Top-down, Stepwise Refinement: Case Study 2 (Sentinel-Controlled Repetition)
- 3.10 Formulating Algorithms with Top-down, Stepwise Refinement: Case Study 3 (Nested Control Structures)
- 3.11 Assignment Operators
- 3.12 Increment and Decrement Operators



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

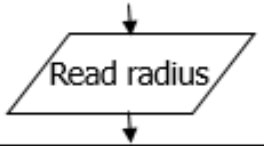
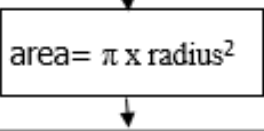
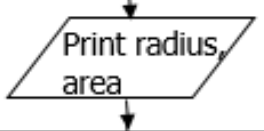
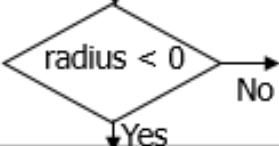
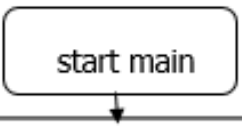


Algorithm Development

- **Decomposition outlines** provide the first definition of a problem solution.
- This outline is written sequential steps and can be shown in a diagram or a step-by-step outline.
- For very simple problems we can go from the decomposition outline directly to the C statements.
- However, for most problems the decomposition outline needs to be refined into a description with more detail by breaking the problem solution into smaller and smaller portions.
- The refinement of an outline into more detailed steps can be done with pseudocode or a flowchart.
- **Pseudocode** uses English-like statements to describe the steps in an algorithm; a **flowchart** uses a diagram to describe the steps in an algorithm.



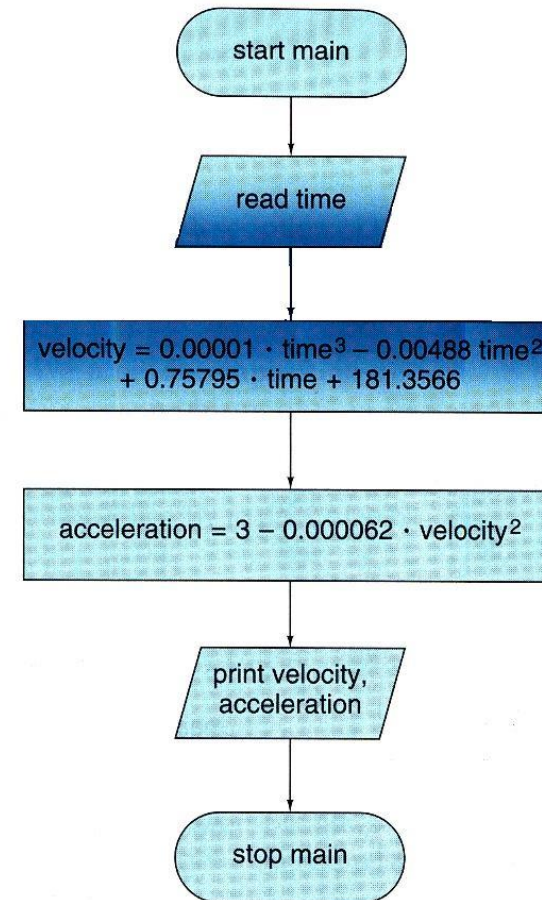
Algorithm Development

Basic Operation	Pseudocode Notation	
Input	Read radius	
Computation	Set area to $\pi \times \text{radius}^2$	
Output	Print radius, area	
Comparisons	if radius < 0 then ...	
Beginning of algorithm	main:	
End of algorithm		



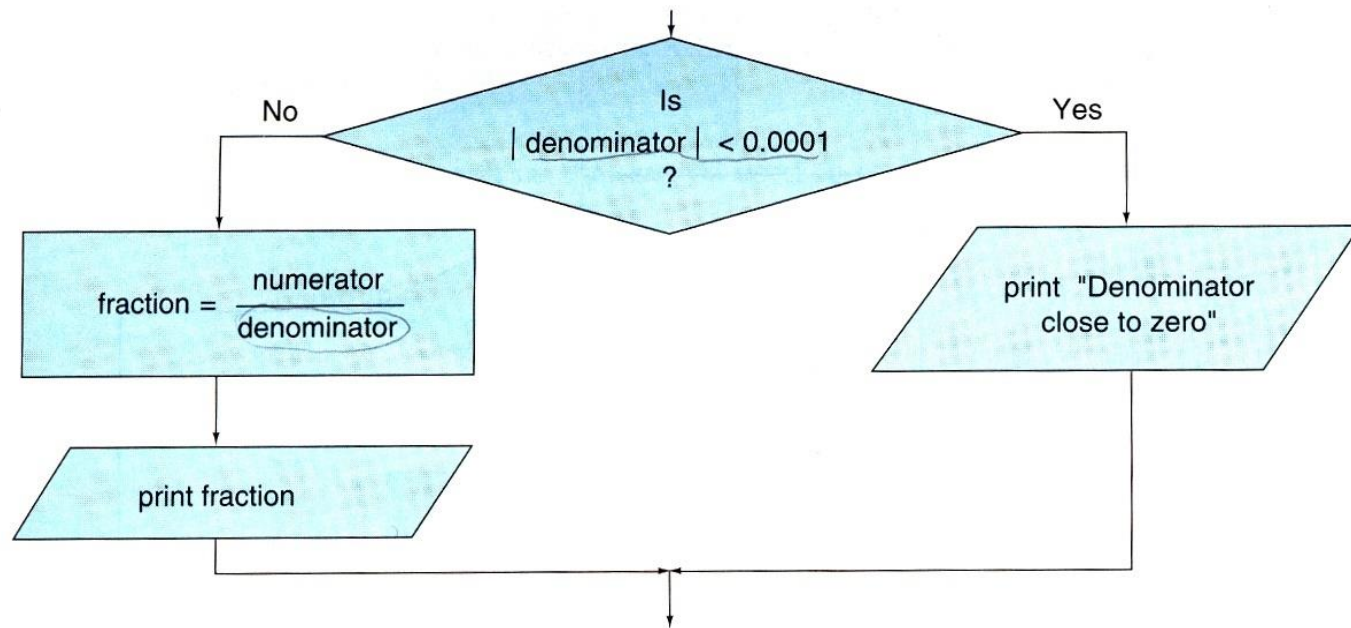
Structured Program

- A **structured program** is one written using simple control structures,
 - sequence,
 - selection
 - repetitionto organize the solution into a problem.
- A **sequence structure** contains steps that are performed one after another.



Structured Program

- A **selection structure** contains one set of steps that is performed if a condition is true and another set of steps that is performed if the condition is false.



Structured Program

- A **repetition structure** contains a set of steps that is repeated as long as a condition is true.

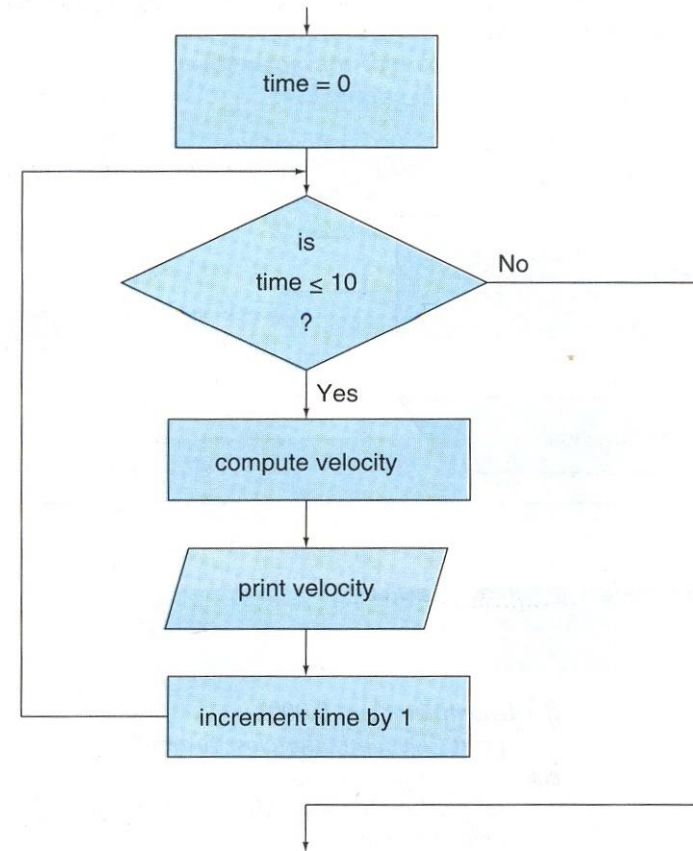
set time to 0

while time ≤ 10

 compute velocity

 print velocity

 increment time by 1



3.4 Control Structures

- Sequential execution
 - Statements executed one after the other in the order written
- 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

- 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:

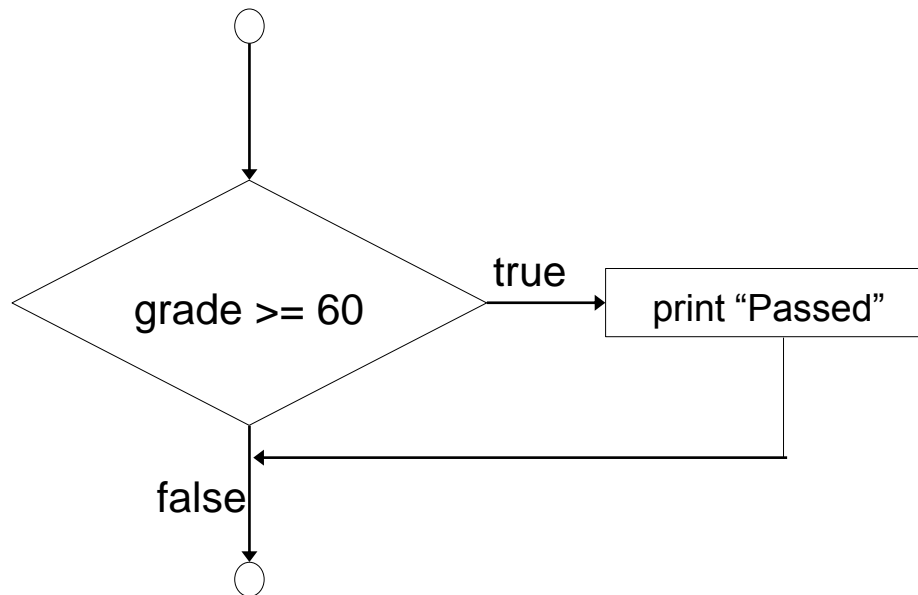
```
if ( grade >= 60 )  
    printf( "Passed\n" );
```

- 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



3.5 The if Selection Statement

- **General Form:** if the condition is true then statement 1 is executed. If the condition is false then statement 1 is skipped.

```
if (condition)
    statement 1;
```

- A **compound statement** or **block**, which is composed of a set of statements enclosed in braces, can also be used.

```
if (condition)
{
    statement 1;
    statement 2;
    ...
    statement n;
}
```

- if statements can also be nested:

Nested if Example:

```
if (a < 50)
{
    ++count;
    sum += a;
    if (b > a)
        b = 0;
}
```



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" );
```

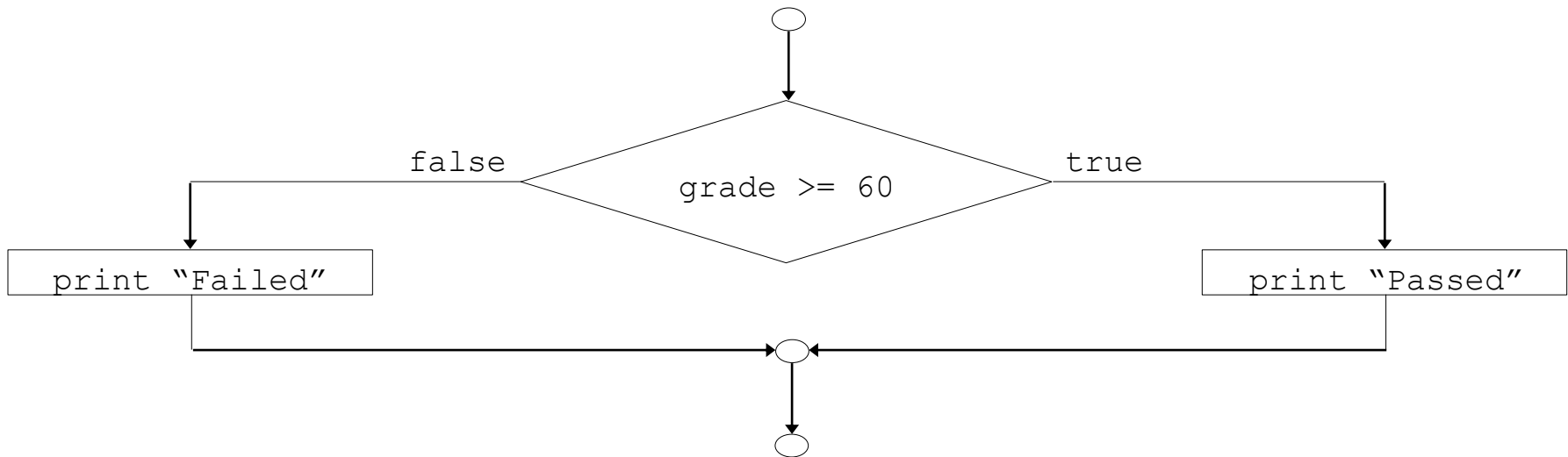
- Or it could have been written:

```
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



if/else Statement

- The if/else statement allows to execute one set of statements if a condition is true and a different set if the condition is false.

- General Form:**

```
if (condition)
    statement1;
else
    statement2;
```

- Statements 1 and 2 can also be an empty statement, which is a semicolon.

```
if (a < b)
    ;
else
    count++;
```

Nested if-else Example:

```
if (x > y)
    if (y < z)
        k++;
    else
        m++;
else
    j++;
```



if/else Statement

- Indentation is just for style and it does not change interpretation.

<pre>if (x > y) if (y < z) k++; else <u>j++;</u></pre>	<pre>if (x > y) if (y < z) k++; else j++;</pre>	<pre>if (x > y) { if (y < z) k++; } else <u>j++;</u></pre>
--	---	--



Good Programming Practice

- In general do not use equality operator when comparing floating-point numbers. For example, instead of comparing denominator to zero, the absolute value of the denominator can be compared with a very small value.
- **Example:**

```
if (fabs(denominator) < 0.0001)
    printf("Denominator of x is equal to zero\n");
else
{
    fraction = numerator/denominator;
    printf("fraction = %f \n",fraction);
}
```



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"



else-if Statement

- If's and else's can be used to construct logic that branches one of several ways and then rejoins, a common programming structure, in this way:

If (...)	
{...}	if(('A'<=c && c<='Z') ('a'<=c && c<='z'))
else if (...)	++let;
{...}	else if('0'<=c && c<='9')
else if(...)	++dig;
{...}	else
else	++other;
{...}	

- The conditions are tested in order, and exactly one block is executed; either the first one whose if is satisfied, or the one for the last else. When this block is finished, the next statement executed is the one after the last else. If no action is to be taken for the ``default" case, omit the last else.



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



Example

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     float grade;           //Declare variable for the User's entered grade
6
7     //Take the unser input for the grade
8     printf("What is your Grade? ");
9     scanf("%f",&grade);
10
11
12     //A set of if and else if statements compare the grades and takes the
13     //appropriate path
14     if(grade>=90)
15     {
16         printf("You get A. Excellent Work!\n");
17     }
18     else if(grade>=85 && grade<90)
19     {
20         printf("You get A-\n");
21     }
22     else if(grade>=80 && grade<85)
23     {
24         printf("You get B+\n");
25     }
26     else if(grade>=75 && grade<80)
27     {
28         printf("You get B\n");
29     }
30     else if(grade>=70 && grade<75)
31     {
```

Example-cont.

```
32         printf("You get B-\n");
33     }
34     else if(grade>=65 && grade<70)
35     {
36         printf("You get C+\n");
37     }
38     else if(grade>=60 && grade<65)
39     {
40         printf("You get C\n");
41     }
42     else if(grade>=55 && grade<60)
43     {
44         printf("You get C-\n");
45     }
46     else if(grade>=50 && grade<55)
47     {
48         printf("You get D\n");
49     }
50     else
51     {
52         printf("You get F. You have Failed!\n");
53     }
54
55
56     system("PAUSE");
57     return 0;
58 }
```



Logical Operators

- **Logical operators** can be used to compare conditions and generate new conditions.

Example:

$a < b \ \&\& \ b < c$ (The relational operators have higher precedence than the logical operator.)

Logical Operator	Symbol
not	!
and	&&
or	

A	B	A&&B	A B	!A	!B
False	False	False	False	True	True
False	True	False	True	True	False
True	False	False	True	False	True
True	True	True	True	False	False



3.7 The `while` Repetition Statement

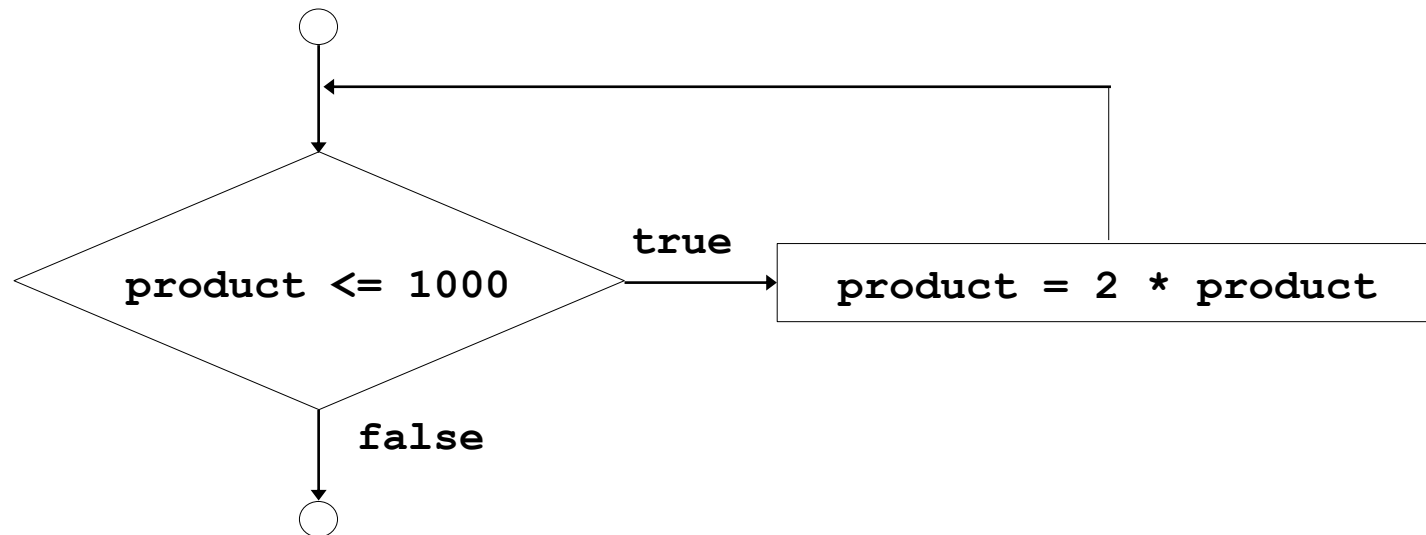
- Repetition structure
 - Programmer specifies an action to be repeated while some condition remains `true`
 - Psuedocode:
 - While there are more items on my shopping list*
Purchase next item and cross it off my list
 - `while` loop repeated until condition becomes `false`



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



**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
```



Outline



fig03_06.c (Part 2 of 2)

```
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
```

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”





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     }
```




Outline



fig03_08.c (Part 2 of 2)

```
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](#)**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"



**fig03_10.c (Part 1 of 2)**

```
1  /* Fig. 3.10: fig03_10.c
2     Analysis of examination results */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     /* initialize variables in definitions */
9     int passes = 0;    /* number of passes */
10    int failures = 0; /* number of failures */
11    int student = 1;   /* student counter */
12    int result;        /* one exam result */
13
14    /* process 10 students using counter-controlled loop */
15    while ( student <= 10 ) {
16
17        /* prompt user for input and obtain value from user */
18        printf( "Enter result ( 1=pass,2=fail ): " );
19        scanf( "%d", &result );
20
21        /* if result 1, increment passes */
22        if ( result == 1 ) {
23            passes = passes + 1;
24        } /* end if */
25    }
```



Outline



fig03_10.c (Part 2 of 2)

```
25     else { /* otherwise, increment failures */
26         failures = failures + 1;
27     } /* end else */
28
29     student = student + 1; /* increment student counter */
30 } /* end while */
31
32 /* termination phase; display number of passes and failures */
33 printf( "Passed %d\n", passes );
34 printf( "Failed %d\n", failures );
35
36 /* if more than eight students passed, print "raise tuition" */
37 if ( passes > 8 ) {
38     printf( "Raise tuition\n" );
39 } /* end if */
40
41 return 0; /* indicate program ended successfully */
42
43 } /* end function main */
```


**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

<i>Assume: int c = 3, d = 5, e = 4, f = 6, g = 12;</i>			
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		





```
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 } /* end function main */
```



Outline



Program Output

5
5
6

5
6
6

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.

