3

Structured Program Development in C

© 2007 Pearson Education, Inc. All rights reserved.

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**
- **Check the extra slides for more details …**

© 2007 Pearson Education, Inc. All rights reserved.

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**

© 2007 Pearson Education, Inc. All rights reserved.

Software Engineering Observation 3.6

Experience has shown that the most difficult part of solving a problem on a computer is developing the algorithm for the solution. Once a correct algorithm has been specified, the process of producing a working C program is normally straightforward.

© 2007 Pearson Education, Inc. All rights reserved.

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`**

© 2007 Pearson Education, Inc. All rights reserved.

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**

© 2007 Pearson Education, Inc. All rights reserved.

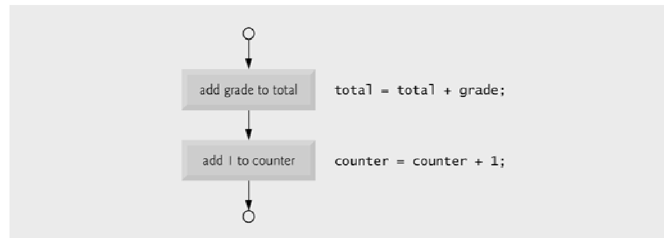


Fig. 3.1 | Flowcharting C's sequence structure.

© 2007 Pearson Education, Inc. All rights reserved.

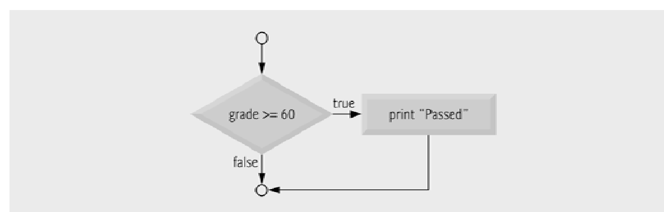


Fig. 3.2 | Flowcharting the single-selection `if` statement.

© 2007 Pearson Education, Inc. All rights reserved.

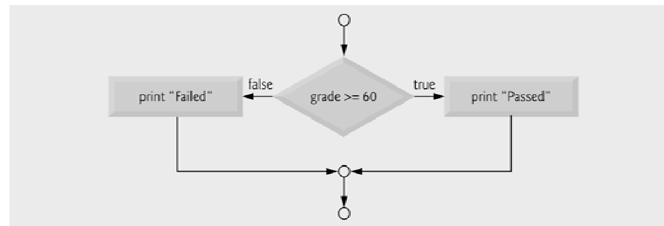


Fig. 3.3 | Flowcharting the double-selection `if...else` statement.

© 2007 Pearson Education, Inc. All rights reserved.

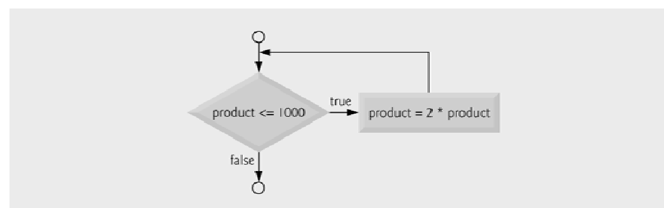


Fig. 3.4 | Flowcharting the `while` repetition statement.

© 2007 Pearson Education, Inc. All rights reserved.

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)**

© 2007 Pearson Education, Inc. All rights reserved.

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**

© 2007 Pearson Education, Inc. All rights reserved.

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)

© 2007 Pearson Education, Inc. All rights reserved.

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"

- **Common Programming Error 3.7:**
  - **An attempt to divide by zero causes a fatal error**
  - **→ explicitly test for this case and handle it appropriately in your program (such as printing an error message) rather than allowing the fatal error to occur**

© 2007 Pearson Education, Inc. All rights reserved.

```
1  Initialize total to zero
2  Initialize counter to zero
3
4  Input the first grade
5  While the user has not as yet entered the sentinel
6    Add this grade into the running total
7    Add one to the grade counter
8    Input the next grade (possibly the sentinel)
9
10 If the counter is not equal to zero
11   Set the average to the total divided by the counter
12   Print the average
13 else
14   Print "No grades were entered"
```

Fig. 3.7 | Pseudocode algorithm that uses sentinel-controlled repetition to solve the class average problem.

© 2007 Pearson Education, Inc. All rights reserved.

Outline

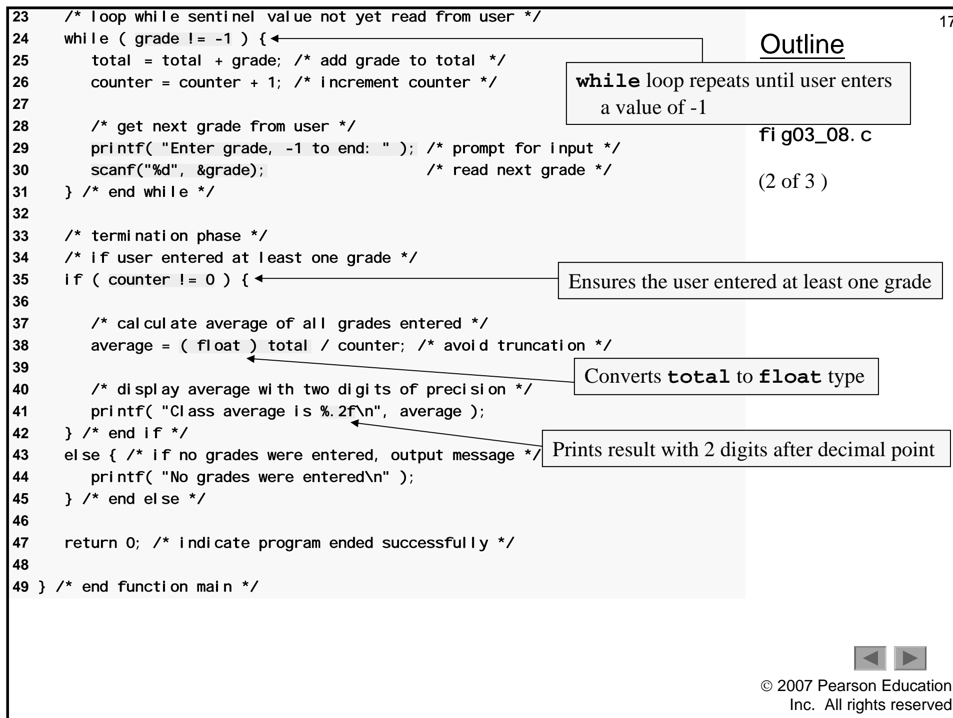
fig03_08.c

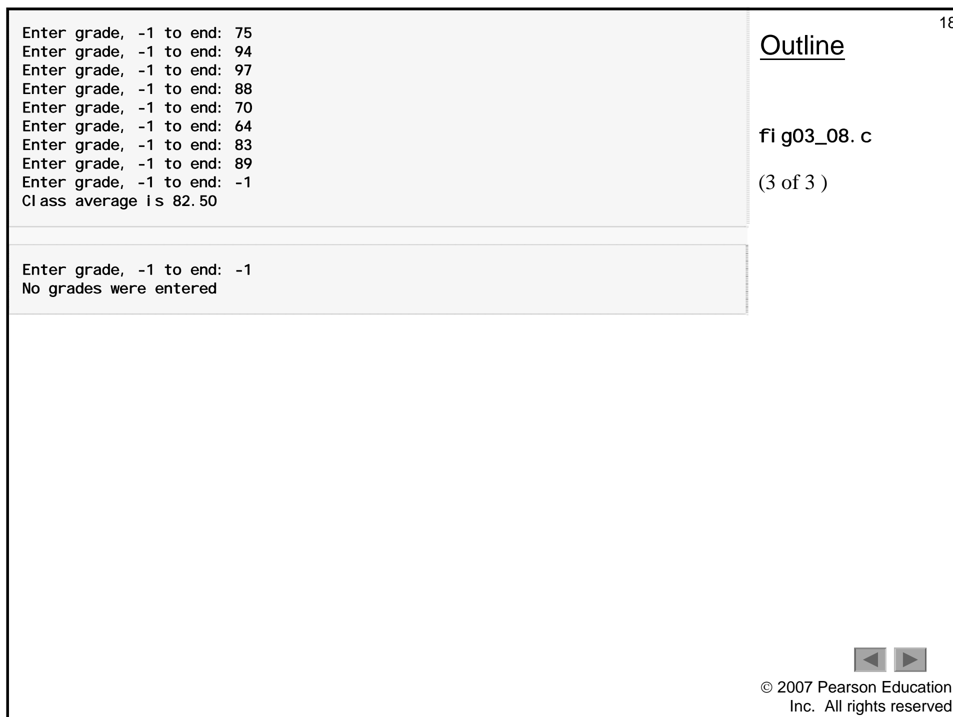
(1 of 3)

```
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( void )
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
```

float type indicates variable can be a non-integer

© 2007 Pearson Education, Inc. All rights reserved.


17
Outline
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 /* get next grade from user */
29 printf("Enter grade, -1 to end: "); /* prompt for input */
30 scanf("%d", &grade); /* read next grade */
31 } /* end while */
32
33 /* termination phase */
34 /* if user entered at least one grade */
35 if (counter != 0) {
36
37 /* calculate average of all grades entered */
38 average = (float) total / counter; /* avoid truncation */
39
40 /* display average with two digits of precision */
41 printf("Class average is %.2f\n", average);
42 } /* end if */
43 else { /* if no grades were entered, output message */
44 printf("No grades were entered\n");
45 } /* end else */
46
47 return 0; /* indicate program ended successfully */
48
49 } /* end function main */
while loop repeats until user enters a value of -1
fig03_08.c
(2 of 3)
Ensures the user entered at least one grade
Converts total to float type
Prints result with 2 digits after decimal point
© 2007 Pearson Education, Inc. All rights reserved.


18
Outline
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
fig03_08.c
(3 of 3)
© 2007 Pearson Education, Inc. All rights reserved.

Good Programming Practice 3.6

In a sentinel-controlled loop, the prompts requesting data entry should explicitly remind the user what the sentinel value is.

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

© 2007 Pearson Education, Inc. All rights reserved.

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**

© 2007 Pearson Education, Inc. All rights reserved.

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

© 2007 Pearson Education, Inc. All rights reserved.

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"

© 2007 Pearson Education, Inc. All rights reserved.

```
1  Initialize passes to zero
2  Initialize failures to zero
3  Initialize student to one
4
5  While student counter is less than or equal to ten
6      Input the next exam result
7
8      If the student passed
9        Add one to passes
10     else
11       Add one to failures
12
13     Add one to student counter
14
15 Print the number of passes
16 Print the number of failures
17 If more than eight students passed
18     Print "Raise tuition"
```

Fig. 3.9 | Pseudocode for examination results problem.

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig03_10.c

(1 of 3)

```
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( void )
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       else { /* otherwise, increment failures */
26          failures = failures + 1;
27       } /* end else */
28
29       student = student + 1; /* increment student counter */
30    } /* end while */
```

while loop continues until 10 students have been processed

if and **else** statements are nested inside **while** loop

© 2007 Pearson Education, Inc. All rights reserved.

```
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 */
```

```
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
```

(continued on next slide…)

Outline

fig03_10.c

(2 of 3)

© 2007 Pearson Education, Inc. All rights reserved.

(continued from previous slide…)

```
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
```

Outline

fig03_10.c

(3 of 3)

© 2007 Pearson Education, Inc. All rights reserved.

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)
```

© 2007 Pearson Education, Inc. All rights reserved.

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`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.

© 2007 Pearson Education, Inc. All rights reserved.

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**

© 2007 Pearson Education, Inc. All rights reserved.

3.12 Increment and Decrement Operators

- **If** `c` **equals** 5, **then**

```
printf( "%d", ++c );
```

  - **Prints** 6

```
printf( "%d", c++ );
```

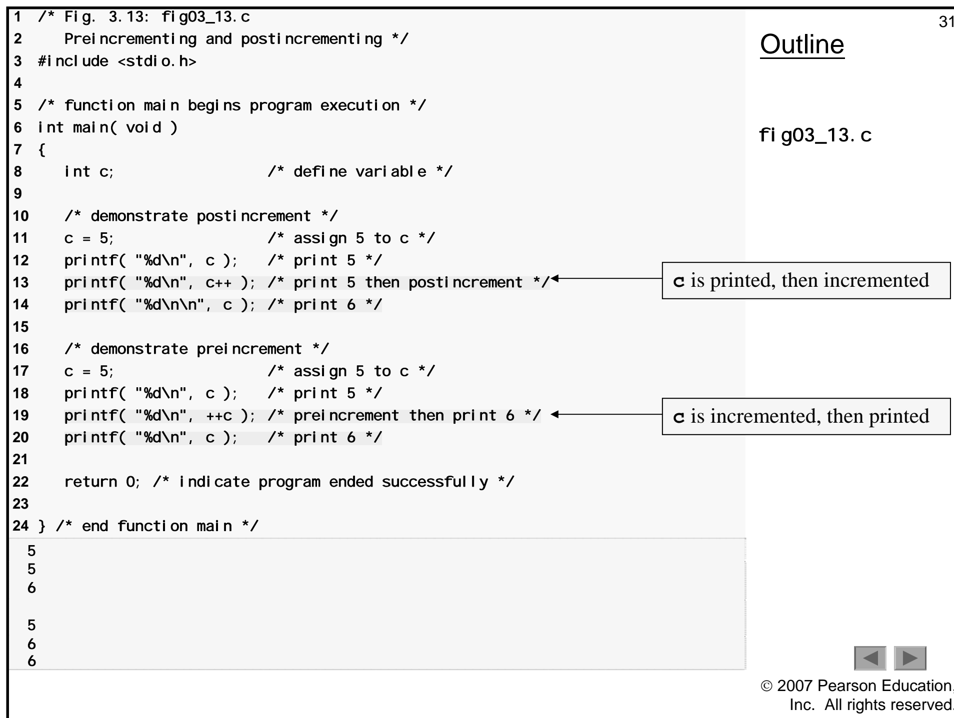
  - **Prints** 5
  - **In either case,** `c` **now has the value of** 6
- **When variable not in an expression**
  - **Preincrementing and postincrementing have the same effect**

```
++c;
printf( “%d”, c );
```

  - **Has the same effect as**

```
c++;
printf( “%d”, c );
```

© 2007 Pearson Education, Inc. All rights reserved.



Outline

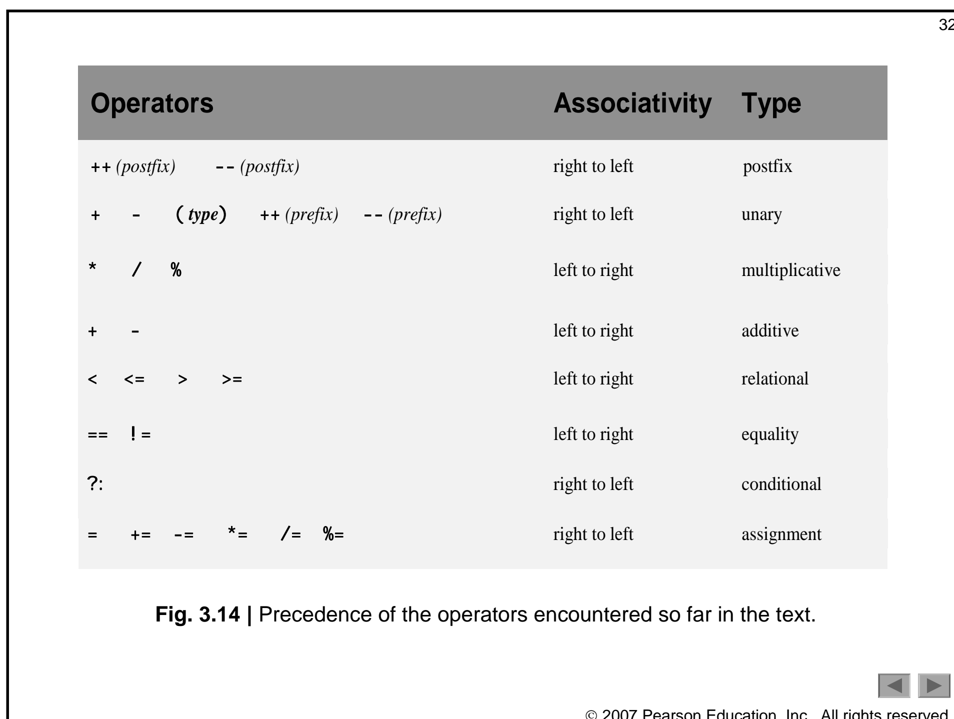
fig03_13.c

```
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( void )
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 */
```

```
5
5
6

5
6
6
```


© 2007 Pearson Education, Inc. All rights reserved.



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

© 2007 Pearson Education, Inc. All rights reserved.

4

C Program Control

© 2007 Pearson Education, Inc. All rights reserved.

4.12 Structured Programming Summary

- Structured programming
  - Easier than unstructured programs to understand, test, debug and, modify programs

Rules for Forming Structured Programs
1) Begin with the "simplest flowchart" (Fig. 4.19).
2) Any rectangle (action) can be replaced by two rectangles (actions) in sequence.
3) Any rectangle (action) can be replaced by any control statement (sequence, `if`, `if...else`, `switch`, `while`, `do...while` or `for`).
4) Rules 2 and 3 may be applied as often as you like and in any order.

Fig. 4.18 | Rules for forming structured programs.

© 2007 Pearson Education, Inc. All rights reserved.

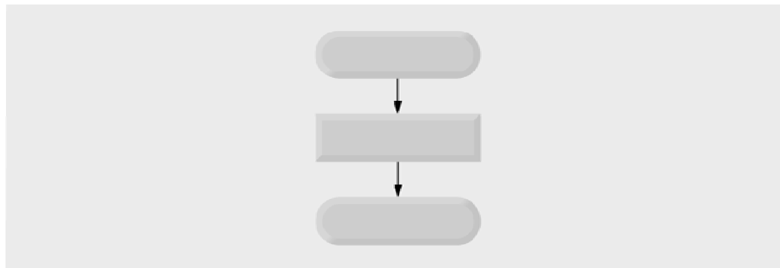


Fig. 4.19 | Simplest flowchart.

© 2007 Pearson Education, Inc. All rights reserved.

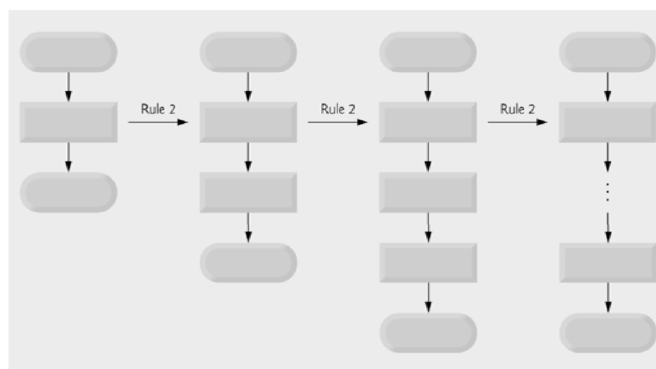


Fig. 4.20 | Repeatedly applying rule 2 of Fig. 4.18 to the simplest flowchart.

© 2007 Pearson Education, Inc. All rights reserved.

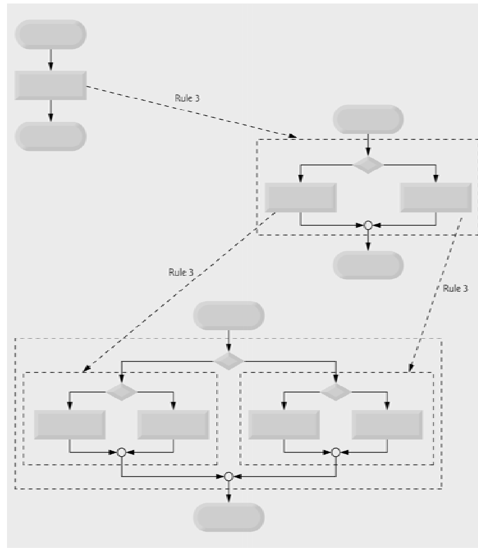


Fig. 4.21 | Applying rule 3 of Fig. 4.18 to the simplest flowchart.

© 2007 Pearson Education, Inc. All rights reserved.

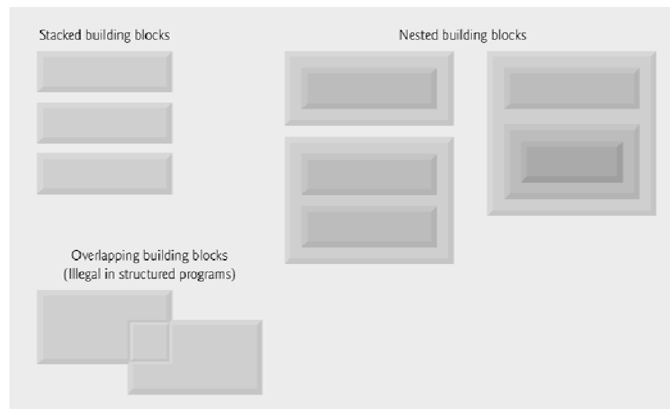


Fig. 4.22 | Stacked, nested and overlapped building blocks.

© 2007 Pearson Education, Inc. All rights reserved.

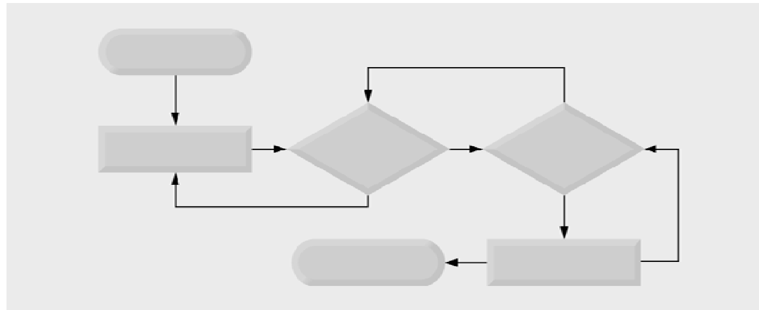


Fig. 4.23 | An unstructured flowchart.

© 2007 Pearson Education, Inc. All rights reserved.

4.12 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**

© 2007 Pearson Education, Inc. All rights reserved.