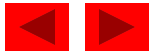
Chapter 3 - Structured Program Development

Outline

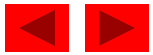


Objectives

- 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

- Steps to write a program (Review):
  - Define the problem to be solved with the computer
  - Design the program's input/output (what the user should give/see)
  - Break the problem into logical steps to achieve this output
  - Write the program (with an editor)
  - Compile the program
  - Test the program to make sure it performs as you expected
- 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

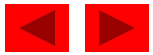


3.2 Algorithms (演算法)

- Computing problems
  - All can be solved by executing a series of actions in a specific order
- Algorithm: procedure in terms of
  1. *Actions* to be executed
  2. The *order* in which these actions are to be executed
  - Example: "rise-and-shine algorithm"
    - Get out of bed
    - Take off pajamas
    - Take a shower
    - Get dressed
    - Eat breakfast
    - Carpool to work

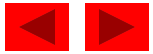
But if

- Program control
  - Specify order in which statements are to be executed



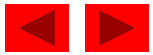
3.3 Pseudocode

- Pseudocode (虛擬碼)
  - 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



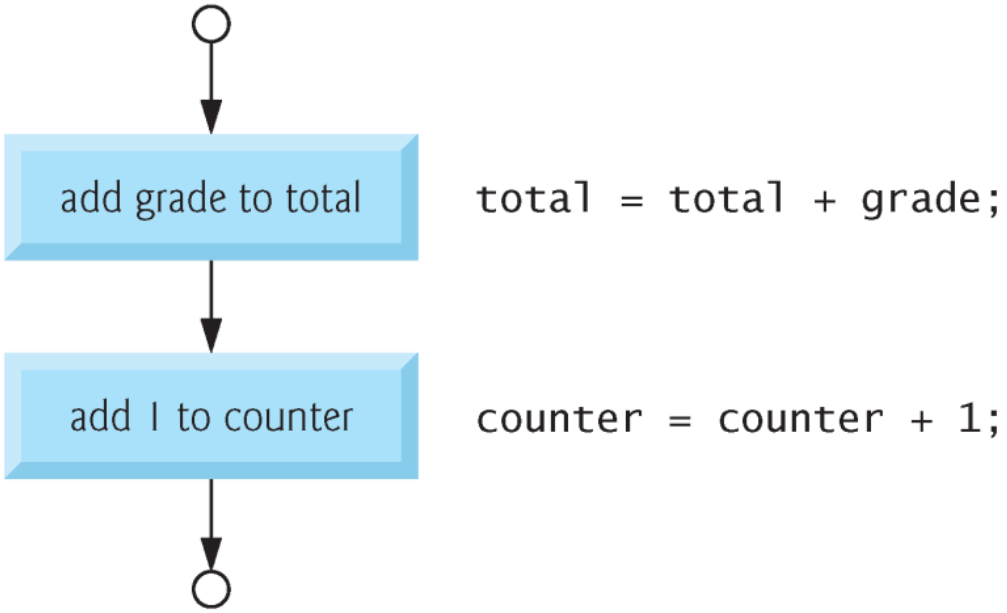
3.4 Control Structures

- Sequential execution (依序執行)
  - 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 showed that
  - All programs can be written in terms of *3 control structures*
    - Sequence structures: Built into C. Programs executed sequentially by default
    - Selection structures (選擇): C has three types: `if`, `if…else`, and `switch`
    - Repetition structures (迴圈): C has three types: `while`, `do…while` and `for`



3.4 Control Structures

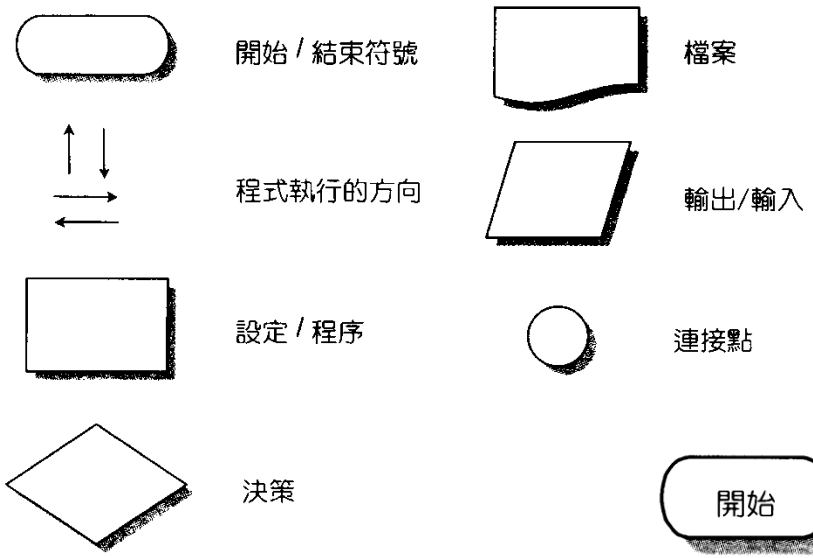
Figure 3.1 Flowcharting (流 程 圖) C's sequence structure.

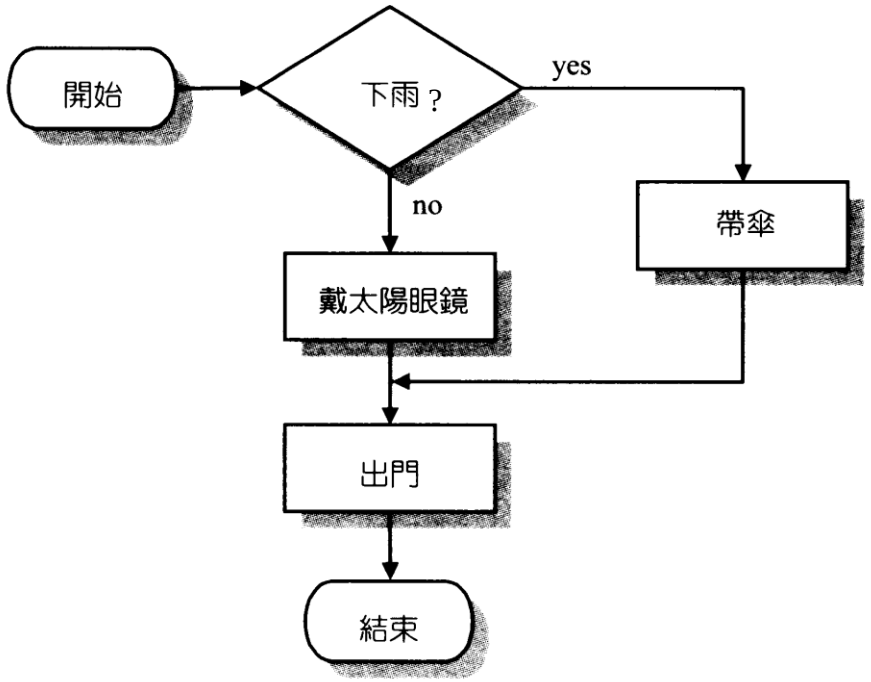


3.4 Control Structures

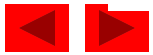
- Flowchart (流程圖)
  - *Graphical representation* of an algorithm
  - Drawn using certain *special-purpose symbols* connected by *arrows* called flowlines
  - *Rectangle* symbol (action symbol):
    - Indicates any type of action
  - *Oval* symbol:
    - Indicates the beginning or end of a program or a section of code
  - *Small circle* symbol (connector symbol):
    - Beginning or end of a small portion of an algorithm
  - *Diamond* symbol (decision symbol)
    - Indicates a decision is to be made (will be discussed next section)
- 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

Flowchart Symbols and Examples





From 洪維恩著 “C語言教學手冊”

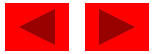


3.5 The `if` Selection Statement

- 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

- C Code:

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

or

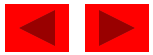
```
if ( grade >= 60 )
   {  printf( "Passed\n" );  }
```

- Psuedocode:

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

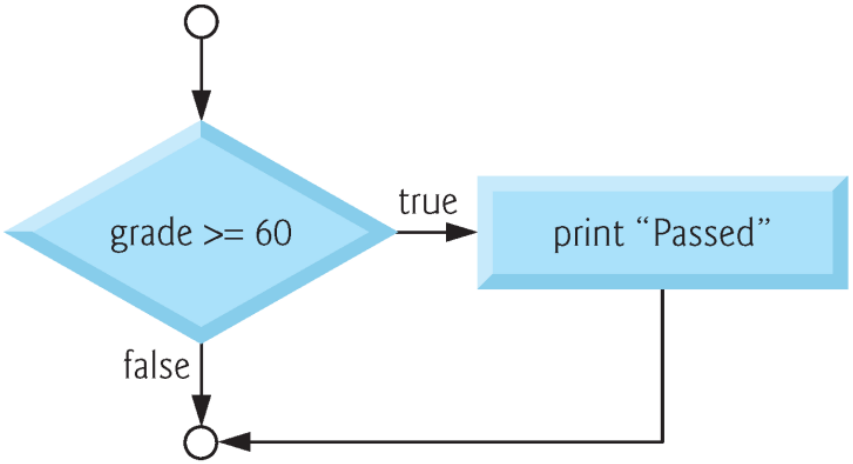
  – C code corresponds closely to the pseudocode

- 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 when the condition is `true`
  - another action 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:

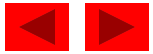
```
if ( grade >= 60 )
   printf( "Passed\n");
else
   printf( "Failed\n");
```

- Ternary conditional operator (`?` `:`)
  - Takes three arguments

    condition ? value if `true` : value if `false`
  - Our pseudocode could be written:

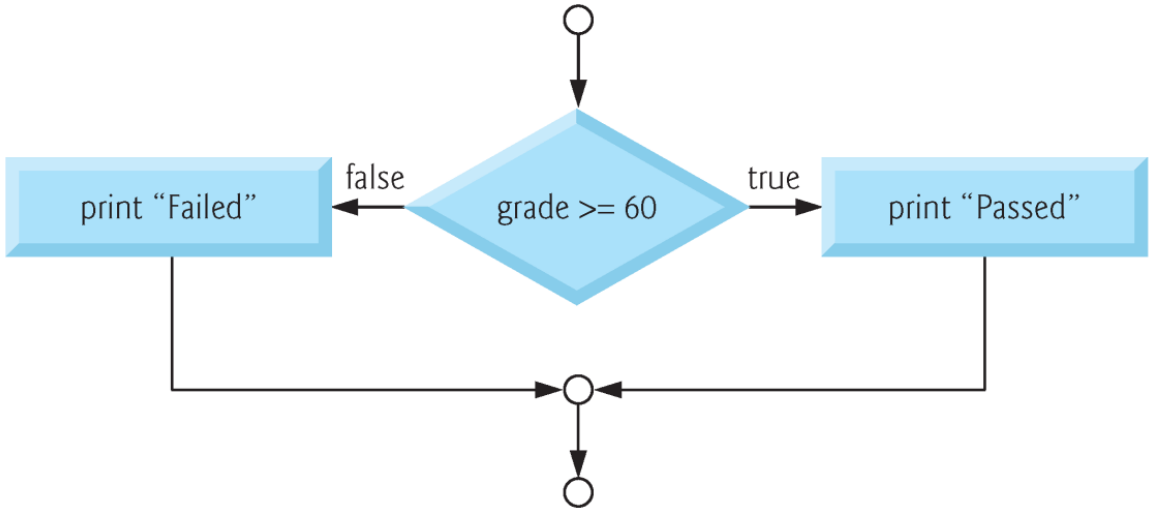
```
printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );
```
  - Or it could have been written:

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

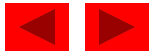



3.6 The `if...else` Selection Statement

- 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



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" );
   }
```

  - What is the difference between the above statement and

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else
   printf( "Failed.\n" );
   printf( "You must take this course again.\n" );
```

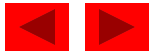
  - Answer: Same as

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else
   printf( "Failed.\n" );
printf( "You must take this course again.\n" );
```

    That is,

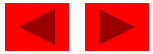
```
printf( "You must take this course again.\n" );
```

    would always be executed for the second case.



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



3.6 if...else 配 對 問 題

Determine the output

(1) when x = 9 and y = 11 and

```
if ( x < 10 )
   if ( y > 10 )
      printf( "*****\n" );
   else
      printf( "#####\n" );
printf( "$$$$$\n" );
```

Ans: x = 9, y =11

*****

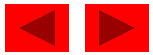
$$$$$

(2) when x = 11 and y = 9

```
if ( x < 10 )
   if ( y > 10 )
      printf( "*****\n" );
   else
      printf( "#####\n" );
printf( "$$$$$\n" );
```

Ans: x = 11, y =9

$$$$$



3.6 if...else 配 對 問 題

(1) when x = 9 and y = 11 and

```
if ( x < 10 ) {
if ( y > 10 )
printf( "*****\n" );
}
else {
printf( "#####\n" );
printf( "$$$$$\n" );
}
```

Ans for x = 9, y =11

*****

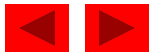
(2) when x = 11 and y = 9

```
if ( x < 10 ) {
   if ( y > 10 )
      printf( "*****\n" );
   }
else {
   printf( "#####\n" );
   printf( "$$$$$\n" );
   }
```

Ans for x = 11, y =9

#####

$$$$$



3.6 if...else 配對問題

(1) when x = 9 and y = 11

```
if ( x < 10 )
if ( y > 10 ) {
printf( "*****\n" );
}
else {
printf( "#####\n" );
printf( "$$$$$\n" );
}
```

*****

(2) when x = 11 and y = 9

```
if ( x < 10 )
   if ( y > 10 ) {
      printf( "*****\n" );
   }
   else {
      printf( "#####\n" );
      printf( "$$$$$\n" );
   }
```

Nothing!

3.6 if...else 配對問題

3.32 Modify the following code to produce the output shown.

```
if ( y == 8 )
if ( x == 5 )
printf( "@@@@@\n" );
else
printf( "#####\n" );
printf( "$$$$$\n" );
printf( "&&&&&\n" );
```

Assuming x = 5 and y = 8,

the following output is produced.

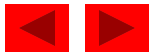
```
@@@@@
$$$$$
&&&&&
```

```
if ( y == 8 )
   if ( x == 5 )
      printf( "@@@@@\n" );
   else {
      printf( "#####\n" );
      printf( "$$$$$\n" );
   }

printf( "&&&&&\n" );
```

Assuming x=5 and y=8, then

@@@@@

&&&&&



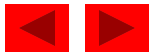
3.6 if...else 配對問題

What is the output for the following code?

```
int course, code;
course = 1;
code = 2;
if ( course == 1 )
   if ( code < 2 )
      printf( "Chemical Engineering\n" );
else
      printf( "No course listed\n");
printf( "*** End of course listings *** \n" );
```

Which one is the correct output?

```
   No course listed
   *** End of course listings ***
or
   *** End of course listings ***
```

3.6 The `if...else` Selection Statement

- C code:

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

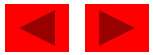
- Ternary conditional operator (? :)
  - Takes three arguments

    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:

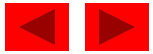
```
grade >= 60 ? printf( “Passed\n” ) : printf( “Failed\n” );
```

3.6 The `if...else` Selection Statement

– 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

```
if ( grad >= 90 )
   printf( "A\n" );
else
   if ( grade >= 80)
      printf( "B\n" );
   else
      if ( grade >= 70 )
         printf( "C\n" );
      else
         if ( grade >= 60 )
            printf( "D\n" );
         else
            printf( "F\n" );
```

```
if ( grad >= 90 )
   printf( "A\n" );
else if ( grade >= 80)
   printf( "B\n" );
else if ( grade >= 70 )
   printf( "C\n" );
else if ( grade >= 60 )
   printf( "D\n" );
else
   printf( "F\n" );
```


3.7 The `while` Repetition Statement

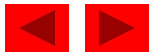
- Repetition structure
  - Repetition structures (迴圈):
    (1) `while`, (2) **`do…while`** and (3) **`for`**
  - Programmer specifies **an action to be repeated** *while some condition remains* `true`
  - 在此先介紹 **`while`**迴圈
  - Psuedocode

    While there are more items on my shopping list

    Purchase next item and cross it off my list

    Condition

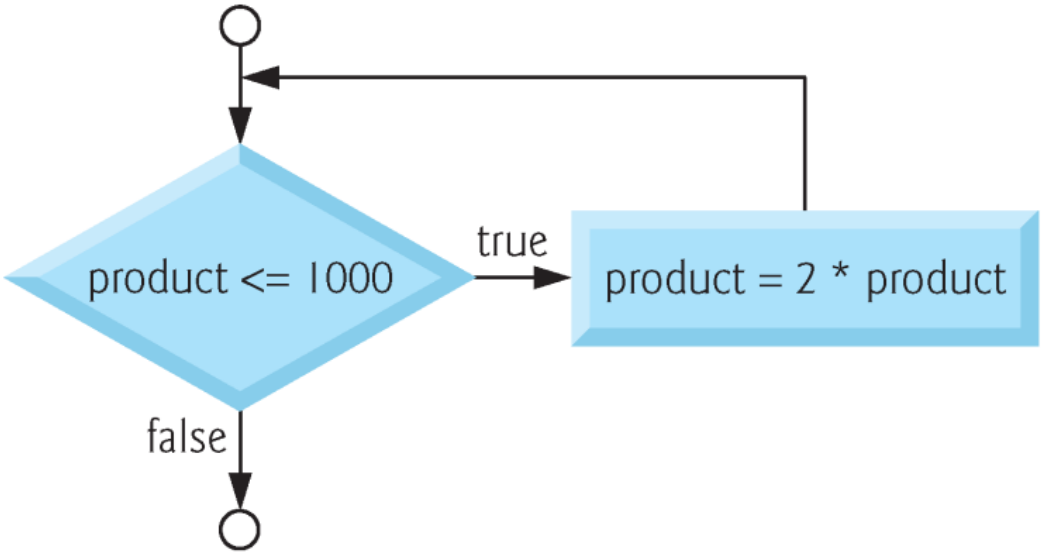
    Actions
  - `while` loop repeated until condition becomes `false`



3.7 The `while` Repetition Statement

- Example:

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



The final value of product will be 1024.

3.8 Formulating Algorithms for 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 for counter-controlled loops:

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

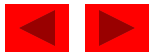
Initialization Phase

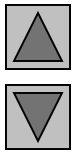
```
While grade counter is less than or equal to ten
    Input grade
    Add the grade into the total
    Add one to the grade counter
```

Processing Phase

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

Termination Phase

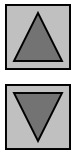




Outline

fig03_06.c (Part 1 of 2)

```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 */

```



Outline

```
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 */
```

fig03_06.c (Part 2 of 2)

```
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 for Sentinel-Controlled Repetition with Top-Down, Stepwise Refinement

- Problem in Sec 3.8 becomes:

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

  – Unknown number of students

  – Question: How will the program know to end??????

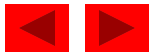
- Use sentinel value (步哨值、訊號)

  – Also called signal value, dummy value, or flag value (旗標值)

  – 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)

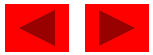


3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

- Top-down, stepwise refinement
  - Begin with a pseudocode representation of the *top* (a single statement that conveys the program's overall function):

    Determine the class average for the quiz
  - Divide *top* into smaller tasks (refinement) 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:

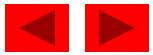
  Initialize total to zero
  Initialize counter to zero

- 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)

- 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"

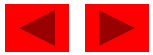


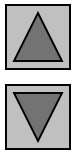
3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

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"

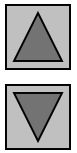




Outline

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
```

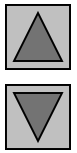


Outline

fig03_08.c (Part 2 of 2)

```
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       /* Get the next grade from user */
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 */
```

因為分數不會是負分，因此用 -1 值當成訊號，也就是當分數為 -1 時，輸入分數的迴圈就停止，程式進入下一個階段。



Outline

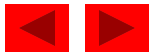
Program Output

```
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 (a known value) test results
    - Therefore, *counter-controlled* loop will be used
  - *Two additional 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

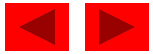
  Print a summary of the exam results and decide if tuition should be raised

- Refine *Initialize variables* to

  Initialize passes to zero

  Initialize failures to zero

  Initialize student counter to one



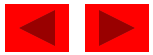
3.10 *Nested* Control Structures

- 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

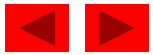
While student counter is less than or equal to ten
Input the next exam result

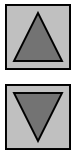
If the student passed
Add one to passes
else
Add one to failures

Add one to student counter

Nested!

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

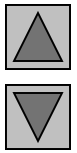




Outline

fig03_10.c (Part 1 of 2)

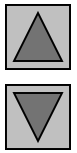
```
1  /* Fig. 3.10: fig03_10.c
2     Analysis of examination results */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     /* initialize variables in definitions */
9     int passes = 0;   /* number of passes */
10    int failures = 0; /* number of failures */
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       else { /* otherwise, increment failures */
26          failures = failures + 1;
27       } /* end else */
28
29       student = student + 1; /* increment student counter */
30    } /* end while */
```



Outline

fig03_10.c (Part 2 of 2)

```
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 */
```



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

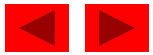
- *Assignment operators* **abbreviate** *assignment expressions*,
- e.g., assignment expression `c = c + 3;` can be abbreviated with the addition assignment operator `c += 3;`
- Statements of the form

  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)
```

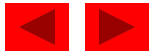



3.11 Assignment Operators

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

3.12 Increment and Decrement Operators

- Increment operator (++)
  - Can be used instead of `c = c + 1` or `c += 1`
- Decrement operator (--)
  - Can be used instead of `c = c - 1` or `c -= 1`
- Pre-increment or pre-decrement
  - Operator is used before the variable (`++c` or `--c`)
  - ++c or –-c may appear in an expression for additional calculation
  - Variable is changed, then the expression it is in is evaluated
- Post-increment or post-decrement
  - Operator is used after the variable (`c++` or `c--`)
  - c++ or c– may appear in an expression for additional calculation
  - Expression executes, then the variable is changed



3.12 Increment and Decrement Operators

- If `c = 5`, then

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

  - Prints 6

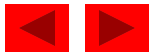
```
printf( "%d", c++ );
```

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

```
++c;
printf( “%d”, c );
```

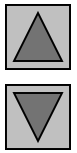
  - Has the same effect as

```
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.



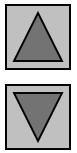
Outline

fig03_13.c

```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

```
passes = passes + 1;
failures = failures + 1;
student = student + 1;
```

```
++passes;
++failures;
++student;
```

```
passes += 1;
failures += 1;
student += 1;
```

```
passes++;
failures++;
student++;
```

Exercise

請在右方寫出螢幕上顯示的結果：

```
#include <stdio.h>
int main()
{
   int c = 5 ;
   printf( "c   =  %d\n",   c );     c   =  5
   printf( "c++ =  %d\n", c++ );     c++ =  5
   printf( "--c =  %d\n", --c );     --c =  5
   printf( "--c =  %d\n", --c );     --c =  4
   printf( "c++ =  %d\n", c++ );     c++ =  4
   printf( "  c =  %d\n",   c );       c =  5
   printf( "++c =  %d\n", ++c );     ++c =  6
   printf( "--c =  %d\n", --c );     --c =  5
   printf( "c-- =  %d\n", c-- );     c-- =  5
   printf( "  c =  %d\n",   c );       c =  4
   return 0;
}
```


3.12 Increment and Decrement Operators

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

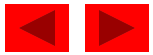
Exercise

```
#include <stdio.h>
int main()
{
   int a, c, d;

   a = 9 ; c = 5 ;
   d = a---c;

   printf( "a = %2d, c = %2d, d = %2d\n",   a, c, d );

   a = 9 ; c = 5 ;
   d = a-- - --c;

   printf( "a = %2d, c = %2d, d = %2d\n",   a, c, d );
   return 0;
}
```

```
a =  8, c =  5, d =  4
a =  8, c =  4, d =  5
```


Review

- In this chapter, we have learned:
  - 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 `? :` , i.e., condition ? value if `true` : value if `false`
  - 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.



Exercise 3.11

Identify and correct the errors in each of the following [*Note*: There may be more than one error in each piece of code]:

```
if ( age >= 65 );
   printf( "Age is greater than or equal to 65\n" );
else
printf( "Age is less than 65\n" );
```

ANS:

```
if ( age >= 65 ) /* ; removed */
   printf( "Age is greater than or equal to 65\n" );
else
printf( "Age is less than 65\n" );
```


Exercise 3.11

Identify and correct the errors in each of the following

[*Note*: There may be more than one error in each piece of code]:

```
int x = 1, total;
while ( x <= 10 ) {
   total += x;
   ++x;
}
ANS:
int x = 1, total = 0;
while ( x <= 10 ) {
   total += x;
   ++x;
}
```

```
While ( x <= 100 )
   total += x;
   ++x;

ANS:
while ( x <= 100 ) {
   total += x;
   ++x;
}
```

```
y = 5;
while ( y > 0 ) {
   printf( "%d\n", y );
   ++y;
}

ANS:
y = 5;
while ( y > 0 ) {
   printf( "%d\n", y );
   --y;
}
```