Chapter 4 – C Program Control

Outline

Objectives

- In this chapter, you will learn:
  - To be able to use the **`for`** and **`do…while`** repetition statements.
  - To understand multiple selection using the **`switch`** selection statement.
  - To be able to use the **`break`** and **`continue`** program control statements
  - To be able to use the logical operators (**`&&`** , **`||`** , **`!`**)

4.1 Introduction

- We have learned
  - Selection structures (選擇): **`if`**, **`if . . . else`**
  - Repetition structures (迴圈): **`while`**
- This chapter introduces
  - Additional repetition control structures
    - **`for`**
    - **`do…while`**
  - **`switch`** multiple selection statement
  - **`break`** statement
    - Used for exiting immediately and rapidly from certain control structures
  - **`continue`** statement
    - Used for skipping the remainder of the body of a repetition structure and proceeding with the next iteration of the loop
  - logical operators (**`&&`** , **`||`** , **`!`**)

4.2 The Essentials of Repetition

- Loop
  - Group of instructions computer executes repeatedly while some condition remains **`true`**
- Counter-controlled repetition
  - Definite repetition: know how many times loop will execute
  - Control variable used to count repetitions
- Sentinel-controlled repetition
  - Indefinite repetition
  - Used when number of repetitions not known
  - Sentinel value indicates "end of data"

4.3 Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires
  - The name of a control variable (or loop counter)
  - The initial value of the control variable
  - An increment (or decrement) by which the control variable is modified each time through the loop
  - A condition that tests for the final value of the control variable (i.e., whether looping should continue)

4.3 Essentials of Counter-Controlled Repetition

- Example:

```
int counter = 1;           // initialization
while ( counter <= 10 ) { // repetition condition
   printf( "%d\n", counter );
   ++counter;              // increment
}
```

  - The statement

```
int counter = 1;
```

    - Names `counter`
    - Defines it to be an integer
    - Reserves space for it in memory
    - Sets it to an initial value of `1`

Counter-Control Repetition

Outline

```
1  /* Fig. 4.1: fig04_01.c
2     Counter-controlled repetition */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter = 1;              /* initialization */
9
10    while ( counter <= 10 ) {     /* repetition condition */
11       printf ( "%d\n", counter ); /* display counter */
12       ++counter;                  /* increment */
13    } /* end while */
14
15    return 0; /* indicate program ended successfully */
16
17 } /* end function main */
```

fig04_01.c

```
1
2
3
4
5
6
7
8
9
10
```

Program Output

4.3 Essentials of Counter-Controlled Repetition

- Condensed code
  - C Programmers would make the program more concise
  - Initialize `counter` to `0`

```
counter = 0;

while ( ++counter <= 10 )
   printf( "%d\n, counter );
```

此處 counter 先加 1，再判斷是否 <= 10

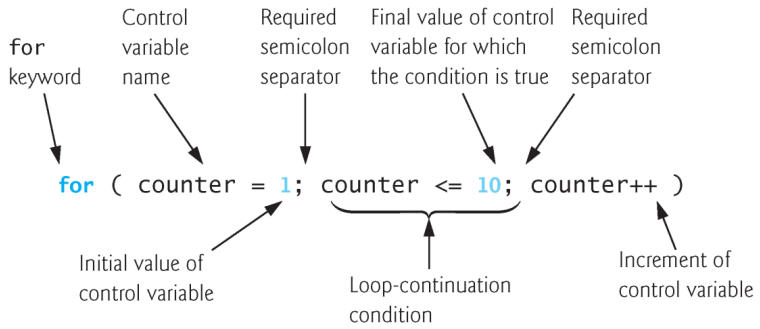
請問迴圈會做幾次？

如果改成

```
while ( counter++ <=  10 )
printf( "%d\n", counter );
```

會做幾次？

4.4 The for Repetition Statement



Outline

4.4 The for Repetition Statement

Rewrite the above program with **for** repetition statement

```
1  /* Fig. 4.2: fig04_02.c
2     Counter-controlled repetition with the for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter; /* define counter */
9
10    /* initialization, repetition condition, and increment
11       are all included in the for statement header. */
12    for ( counter = 1; counter <= 10; counter++ ) {
13       printf( "%d\n", counter );
14    } /* end for */
15
16    return 0; /* indicate program ended successfully */
17
18 } /* end function main */
```

```
8     int counter = 1;              /* initialization */
9
10    while ( counter <= 10 ) {     /* repetition condition */
11       printf ( "%d\n", counter ); /* display counter */
12       ++counter;                 /* increment */
13    } /* end while */
```


4.4 The for Repetition Statement

- Format when using `for` loops

  for (*initialization*; *loopContinuationTest*; *increment*)
  statement

- **for** loops can usually be rewritten as **while** loops:

  initialization;
  while (*loopContinuationTest*) **{**
  statement;
  increment;
  }

- Example:

```
for( counter = 1; counter <= 10; counter++ )
   printf( "%d\n", counter );
```

  – Prints the integers from one to ten

No semicolon (;) after last expression

4.4 The for Repetition Statement

- Initialization and increment
  - Can be comma-separated lists
  - Example:

```
for ( i = 0, j = 0;  j + i <= 10; j++, i++ )
   printf( "%d\n", j + i );
```

- Increment Expression（以下四種都可用）

```
counter = counter + 1
counter += 1
++counter
counter++          /* preferred */
```

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions. If x equals 2 and y equals 10

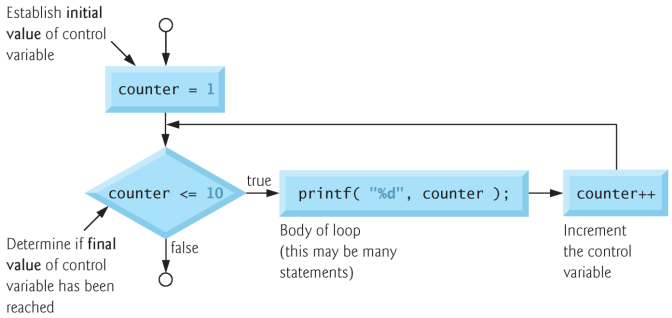
```
for ( j = x; j <= 4 * x * y; j += y / x )
```

  is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```

4.5 The for Statement : Notes and Observations

- Notes about the for statement:
  - "Increment" may be negative (decrement)
  - If the loop continuation condition is initially false
    - The body of the for statement is not performed
    - Control proceeds with the next statement after the for statement
  - Control variable
    - Often printed or used inside for body, but not necessary



Using for to Sum Numbers

Outline

fig04_05.c

```
1  /* Fig. 4.5: fig04_05.c
2     Summation with for */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int sum = 0; /* initialize sum */
9     int number;  /* number to be added to sum */
10
11    for ( number = 2; number <= 100; number += 2 ) {
12       sum += number; /* add number to sum */
13    } /* end for */
14
15    printf( "Sum is %d\n", sum ); /* output sum */
16
17    return 0; /* indicate program ended successfully */
18
19 } /* end function main */
```

```
Sum is 2550
```

2 + 4 + 6 + 8 + . . . 100 = 2550

4.6 Compound Interest (複利)

A person invests $1000.00 in a savings account yielding 5% interest. Assuming that all interest is left on deposit in the account, calculate and print the amount of money in the account at the end of each year for 10 years. Use the following formula for determining these amounts:

$$a = p\,(1 + r)^n$$

where

p is the principal (本金)
r is the interest rate (利率)
n is the number of years
a is the 本利和 (deposit)

Calculating Compound Interest with for

Outline

fig04_06.c (Part 1 of 2)

```
1  /* Fig. 4.6: fig04_06.c
2     Calculating compound interest */
3  #include <stdio.h>
4  #include <math.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     double amount;           /* amount on deposit */
10    double principal = 1000.0; /* starting principal */
11    double rate = .05;       /* interest rate */
12    int year;                /* year counter */
13
14    /* output table column head */
15    printf( "%4s%21s\n", "Year", "Amount on deposit" );
16
17    /* calculate amount on deposit for each of ten years */
18    for ( year = 1; year <= 10; year++ ) {
19
20       /* calculate new amount for specified year */
21       amount = principal * pow( 1.0 + rate, year );
22
23       /* output one table row */
24       printf( "%4d%21.2f\n", year, amount );
25    } /* end for */
26
```

如果程式中用到數學函數時，必須引入 math.h 標頭檔

%d: integers
%f: floating-point values
%s: strings (字串)

%4s%21s and %4d%21.2f in the printf, same as printf("Year Amount on deposit\n");

pow(x,y) calculates x^y where x and y are double

Need #include <math.h>

%4d%21.2f in the printf

```
27    return 0; /* indicate program ended successfully */
28
29 } /* end function main */
```

```
Year    Amount on deposit
   1              1050.00
   2              1102.50
   3              1157.63
   4              1215.51
   5              1276.28
   6              1340.10
   7              1407.10
   8              1477.46
   9              1551.33
  10              1628.89
```

Outline

fig04_06.c (Part 2 of 2)

Program Output

九九乘法表 – Nested for Loops

```
1×1= 1  1×2= 2  1×3= 3  1×4= 4  1×5= 5  1×6= 6  1×7= 7  1×8= 8  1×9= 9
2×1= 2  2×2= 4  2×3= 6  2×4= 8  2×5=10  2×6=12  2×7=14  2×8=16  2×9=18
3×1= 3  3×2= 6  3×3= 9  3×4=12  3×5=15  3×6=18  3×7=21  3×8=24  3×9=27
4×1= 4  4×2= 8  4×3=12  4×4=16  4×5=20  4×6=24  4×7=28  4×8=32  4×9=36
5×1= 5  5×2=10  5×3=15  5×4=20  5×5=25  5×6=30  5×7=35  5×8=40  5×9=45
6×1= 6  6×2=12  6×3=18  6×4=24  6×5=30  6×6=36  6×7=42  6×8=48  6×9=54
7×1= 7  7×2=14  7×3=21  7×4=28  7×5=35  7×6=42  7×7=49  7×8=56  7×9=63
8×1= 8  8×2=16  8×3=24  8×4=32  8×5=40  8×6=48  8×7=56  8×8=64  8×9=72
9×1= 9  9×2=18  9×3=27  9×4=36  9×5=45  9×6=54  9×7=63  9×8=72  9×9=81
Press any key to continue
```

```
/* Example Table99.c, nested for loops 印出九九乘法表 */
#include <stdio.h>

int main()
{
   int i,j;

   for ( i=1 ; i<=9 ; i++ )                    /* 外層迴圈 */
   {
      for ( j=1 ; j<=9 ; j++ )                 /* 內層迴圈 */
         printf("%d*%d=%2d  ",i,j,i*j);

   printf("\n");
   }

   return 0;
}
```

迴圈在哪裡？

九九乘法表 – Nested while Loops

```
/* Example Table99, nested while loops 求9*9乘法表 */
#include <stdio.h>

int main()
{
   int i=1, j=1;                          /* 設定迴圈控制變數的初值 */

   while (i<=9)                           /* 外層迴圈 */
   {

      while (j<=9)                        /* 內層迴圈 */
      {
         printf("%d*%d=%2d  ",i,j,i*j);
         j++;
      }

      printf("\n");
      i++;
      j=1;
   }

   return 0;
}
```


九九乘法表 – Nested for Loops

Question: How to modify the source code to produce

```
1×1= 1
2×1= 2  2×2= 4
3×1= 3  3×2= 6  3×3= 9
4×1= 4  4×2= 8  4×3=12  4×4=16
5×1= 5  5×2=10  5×3=15  5×4=20  5×5=25
6×1= 6  6×2=12  6×3=18  6×4=24  6×5=30  6×6=36
7×1= 7  7×2=14  7×3=21  7×4=28  7×5=35  7×6=42  7×7=49
8×1= 8  8×2=16  8×3=24  8×4=32  8×5=40  8×6=48  8×7=56  8×8=64
9×1= 9  9×2=18  9×3=27  9×4=36  9×5=45  9×6=54  9×7=63  9×8=72  9×9=81
```

```
1×1= 1  1×2= 2  1×3= 3  1×4= 4  1×5= 5  1×6= 6  1×7= 7  1×8= 8  1×9= 9
2×1= 2  2×2= 4  2×3= 6  2×4= 8  2×5=10  2×6=12  2×7=14  2×8=16
3×1= 3  3×2= 6  3×3= 9  3×4=12  3×5=15  3×6=18  3×7=21
4×1= 4  4×2= 8  4×3=12  4×4=16  4×5=20  4×6=24
5×1= 5  5×2=10  5×3=15  5×4=20  5×5=25
6×1= 6  6×2=12  6×3=18  6×4=24
7×1= 7  7×2=14  7×3=21
8×1= 8  8×2=16
9×1= 9
```


4.7 The switch Multiple-Selection Statement

- switch
  - Useful when a variable or expression is tested for all the values it can assume and different actions are taken
- Format
  - Series of **case** labels and an optional **default** case

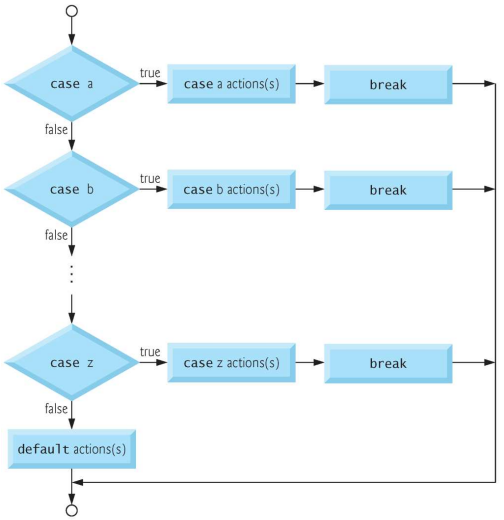
```
switch ( value ){
    case 1:
        actions;
        break;      //如果不加 break ，會繼續執行下面的 actions
    case 2:
        actions;
        break;
    default:
        actions;
        break;      //可加可不加
    }
```

如果變數名稱 value 是數值的話，此處 case 1 其中的 1 爲 value 的數值，但是如果 value 是字元的話，必須用 case 'A'，其中的 A 爲 value 的值。

  - **break**; exits from statement

4.7 The switch Multiple-Selection Statement

- Flowchart of the switch statement



Counting Letter Grades with switch

Outline

fig04_07.c (Part 1 of 3)

```
1  /* Fig. 4.7: fig04_07.c
2     Counting letter grades */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int grade;      /* one grade */
9     int aCount = 0; /* number of As */
10    int bCount = 0; /* number of Bs */
11    int cCount = 0; /* number of Cs */
12    int dCount = 0; /* number of Ds */
13    int fCount = 0; /* number of Fs */
14
15    printf(  "Enter the letter grades.\n"  );
16    printf(  "Enter the EOF character to end input.\n"  );
17
18    /* loop until user types end-of-file key sequence */
19    while ( ( grade = getchar() ) != EOF ) {
20
21       /* determine which grade was input */
22       switch ( grade ) { /* switch nested in while */
23
24          case 'A':   /* gra
25          case 'a':   /* or lowercase a */
26             ++aCount; /* increment aCount
27             break;    /* necessary to exit switch */
28
```

此處也可宣告 **grade** 爲字元，也就是用 **char grade;**

此處 aCount, bCount 等分別是出現 A，B，...等字元的次數。

The **getchar( )** function reads one character from the keyboard and stores that character in integer variable **grade**

EOF (end-of-file) is system dependent. In MS Windows, EOF is <crtl-z>

此處 value 是字元，所以用 case 'A'，其中的 A 爲 value 的值。

Outline

fig04_07.c (Part 2 of 3)

```
29          case 'B':   /* grade was uppercase B */
30          case 'b':   /* or lowercase b */
31             ++bCount; /* increment bCount */
32             break;    /* exit switch */
33
34          case 'C':   /* grade was uppercase C */
35          case 'c':   /* or lowercase c */
36             ++cCount; /* increment cCount */
37             break;    /* exit switch */
38
39          case 'D':   /* grade was uppercase D */
40          case 'd':   /* or lowercase d */
41             ++dCount; /* increment dCount */
42             break;    /* exit switch */
43
44          case 'F':   /* grade was uppercase F */
45          case 'f':   /* or lowercase f */
46             ++fCount; /* increment fCount */
47             break;    /* exit switch */
48
49          case '\n':  /* ignore newlines, */
50          case '\t':  /* tabs, */
51          case ' ':   /* and spaces in input */
52             break;    /* exit switch */
53
```

注意：每個 case 最後必須加上 break，跳出 switch 區塊，否則會繼續執行下一個 case 。

```
54          default:     /* catch all other characters */
55             printf( "Incorrect letter grade entered." );
56             printf( " Enter a new grade.\n" );
57             break;    /* optional; will exit switch anyway */
58       } /* end switch */
59
60    } /* end while */
61
62    /* output summary of results */
63    printf( "\nTotals for each letter grade are:\n" );
64    printf( "A: %d\n", aCount ); /* display number of A grades */
65    printf( "B: %d\n", bCount ); /* display number of B grades */
66    printf( "C: %d\n", cCount ); /* display number of C grades */
67    printf( "D: %d\n", dCount ); /* display number of D grades */
68    printf( "F: %d\n", fCount ); /* display number of F grades */
69
70    return 0; /* indicate program ended successfully */
71
72 } /* end function main */
```

Outline

fig04_07.c (Part 3 of 3)


```
Enter the letter grades.
Enter the EOF character to end input.
a
b
C
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

Outline

Program Output

4.8 The do…while Repetition Statement

- The do…while repetition statement
  - Similar to the while structure
  - Condition for repetition tested after the body of the loop is performed
    - All actions are performed at least once
  - Format:

```
do {
   statement;
} while ( condition );
```

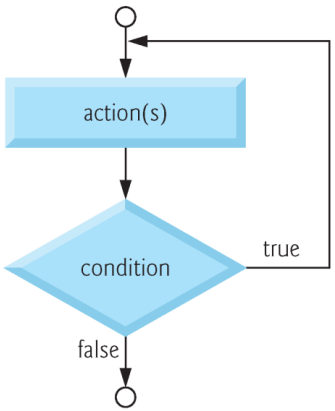
- Example (initially, counter = 1):

```
do {
   printf( "%d  ", counter );
} while (++counter <= 10);
```

  - Prints the integers from 1 to 10

4.8 The do…while Repetition Statement

- Flowchart of the do…while repetition statement



Example of `do ... while` Statement

Outline

```
1  /* Fig. 4.9: fig04_09.c
2     Using the do/while repetition statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter = 1; /* initialize counter */
9
10    do {
11       printf( "%d  ", counter ); /* display counter */
12    } while ( ++counter <= 10 ); /* end do...while */
13
14    return 0; /* indicate program ended successfully */
15
16 } /* end function main */
```

fig04_09.c

```
1  2  3  4  5  6  7  8  9  10
```

Program Output

Question: What if we set `while ( counter++ <= 10 );` ?

```
1 2 3 4 5 6 7 8 9 10 11
```


while, for, do . . . while 之比較

語法：

```
for ( initialization; loopContinuationTest; increment )
{
 statement;
}
```

```
initialization;
while ( loopContinuationTest ) {
   statement;
   increment;
}
```

```
do {
      statement;
   } while ( condition );
```

for、while 與 do while 迴圈的比較

迴圈特性	迴圈種類		
	for	while	do while
前端測試判斷條件	是	是	否
後端測試判斷條件	否	否	是
於迴圈主體中需要更改控制變數的值	否	是	是
迴圈控制變數會自動增加	是	否	否
迴圈重複的次數	已知	未知	未知
至少執行迴圈主體的次數	0 次	0 次	1 次
何時重複執行迴圈	條件成立	條件成立	條件成立

4.9 The `break` and `continue` Statements

- `break`
  - Causes immediate exit from a `while`, `for`, `do...while` or `switch` statement
  - Program execution continues with the first statement after the structure
  - Common uses of the `break` statement
    - Escape early from a loop
    - Skip the remainder of a `switch` statement

Using the `break` Statement in a `for` Statement

Outline

```
1  /* Fig. 4.11: fig04_11.c
2     Using the break statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13       /* if x is 5, terminate loop */
14       if ( x == 5 ) {
15          break; /* break loop only if x is 5 */
16       } /* end if */
17
18       printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nBroke out of loop at x == %d\n", x );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

fig04_11.c

Use `break` to break out of the loop at `x == 5` (當 `x == 5` 時，跳出 `for` 迴圈)

```
1 2 3 4
```

4.9 The break and continue Statements

- continue
  - Skips the remaining statements in the body of a while, for or do…while statement (不會跳出迴圈)
    - Proceeds with the next iteration of the loop
  - while and do…while
    - Loop-continuation test is evaluated immediately after the continue statement is executed
  - for
    - Increment expression is executed, then the loop-continuation test is evaluated

Using a continue Statement in a for Statement

Outline

fig04_12.

```
1  /* Fig. 4.12: fig04_12.c
2     Using the continue statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13       /* if x is 5, continue with next iteration of loop */
14       if ( x == 5 ) {
15          continue; /* skip remaining code in loop body */
16       } /* end if */
17
18       printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nUsed continue to skip printing the value 5\n" );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

Use continue to skip printing the value 5

x 值繼續加一，沒有跳出 for 迴圈，與 break 不同！！

如何將 for 迴圈改成 while 迴圈？需要注意哪些地方？

```
1 2 3 4 6 7 8 9 10

Used continue to skip printing the value 5
```


4.10 Logical Operators

- && (logical AND)
  - Returns true if both conditions are true
- || (logical OR)
  - Returns true if either of its conditions are true
- ! (logical NOT, logical negation)
  - Reverses the truth/falsity of its condition
  - Unary operator, has one operand
- Useful as conditions in loops

Expression	Result
true && false	false
true || false	true
!false	true

4.10 Logical Operators

expression1	expression2	expression1 && expression2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Fig. 4.13 | Truth table for the logical AND (&&) operator.

expression1	expression2	expression1 || expression2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Fig. 4.14 | Truth table for the logical OR (||) operator.

expression	!expression
0	1
nonzero	0

Fig. 4.15 | Truth table for operator ! (logical negation).

4.10 Logical Operators

- The Code

```
if ( a > b > c )  /* 錯誤!! */
```

should be

```
if ( a > b && b > c )
```

```
int a = 5, b = 4, c = 3;
if( a > b > c )
    printf( " a > b > c is true \n");
```
會列印嗎？

- The Code

```
if ( semesterAverage >= 90 || finalExam >= 90 )
   printf( "Student grade is A\n");
```

- The Codes (with x = 10, y = 1, a = 3, b = 3, g = 5, i = 2, j = 9)

```
!( x < 5 ) && !( y >= 7 )
!( a == b ) || !( g != 5 )
!( ( x <= 8 ) && ( y > 4 ) )
!( ( i > 4 ) || ( j <= 6 ) )
```


4.10 Logical Operators

Assume i = 1, j = 2, k = 3 and m = 2. What does each of the following statements print?

```
printf( "%d", i == 1 );                    ANS: 1
printf( "%d", j == 3 );                    ANS: 0
printf( "%d", i >= 1 && j < 4 );           ANS: 1
printf( "%d", m < = 99 && k < m );         ANS: 0
printf( "%d", j >= i || k == m );          ANS: 1
printf( "%d", k + m < j || 3 - j >= k );
                                           ANS: 0
printf( "%d", !m );                        ANS: 0
printf( "%d", !( j - m ) );                ANS: 1
printf( "%d", !( k > m ) );                ANS: 0
printf( "%d", !( j > k ) );                ANS: 1
```


4.10 Logical Operators

Operators	Associativity	Type
++ *(postfix)* -- *(postfix)*	right to left	postfix
+ - ! ++ *(prefix)* -- *(prefix)* (*type*)	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

Fig. 4.16 | Operator precedence and associativity.

4.11 Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are `true`, zero values are `false`
  - Example using `==`:

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );
```

    - Checks `payCode`, if it is 4 then a bonus is awarded

4.11 Confusing Equality (==) and Assignment (=) Operators

– Example, replacing == with =:

```
if ( payCode = 4 )
   printf( "You get a bonus!\n" );
```

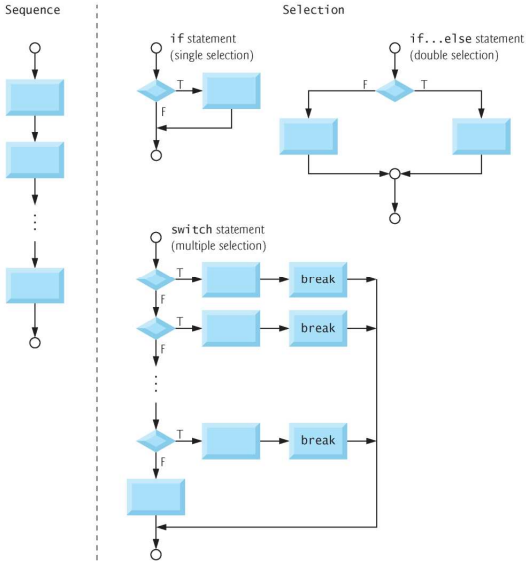
- This sets `payCode` to 4
- 4 is nonzero, so expression is `true`, and bonus awarded no matter what the `payCode` was

– Logic error, not a syntax error

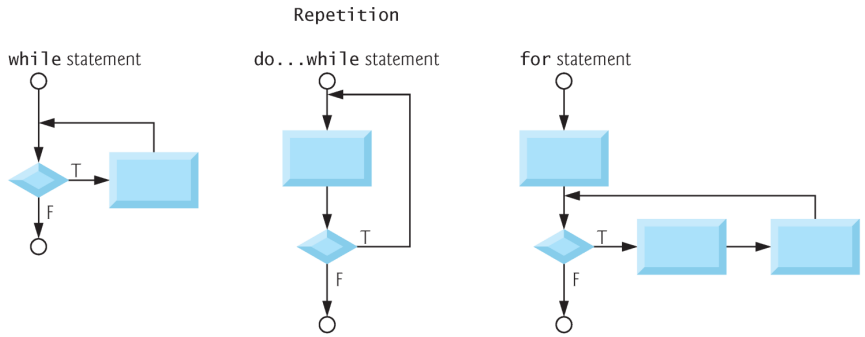
4.11 Confusing Equality (==) and Assignment (=) Operators

- lvalues
  - Expressions that can appear on the left side of an equation
  - Their values can be changed, such as variable names
    - `x = 4;`
- rvalues
  - Expressions that can only appear on the right side of an equation
  - Constants, such as numbers
    - Cannot write `4 = x;`
    - Must write `x = 4;`
  - lvalues can be used as rvalues, but not vice versa
    - `y = x;`

4.12 Structured-Programming Summary



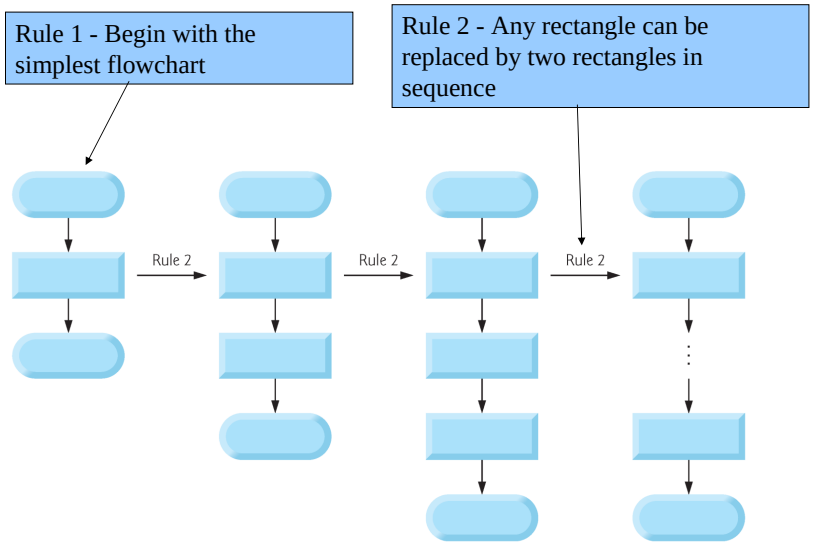
4.12 Structured-Programming Summary



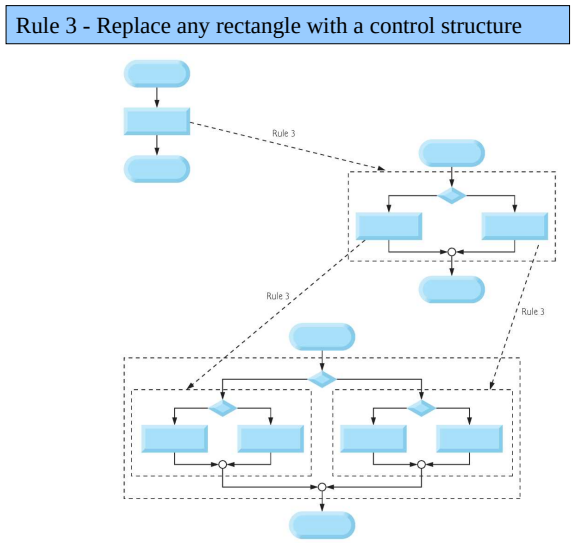
4.12 Structured-Programming Summary

- Structured Programming
  - Easier than unstructured programs to understand, test, debug and, modify programs
- Rules for Forming Structured Programming
  - Rules developed by programming community
  - Only single-entry/single-exit control structures are used
  - Rules:
    1. Begin with the "simplest flowchart"
    2. Stacking (堆疊) rule: Any rectangle (action) can be replaced by two rectangles (actions) in sequence
    3. Nesting (層狀) rule: Any rectangle (action) can be replaced by any control structure (sequence, `if`, `if…else`, `switch`, `while`, `do…while` or `for`)
    4. Rules 2 and 3 can be applied in any order and multiple times

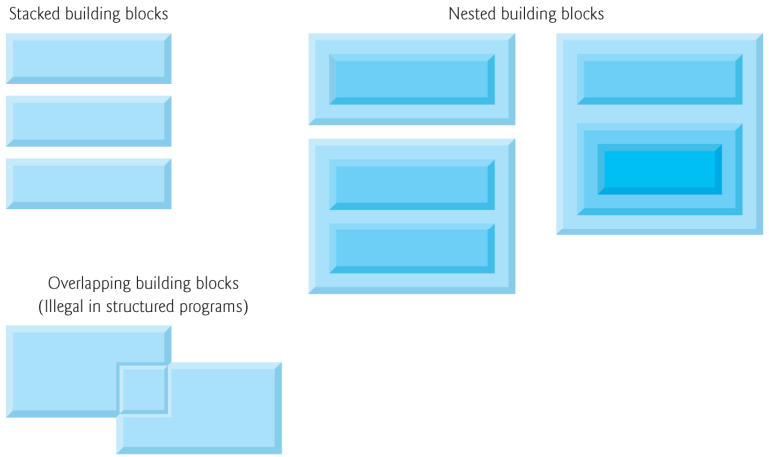
4.12 Structured-Programming Summary



4.12 Structured-Programming Summary

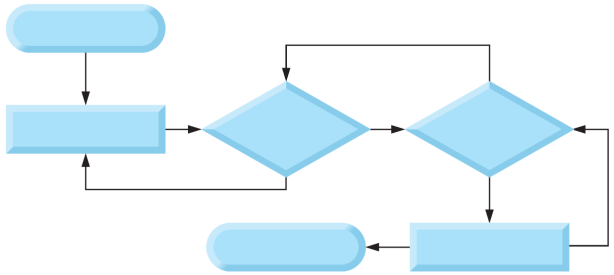


4.12 Structured-Programming Summary



4.12 Structured-Programming Summary

Figure 4.23 An unstructured flowchart.



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

Review

- In this chapter, we have learned:
  - To be able to use the `for` and `do...while` repetition statements.
  - To understand multiple selection using the `switch` selection statement.
  - To be able to use the `break` and `continue` program control statements
  - To be able to use the logical operators (`&&` , `||` , `!`)

Exercises