

計算機程式 OBJECT-ORIENTED PROGRAMMING 物件導向程式設計 DME1584

Lecture #02

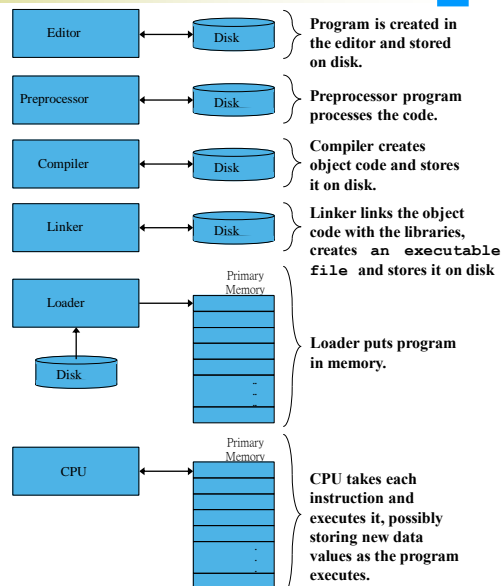
The C language Preview

Part 1

Basics of a Typical C++ Environment

Phases of C++ Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



Basics of a Typical C++ Environment

3

- Common Input/output functions

- **cin**
 - Standard input stream
 - Normally keyboard
- **cout**
 - Standard output stream
 - Normally computer screen
- **cerr**
 - Standard error stream
 - Display error messages

A Simple Program: Printing a Line of Text

4

- Before writing the programs

- Comments
 - Document programs
 - Improve program readability
 - Ignored by compiler
 - Single-line comment
 - Use C's comment `/* .. */` OR Begin with `//` or
- Preprocessor directives
 - Processed by preprocessor before compiling
 - Begin with `#`

5

```

1 // Fig. 1.2: fig01_02.cpp
2 // A first program
3 #include <iostream>
4 // function main body
5 int main()
6 {
7     std::cout << "Welcome to C++!" << endl;
8     return 0;
9 }
10
11
12

```

Single-line comments.

Function **main** returns an **int**.

Left brace { begins function body.

Right brace } ends function body.

Statements end with a semicolon ;.

Stream insertion operator.

Namespace **std**.

Keyword **return** is one of several means to exit function; value 0 indicates program terminated successfully.

fig01_02.cpp output (1 of 1)

Welcome to C++!

6

A Simple Program: Printing a Line of Text

- Standard output stream object
 - **std::cout**
 - “Connected” to screen
 - **<<**
 - Stream insertion operator
 - Value to right (right operand) inserted into output stream
- Namespace
 - **std::**: specifies using name that belongs to “namespace” **std**
 - **std::** removed through use of **using** statements
- Escape characters
 - ****
 - Indicates “special” character output

7

A Simple Program: Printing a Line of Text

Escape Sequence	Description
<code>\n</code>	Newline. Position the screen cursor to the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
<code>\a</code>	Alert. Sound the system bell.
<code>\\</code>	Backslash. Used to print a backslash character.
<code>\"</code>	Double quote. Used to print a double quote character.

8

Another Simple Program: Adding Two Integers

- Variables
 - Location in memory where value can be stored
 - Common data types
 - `int` - integer numbers
 - `char` - characters
 - `double` - floating point numbers
 - Declare variables with name and data type before use


```
int integer1;
int integer2;
int sum;
```
 - Can declare several variables of same type in one declaration
 - Comma-separated list


```
int integer1, integer2, sum;
```

9

Another Simple Program: Adding Two Integers

- Input stream object
 - >> (stream extraction operator)
 - Used with `std::cin`
 - Waits for user to input value, then press *Enter* (Return) key
 - Stores value in variable to right of operator
 - Converts value to variable data type
- = (assignment operator)
 - Assigns value to variable
 - Binary operator (two operands)
 - Example:


```
sum = variable1 + variable2;
```

```

1 // Fig. 1.6: fig01_06.cpp
2 // Addition program.
3 #include <iostream>
4
5 // function main begins program execution
6 int main()
7 {
8     int integer1; // first number to be input by user
9     int integer2; // second number to be input by user
10    int sum; // variable to hold the sum
11
12    std::cout << "Enter first integer: ";
13    std::cin >> integer1; // read an integer
14
15    std::cout << "Enter second integer\n"; // prompt
16    std::cin >> integer2;
17
18    sum = integer1 + integer2;
19
20    std::cout << "Sum is " << sum << std::endl; // print sum
21
22    return 0; // indicate that program ended successfully
23
24 } // end function main

```

Declare integer variables.

Use stream extraction operator with standard input stream to obtain user input.

Calculations can be performed in output statements: alternative for lines 18 and 20:

`std::cout << "Sum is " << integer1 + integer2 << std::endl;`

Concatenating, chaining or cascading stream insertion operations.



Outline

fig01_06.cpp
(1 of 1)

10

11

Memory Concepts

- Variable names

- Correspond to actual locations in computer's memory
- Every variable has name, type, size and value
- When new value placed into variable, overwrites previous value

- `std::cin >> integer1;`

- Assume user entered 45

integer1	45
----------	----

- `std::cin >> integer2;`

- Assume user entered 72

integer1	45
integer2	72

- `sum = integer1 + integer2;`

integer1	45
integer2	72
sum	117

12

Arithmetic

- Arithmetic calculations

- `*` : Multiplication
- `/` : Division
 - Integer division truncates remainder
 - `7 / 5` evaluates to 1
- `%` : Modulus operator returns remainder
 - `7 % 5` evaluates to 2

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
*, /, or %	Multiplication Division Modulus	Evaluated second. If there are several, they re evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

13

Decision Making: Equality and Relational Operators

- **if** structure
 - Make decision based on truth or falsity of condition
 - If condition met, body executed
 - Else, body not executed
- Equality and relational operators
 - Equality operators
 - Same level of precedence
 - Relational operators
 - Same level of precedence
 - Associate left to right
- **using** statements
 - Eliminate use of **std::** prefix
 - Write **cout** instead of **std::cout**

14

Decision Making: Equality and Relational Operators

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

```

1  // Fig. 1.14: fig01_14.cpp
2  // Using if statements, relational
3  // operators, and equality operators.
4  #include <iostream>
5
6  using std::cout; // program uses cout
7  using std::cin;  // program uses cin
8  using std::endl; // program uses endl
9
10 // function main begins program
11 int main()
12 {
13     int num1; // first number
14     int num2; // second number
15
16     cout << "Enter two integers, and I will tell you
17     << "the relationships they satisfy: ";
18     cin >> num1 >> num2; // read two integers
19
20     if ( num1 == num2 )
21         cout << num1 << " is equal to " << num2 << endl;
22     if ( num1 != num2 )
23         cout << num1 << " is not equal to " << num2 << endl;
24
25

```

Outline

fig01_14.cpp
(1 of 2)

15

using statements eliminate need for `std::` prefix.

Declare variables.

Can write `cout` and `cin` without `std::` prefix.

if structure compares values of `num1` and `num2`. If condition is true (i.e., values are equal), execute this statement.

if structure compares values of `num1` and `num2`. If condition is true (i.e., values are not equal), execute this statement.

```

26     if ( num1 < num2 )
27         cout << num1 << " is less than " << num2 << endl;
28
29     if ( num1 > num2 )
30         cout << num1 << " is greater than " << num2 << endl;
31
32     if ( num1 <= num2 )
33         cout << num1 << " is less than or equal to "
34         << num2 << endl;
35
36     if ( num1 >= num2 )
37         cout << num1 << " is greater than or equal to "
38         << num2 << endl;
39
40     return 0; // indicate that program ended successfully
41
42 } // end function main

```

Outline

fig01_14.cpp
output (1 of 2)

16

Statements may be split over several lines.

```

Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12

```


Algorithms

17

- Computing problems
 - Solved by executing a series of actions in a specific order
- Algorithm a procedure determining
 - Actions to be executed
 - Order to be executed
 - Example: recipe
- Program control
 - Specifies the order in which statements are executed

Pseudocode

18

- Pseudocode
 - Artificial, informal language used to develop algorithms
 - Similar to everyday English
- Not executed on computers
 - Used to think out program before coding
 - Easy to convert into C++ program
 - Only executable statements
 - No need to declare variables

19

Control Structures

- Sequential execution
 - Statements executed in order
- Transfer of control
 - Next statement executed *not* next one in sequence
 - Structured programming – “goto”-less programming
- 3 control structures to build any program
 - Sequence structure
 - Programs executed sequentially by default
 - Selection structures
 - `if`, `if/else`, `switch`
 - Repetition structures
 - `while`, `do/while`, `for`

20

Keywords

- C++ keywords
 - Cannot be used as identifiers or variable names

C++ Keywords

Keywords common to the C and C++ programming languages

<code>auto</code>	<code>break</code>	<code>case</code>	<code>char</code>	<code>const</code>
<code>continue</code>	<code>default</code>	<code>do</code>	<code>double</code>	<code>else</code>
<code>enum</code>	<code>extern</code>	<code>float</code>	<code>for</code>	<code>goto</code>
<code>if</code>	<code>int</code>	<code>long</code>	<code>register</code>	<code>return</code>
<code>short</code>	<code>signed</code>	<code>sizeof</code>	<code>static</code>	<code>struct</code>
<code>switch</code>	<code>typedef</code>	<code>union</code>	<code>unsigned</code>	<code>void</code>
<code>volatile</code>	<code>while</code>			

C++ only keywords

<code>asm</code>	<code>bool</code>	<code>catch</code>	<code>class</code>	<code>const_cast</code>
<code>delete</code>	<code>dynamic_cast</code>	<code>explicit</code>	<code>false</code>	<code>friend</code>
<code>inline</code>	<code>mutable</code>	<code>namespace</code>	<code>new</code>	<code>operator</code>
<code>private</code>	<code>protected</code>	<code>public</code>	<code>reinterpret_cast</code>	
<code>static_cast</code>	<code>template</code>	<code>this</code>	<code>throw</code>	<code>true</code>
<code>try</code>	<code>typeid</code>	<code>typename</code>	<code>using</code>	<code>virtual</code>
<code>wchar_t</code>				

21

Control Structures

- Flowchart
 - Graphical representation of an algorithm
 - Special-purpose symbols connected by arrows (flowlines)
 - Rectangle symbol (action symbol)
 - Any type of action
 - Oval symbol
 - Beginning or end of a program, or a section of code (circles)
- Single-entry/single-exit control structures
 - Connect exit point of one to entry point of the next
 - Control structure stacking

22

if Selection Structure

- Selection structure
 - Choose among alternative courses of action
 - Pseudocode example:
 - If student's grade is greater than or equal to 60*
 - Print "Passed"*
 - If the condition is **true**
 - Print statement executed, program continues to next statement
 - If the condition is **false**
 - Print statement ignored, program continues
 - Indenting makes programs easier to read
 - C++ ignores whitespace characters (tabs, spaces, etc.)

23

if Selection Structure

- Translation into C++

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

```
if ( grade >= 60 )
    cout << "Passed";
```

- Diamond symbol (decision symbol)

- Indicates decision is to be made
- Contains an expression that can be true or false
 - Test condition, follow path

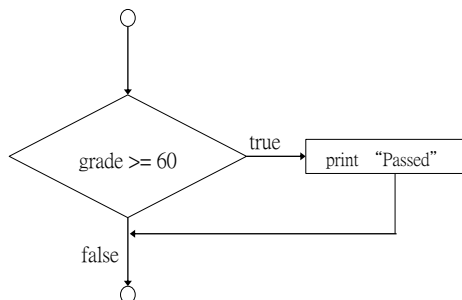
- **if** structure

- Single-entry/single-exit

24

if Selection Structure

- Flowchart of pseudocode statement



A decision can be made on any expression.

zero - **false**

nonzero - **true**

Example:

3 - 4 is true

25

if/else Selection Structure

- **if**
 - Performs action if condition true
- **if/else**
 - Different actions if conditions true or false
- Pseudocode


```
if student's grade is greater than or equal to 60
  print "Passed"
else
  print "Failed"
```
- C++ code


```
if ( grade >= 60 )
    cout << "Passed";
else
    cout << "Failed";
```

26

if/else Selection Structure

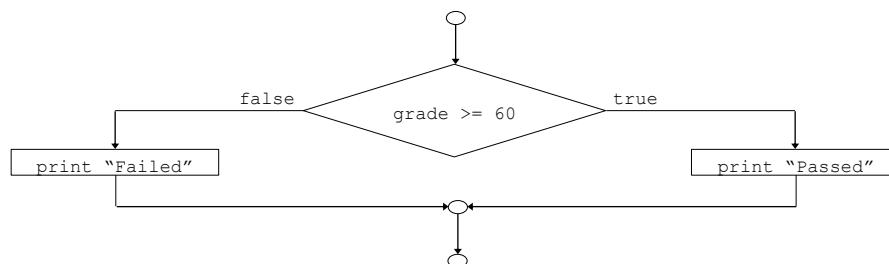
- Ternary conditional operator (?:)
 - Three arguments (condition, value if **true**, value if **false**)
- Code could be written:

```
cout << ( grade >= 60 ? "Passed" : "Failed" );
```

↑
Condition

↑
Value if true

↑
Value if false



27

if/else Selection Structure

- Nested **if/else** structures

- One inside another, test for multiple cases
- Once condition met, other statements skipped

*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"

28

if/else Selection Structure

- Example

```
if ( grade >= 90 )           // 90 and above
    cout << "A";
else if ( grade >= 80 )     // 80-89
    cout << "B";
else if ( grade >= 70 )     // 70-79
    cout << "C";
else if ( grade >= 60 )     // 60-69
    cout << "D";
else                         // less than 60
    cout << "F";
```

29

if/else Selection Structure

- Compound statement
 - Set of statements within a pair of braces


```
if ( grade >= 60 )
    cout << "Passed.\n";
else {
    cout << "Failed.\n";
    cout << "You must take this course again.\n";
}
```
 - Without braces,


```
cout << "You must take this course again.\n";
```

 always executed
- Block
 - Set of statements within braces

30

while Repetition Structure

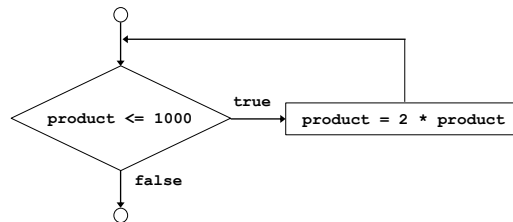
- Repetition structure
 - Action repeated while some condition remains true
 - Psuedocode
 - while there are more items on my shopping list*
 - Purchase next item and cross it off my list*
 - **while** loop repeated until condition becomes false
- Example


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

31

while Repetition Structure

- Flowchart of **while** loop



32

Counter-Controlled Repetition

- Counter-controlled repetition
 - Loop repeated until counter reaches certain value
- Definite repetition
 - Number of repetitions 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.


```

1  // Fig. 2.7: fig02_07.cpp
2  // Class average program with counter-controlled repetition.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // function main begins program execution
10 int main()
11 {
12     int total;           // sum of grades input by user
13     int gradeCounter;    // number of grade to be entered next
14     int grade;           // grade value
15     int average;         // average of grades
16
17     // initialization phase
18     total = 0;           // initialize total
19     gradeCounter = 1;    // initialize loop counter
20

```


[Outline](#)

33

fig02_07.cpp
(1 of 2)

```

21 // processing phase
22 while ( gradeCounter <= 10 ) {           // loop 10 times
23     cout << "Enter grade: ";             // prompt for input
24     cin >> grade;                         // read grade from user
25     total = total + grade;                // add grade to total
26     gradeCounter = gradeCounter + 1;     // increment counter
27 }
28
29 // termination phase
30 average = total / 10;                    // integer division
31
32 // display result
33 cout << "Class average is ";
34
35 return 0; // indicate program ended successfully
36
37 } // end function main

```


[Outline](#)

34

fig02_07.cpp
(2 of 2)

fig02_07.cpp
output (1 of 1)

The counter gets incremented each time the loop executes. Eventually, the counter causes the loop to end.

```

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

```

35

Sentinel-Controlled Repetition

- Suppose 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 program know when to end?
- Sentinel value
 - Indicates “end of data entry”
 - Loop ends when sentinel input
 - Sentinel chosen so it cannot be confused with regular input
 - -1 in this case

```

1  // Fig. 2.9: fig02_09.cpp
2  // Class average program with sentinel-controlled repetition.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::fixed;
9
10 #include <iomanip>           // parameterized stream manipulators
11
12 using std::setprecision; // sets numeric output precision
13
14 // function main begins program execution
15 int main()
16 {
17     int total;           // sum of grades
18     int gradeCounter;    // number of grades entered
19     int grade;           // grade value
20
21     double average;      // number with decimal point for average
22
23     // initialization phase
24     total = 0;           // initialize total
25     gradeCounter = 0;    // initialize loop counter

```

Data type **double** used to represent decimal numbers.



Outline

fig02_09.cpp
(1 of 3)

36

37
[Outline](#)
 fig02_09.cpp
 (2 of 3)

```

26
27 // processing phase
28 // get first grade from user
29 cout << "Enter grade, -1 to end: "; // prompt for input
30 cin >> grade; // read grade from user
31
32 // loop until sentinel value read from user
33 while ( grade != -1
34     total = total + g
35     gradeCounter = gr
36
37     cout << "Enter gr
38     cin >> grade;
39
40 } // end while
41
42 // termination phase
43 // if user entered at least one grade ...
44 if ( gradeCounter != 0 ) {
45
46     // calculate average of all grades entered
47     average = static_cast< double >( total ) / gradeCounter;
48

```

static_cast<double>() treats **total** as a **double** temporarily (casting).

Required because dividing two integers truncates the remainder.

gradeCounter is an **int**, but it gets *promoted* to **double**.

38
[Outline](#)
 fig02_09.cpp
 (3 of 3)
 fig02_09.cpp
 output (1 of 1)

```

49 // display average with two digits of precision
50 cout << "Class average is " << setprecision( 2 )
51     << fixed << average << endl;
52
53 } // end if part of if/else
54
55 else // if no grades were entered, output appropriate message
56     cout << "No grades were entered" << endl;
57
58 return 0; // indicate program ended successfully
59
60 } // end function main

```

setprecision(2) prints two digits past decimal point (rounded to fit precision).

Programs that use this must include **<iomanip>**

fixed forces output to print in fixed point format (not scientific notation). Also, forces trailing zeros and decimal point to print.

Include **<iostream>**

```

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

```

39

switch Multiple-Selection Structure

• switch

- Test variable for multiple values
- Series of **case** labels and optional **default** case

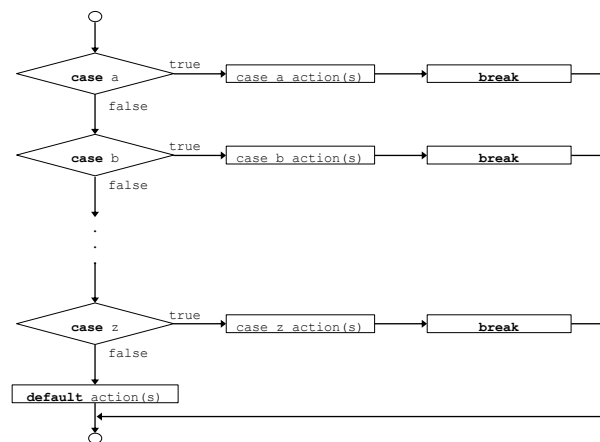
```
switch ( variable ) {
    case value1:      // taken if variable == value1
        statements
        break;       // necessary to exit switch

    case value2:
    case value3:      // taken if variable == value2 or == value3
        statements
        break;

    default:          // taken if none matches
        statements
        break;
}
```

40

switch Multiple-Selection Structure



41

switch Multiple-Selection Structure

- Example upcoming
 - Program to read grades (A-F)
 - Display number of each grade entered
 - Details about characters
 - Single characters typically stored in a **char** data type
 - **char** a 1-byte integer, so **chars** can be stored as **ints**
 - Can treat character as **int** or **char**
 - 97 is the numerical representation of lowercase 'a' (ASCII)
 - Use *single quotes* to get numerical representation of character
- cout << "The character (" << 'a' << ") has the value "
 << static_cast< int > ('a') << endl;
- Prints
- The character (a) has the value 97

```

1  // Fig. 2.22: fig02_22.cpp
2  // Counting letter grades.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // function main begins program execution
10 int main()
11 {
12     int grade;          // one grade
13     int aCount = 0;     // number of As
14     int bCount = 0;     // number of Bs
15     int cCount = 0;     // number of Cs
16     int dCount = 0;     // number of Ds
17     int fCount = 0;     // number of Fs
18
19     cout << "Enter the letter grades." << endl
20          << "Enter the EOF character to end input." << endl;
21

```


[Outline](#)

42

 fig02_22.cpp
 (1 of 4)

43

```

22 // loop until user types end-of-file key sequence
23 while ( ( grade = cin.get() ) != EOF ) {
24     // determine which grade was input
25     switch ( grade ) { // switch structure
26         case 'A': // grade was uppercase A
27             ++aCount; // increment aCount
28             break;
29         case 'a': // or lowercase a
30             ++aCount; // increment aCount
31             break;
32         case 'B': //
33             ++bCount; //
34             break;
35         case 'b': //
36             ++bCount; //
37             break;
38         // ...
39         // ...
40         // ...
41         // ...
42         // ...

```

break causes **switch** to end and the program continues with the first statement after the **switch** structure.

cin.get() uses dot notation (explained chapter 6). This function gets 1 character from the keyboard (after *Enter* pressed), and it is assigned to **grade**.

cin.get() returns EOF (end-of-file) after the EOF character is input, to indicate the end of data. EOF may be ctrl-d or ctrl-z, depending on your OS.

Assignment statements have a value, which is the same as the variable on the left of the **=**. The value of this statement is the same as the value returned by **cin.get()**.

This can also be used to initialize multiple variables:
a = b = c = 0;

Compares **grade** (an **int**) to the numerical representations of **A** and **a**.

Outline

fig02_22.cpp

44

```

43     case 'D': // grade was uppercase D
44     case 'd': // or lowercase d
45         ++dCount; // increment dCount
46         break; // exit switch
47
48     case 'F': // grade was
49     case 'f': // or lowercase f
50         ++fCount; // increment fCount
51         break; // exit switch
52
53     case '\n': // ignore newline
54     case '\t': // tabs,
55     case ' ': // and space
56         break; // exit switch
57
58     default: // catch all other characters
59         cout << "Incorrect letter grade entered."
60             << " Enter a new grade." << endl;
61         break; // optional; will exit switch anyway
62
63 } // end switch
64
65 } // end while
66

```

This test is necessary because *Enter* is pressed after each letter grade is input. This adds a newline character that must be removed. Likewise, we want to ignore any whitespace.

Notice the **default** statement, which catches all other cases.

Outline

fig02_22.cpp
(3 of 4)

```

67 // output summary of results
68 cout << "\n\nTotals for each letter grade are:"
69     << "\nA: " << aCount // display number of A grades
70     << "\nB: " << bCount // display number of B grades
71     << "\nC: " << cCount // display number of C grades
72     << "\nD: " << dCount // display number of D grades
73     << "\nF: " << fCount // display number of F grades
74     << endl;
75
76 return 0; // indicate successful termination
77
78 } // end function main

```


[Outline](#)

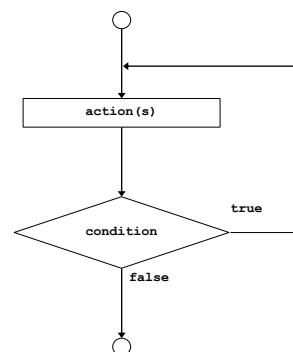
fig02_22.cpp
(4 of 4)

45

do/while Repetition Structure

- Similar to **while** structure
 - Makes loop continuation test at end, not beginning
 - Loop body executes at least once
- Format


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





46

```

1  // Fig. 2.24: fig02_24.cpp
2  // Using the do/while repetition structure.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program execution
9  int main()
10 {
11     int counter = 1;
12
13     do {
14         cout << counter << " "; // display counter
15     } while ( ++counter <= 10 ); // end do/while
16
17     cout << endl;
18
19     return 0; // indicate successful termination
20
21 } // end function main

```

Outline

fig02_24.cpp
(1 of 1)

fig02_24.cpp
output (1 of 1)

47

1 2 3 4 5 6 7 8 9 10

Notice the preincrement in loop-continuation test.

break and continue Statements


48


- **break** statement
 - Immediate exit from **while**, **for**, **do/while**, **switch**
 - Program continues with first statement after structure
- Common uses
 - Escape early from a loop
 - Skip the remainder of **switch**


```

1  // Fig. 2.26: fig02_26.cpp
2  // Using the break statement in a for structure.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program execution
9  int main()
10 {
11
12     int x; // x declared here so it can be used after the loop
13
14     // loop 10 times
15     for ( x = 1; x <= 10; x++ ) {
16
17         // if x is 5, terminate loop
18         if ( x == 5 )
19             break;           // break loop only if x is 5
20
21         cout << x << " "; // display value of x
22
23     } // end for
24
25     cout << "\nBroke out of loop when x became " << x << endl;

```





Outline

fig02_26.cpp
(1 of 2)

49

Exits for structure when break executed.

Logical Operators

50

- Used as conditions in loops, **if** statements
- **&&** (logical **AND**)
 - **true** if both conditions are **true**
 - if (gender == 1 && age >= 65)
 - ++seniorFemales;
- **||** (logical **OR**)
 - **true** if either of condition is **true**
 - if (semesterAverage >= 90 || finalExam >= 90)
 - cout << "Student grade is A" << endl;

51

Logical Operators

- **!** (logical **NOT**, logical negation)
 - Returns **true** when its condition is **false**, & vice versa
 - ```
if (!(grade == sentinelValue))
 cout << "The next grade is " << grade << endl;
```
  - Alternative:
 

```
if (grade != sentinelValue)
 cout << "The next grade is " << grade << endl;
```

52

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

- Common error
  - Does not typically cause syntax errors
- Aspects of problem
  - Expressions that have a value can be used for decision
    - Zero = false, nonzero = true
  - Assignment statements produce a value (the value to be assigned)

53

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

- Example

```
if (payCode == 4)
 cout << "You get a bonus!" << endl;
```

- If paycode is 4, bonus given

- If == was replaced with =

```
if (payCode = 4)
 cout << "You get a bonus!" << endl;
```

- Paycode set to 4 (no matter what it was before)
- Statement is true (since 4 is non-zero)
- Bonus given in every case

54

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

- Lvalues

- Expressions that can appear on left side of equation
- Can be changed (I.e., variables)

```
x = 4;
```

- Rvalues

- Only appear on right side of equation
- Constants, such as numbers (i.e. cannot write 4 = x;)

- Lvalues can be used as rvalues, but not vice versa

55

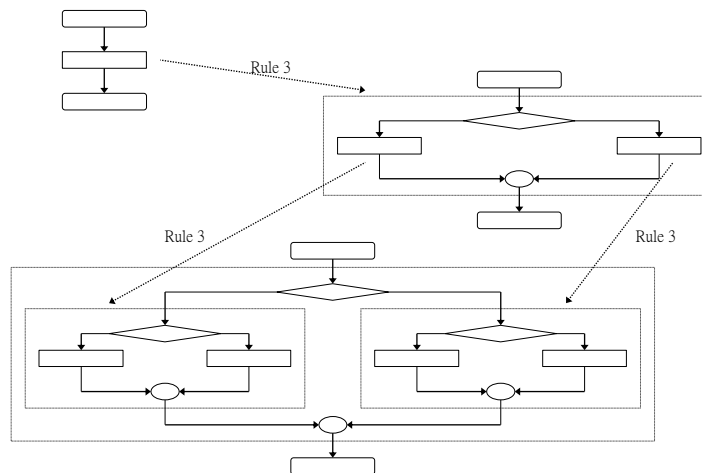
## Structured-Programming Summary

- Structured programming
  - Programs easier to understand, test, debug and modify
- Rules for structured programming
  - Only use single-entry/single-exit control structures
  - Rules
    - 1) Begin with the “simplest flowchart”
    - 2) Any rectangle (action) can be replaced by two rectangles (actions) in sequence
    - 3) 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

56

## Structured-Programming Summary

### Representation of Rule 3 (replacing any rectangle with a control structure)



57

## Structured-Programming Summary

- All programs broken down into
  - Sequence
  - Selection
    - **if**, **if/else**, or **switch**
    - Any selection can be rewritten as an **if** statement
  - Repetition
    - **while**, **do/while** or **for**
    - Any repetition structure can be rewritten as a **while** statement

58

## Practice Time

1. Using Visual Studio to start a new project
2. Download “Lab001.cpp” on E3 website and add it into your project.
3. Rewrite the Lab01.cpp with the following function
  - Adding a variable : Student ID number (學號)
  - Count # of (0, 1, 2, ....., 9) ,  
 例如有幾個 0, 幾個 1.....  
 Show the results on screen 顯示在螢幕上
4. Upload the whole project on E3