Computer Programming
Lecture 4

Hung-Yu Wei
Department of Electrical Engineering
National Taiwan University

Chapter 3: Control Statements

- [review] We have learned
  - If
    - single selection
  - If … else
    - double selection
  - While
    - Repetition

- Now, we will learn
  - How to formulate algorithms with these control statements

Overview: Algorithms in Control Statements

- Counter-Controlled Repetition
- Sentinel-Controlled Repetition
- Nested Control Statement

Problem solving

- Analyze a problem
- Develop an algorithm to solve the problem
- Pseudocodes
- UML activity diagram
- C++ source codes

Algorithm: Counter-controlled repetition

- Counter
  - Count the number of times a group of statements will be executed
  - =iterations of the loop
- Definite repetition (確定次數的loop)
  - Known number of repetitions
  - Contrary to "indefinite repetition"
    - unknown number of repetitions

Counter-controlled repetition

- Syntax
  - An example that run 10 times

```
int counter=1;
while (counter <= 10)
{
  statements to be executed
  counter=counter+1;
}
```

Be careful about the counter

- Initialization
  - int counter=1; OR
  - int counter=0;
- Condition in while
  - while (counter <= 10)
  - while (counter < 10)
- You might run 1 more time (or 1 time less)
  - Check it during programming/debugging

An example problem

- A class of 10 students took a quiz. The grades of the quiz are integer between 0~100. Calculate and display the total of all grades and the class average.

Analyze the problem

- You know the number of student grades → definite repetition
- You could use counter-controlled repetition
- You need to compute the sum of all grades
- You need to compute the average (sum/number of grades)

Pseudocodes for the example

```
1  Set total to zero
2  Set grade counter to one
3
4  While grade counter is less than or equal to ten
5      Prompt the user to enter the next grade
6      Input the next grade
7      Add the grade into the total
8      Add one to the grade counter
9
10 Set the class average to the total divided by ten
11 Print the total of the grades for all students in the class
12 Print the class average
```

```
1  // Fig. 3.8: fig03_08.cpp
2  // Class average program with counter-controlled repetition.
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8     int total; // sum of grades entered by user
9     int gradeCounter; // number of the grade to be entered next
10    int grade; // grade value entered by user
11    int average; // average of grades
12
13    // initialization phase
14    total = 0; // initialize total
15    gradeCounter = 1; // initialize loop counter
16
17    // processing phase
18    while ( gradeCounter <= 10 ) // loop 10 times
19    {
20       cout << "Enter grade: "; // prompt for input
21       cin >> grade; // input next grade
22       total = total + grade; // add grade to total
23       gradeCounter = gradeCounter + 1; // increment counter by 1
24    } // end while
```

Fig. 3.8 | Class average problem using counter-controlled repetition. (Part 1 of 2.)

```
25
26      // termination phase
27      average = total / 10; // integer division yields integer result
28
29      // display total and average of grades
30      cout << "\nTotal of all 10 grades is " << total << endl;
31      cout << "Class average is " << average << endl;
32   } // end main
```

```
Enter grade: 67
Enter grade: 78
Enter grade: 89
Enter grade: 67
Enter grade: 87
Enter grade: 98
Enter grade: 93
Enter grade: 85
Enter grade: 82
Enter grade: 100

Total of all 10 grades is 846
Class average is 84
```

Fig. 3.8 | Class average problem using counter-controlled repetition. (Part 2 of 2.)

Floating-point numbers

- Examples:
  - 1.4523, 592.245, 0.0083
- Data types
  - Float: single precision floating-point numbers
    - 7 bits
  - Double: double precision floating-point numbers
    - 15 bits
- C++ Default: double

Explicit conversion

- static_cast <double> (variable)
  - Change data type temporarily
- Example

```
int num_people;
int sum_grade;
avg_grade=static_cast<double>(sum_grade)/num_people;
```

- Results
  - avg_grade is double
  - num_people and sum_grade are still integer

More about *static_cast*

- Sample Code

```
avg=static_cast<double>(total)/cnt;
```

- Wrong result…do you know why?

```
avg=static_cast<double>(total/cnt);
```

- This is okay

```
avg=static_cast<double>(total)/static_cast<double>(cnt);
```

Format floating-point numbers

- setprecision
  - setprecision (x): 小數下 x 位
  - #include <iomanip>
- fixed
  - Fixed-point format 12.3
  - #include <iostream>
- scientific
  - Scientific format 1.23×10^{2}
  - #include <iostream>
- showpoint
  - Don't show trailing zeros
  - #include <iostream>

Algorithm: Sentinel-Controlled Repetition

- Use a special value (sentinel value)
  - Also known as flag value, or signal value
- Indefinite repetition
  - Unknown number of iterations
- Example problem:

  Develop a class average program that processes grades of an arbitrary numbers of students.

Case Study: Sentinel-Controlled Repetition

- Example

```
while (flag_value != -1)
{
   statements;
}
```

- "flag_value" is sentinel value in this case
- The while loop has "*unknown number of iterations*"
  - Indefinite repetition

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

```
1  // Fig. 3.10: fig03_10.cpp
2  // Class average program with sentinel-controlled repetition.
3  #include <iostream>
4  #include <iomanip> // parameterized stream manipulators
5  using namespace std;
6
7  // determine class average based on 10 grades entered by user
8  int main()
9  {
10    int total; // sum of grades entered by user
11    int gradeCounter; // number of grades entered
12    int grade; // grade value
13    double average; // number with decimal point for average
14
15    // initialization phase
16    total = 0; // initialize total
17    gradeCounter = 0; // initialize loop counter
18
19    // processing phase
20    // prompt for input and read grade from user
21    cout << "Enter grade or -1 to quit: ";
22    cin >> grade; // input grade or sentinel value
23
```

Fig. 3.10 | Class average problem using sentinel-controlled repetition: `GradeBook` source code file. (Part 1 of 3.)

```
24      // loop until sentinel value read from user
25      while ( grade != -1 ) // while grade is not -1
26      {
27         total = total + grade; // add grade to total
28         gradeCounter = gradeCounter + 1; // increment counter
29
30         // prompt for input and read next grade from user
31         cout << "Enter grade or -1 to quit: ";
32         cin >> grade; // input grade or sentinel value
33      } // end while
34
```

Fig. 3.10 | Class average problem using sentinel-controlled repetition: `GradeBook` source code file. (Part 2 of 3.)

```
35      // termination phase
36      if ( gradeCounter != 0 ) // if user entered at least one grade...
37      {
38         // calculate average of all grades entered
39         average = static_cast< double >( total ) / gradeCounter;
40
41         // display total and average (with two digits of precision)
42         cout << "\nTotal of all " << gradeCounter << " grades entered is "
43            << total << endl;
44         cout << "Class average is " << setprecision( 2 ) << fixed << average
45            << endl;
46      } // end if
47      else // no grades were entered, so output appropriate message
48         cout << "No grades were entered" << endl;
49   } // end main
```

```
Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of all 3 grades entered is 257
Class average is 85.67
```

Fig. 3.10 | Class average problem using sentinel-controlled repetition: GradeBook source code file. (Part 3 of 3.)

Algorithm: Nested Control Statement

- Previously, control statements are used "*in sequence*" (one after another)
- Now, we could also use control statements "by nesting" (one within another)

Sample problem

- 10 students took an exam. We will input the exam results (1 means pass; 2 means fail). Count and print the number of passed and failed students. If more than 8 students passed, display “give bonus to instructor”

```
1  Initialize passes to zero
2  Initialize failures to zero
3  Initialize student counter to one
4
5  While student counter is less than or equal to 10
6      Prompt the user to enter the next exam result
7      Input the next exam result
8
9      If the student passed
10         Add one to passes
11     Else
12         Add one to failures
13
14     Add one to student counter
15
16 Print the number of passes
17 Print the number of failures
18
19 If more than eight students passed
20     Print "Bonus to instructor!"
```

Fig. 3.11 | Pseudocode for examination-results problem.

```
1  // Fig. 3.12: fig03_12.cpp
2  // Examination-results problem: Nested control statements.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     // initializing variables in declarations
9     int passes = 0; // number of passes
10    int failures = 0; // number of failures
11    int studentCounter = 1; // student counter
12    int result; // one exam result (1 = pass, 2 = fail)
13
14    // process 10 students using counter-controlled loop
15    while ( studentCounter <= 10 )
16    {
17       // prompt user for input and obtain value from user
18       cout << "Enter result (1 = pass, 2 = fail): ";
19       cin >> result; // input result
20
```

Fig. 3.12 | Examination-results problem: Nested control statements. (Part 1 of 4.)

```
21          // if...else nested in while
22          if ( result == 1 )           // if result is 1,
23             passes = passes + 1;      // increment passes;
24          else                         // else result is not 1, so
25             failures = failures + 1; // increment failures
26
27          // increment studentCounter so loop eventually terminates
28          studentCounter = studentCounter + 1;
29       } // end while
30
31       // termination phase; display number of passes and failures
32       cout << "Passed " << passes << "\nFailed " << failures << endl;
33
34       // determine whether more than eight students passed
35       if ( passes > 8 )
36          cout << "Bonus to instructor!" << endl;
37    } // end main
```

```
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): 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
Passed 9
Failed 1
Bonus to instructor!
```

Assignment operators

- They are the same

  x=x+3;

  x += 3;

- +=
- -=
- *=
- /=
- %=

Examples

- Remember the “memory operation”
- int c=3, d=5, e=4, f=6, g=12;
- c+=7;
  - Assign 10 to c
- d-=4;
  - Assign 1 to d
- e*=5;
  - Assign 20 to e
- f/=3;
  - Assign 2 to f
- g%=9;
  - Assign 3 to g

Increment and Decrement Operators

- Increment operator
  - ++
    - ++x
    - x++
- Decrement operator
  - --
    - --x
    - x--
- No space between “++” and “x”
- When ++ and – occurs in a statement by itself, the preincrement and postincrement have the same effect.
  - x=x+1;
  - ++x;
  - x++;

Continued

- ++x
  - preincrement
  - **First increment** x by 1. Then use the **new** x value in the expression.
- --x
  - predecrement
  - **First decrement** x by 1. Then use the **new** x value in the expression.
- x++
  - Postincrement
  - First use the **old** x value in the expression. Then increment x by 1.
- x--
  - postdecrement
  - First use the **old** x value in the expression. Then decrement x by 1.

```
1  // Fig. 3.15: fig03_15.cpp
2  // Preincrementing and postincrementing.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int c;
9
10    // demonstrate postincrement
11    c = 5; // assign 5 to c
12    cout << c << endl; // print 5
13    cout << c++ << endl; // print 5 then postincrement
14    cout << c << endl; // print 6
15
16    cout << endl; // skip a line
17
18    // demonstrate preincrement
19    c = 5; // assign 5 to c
20    cout << c << endl; // print 5
21    cout << ++c << endl; // preincrement then print 6
22    cout << c << endl; // print 6
23 } // end main
```


3.15 | Preincrementing and postincrementing. (Part 1 of 2.)

Operators						Associativity	Type
::						left to right	scope resolution
()						left to right	parentheses
++	--	static_cast<*type*>()				left to right	unary (postfix)
++	--	+	-			right to left	unary (prefix)
*	/	%				left to right	multiplicative
+	-					left to right	additive
<<	>>					left to right	insertion/extraction
<	<=	>	>=			left to right	relational
==	!=					left to right	equality
?:						right to left	conditional
=	+=	-=	*=	/=	%=	right to left	assignment

Fig. 3.16 | Operator precedence for the operators encountered so far in the text.

[Summary] Chapter 3

- Selection
  - if
  - if… else
- Repetition
  - while
- +=, -=, *=, /=
- ++,--
- Repetition algorithm
  - Counter-controlled repetition
  - Sentinel-controlled repetition