COMP1011
Programming Fundamentals

Lecture 3
Control Structures II

Lecture 3

- Repetition Structures
  - **`while`**
- Hints on Constructing Repetition Structures
- Constants
- Nested Control Structures

Repetition structures

- Suppose we want to display 1 to 100 on the screen
- Tedious to write 100 **`cout`** statements (even you perform copy-and-paste)
- Repetition structures help simplify the code
- Action repeated while some condition remains **`true`**
- Three kinds of structures
  - **`while`**, **`do-while`** and **`for`**

while Repetition Structure

while.cpp

```cpp
// Demonstrating while Repetition Structure
// To print from 1 to 100
#include <iostream>

using namespace std;

int main() {

    int counter = 1;     // start the counter

    while (counter <= 100) {     // check the condition
        cout << counter << endl;         // print the value
        counter++;                       // update the counter
    }

    return 0;
}
```

```
1
2
3
(More numbers are to be printed. We skip here)
```

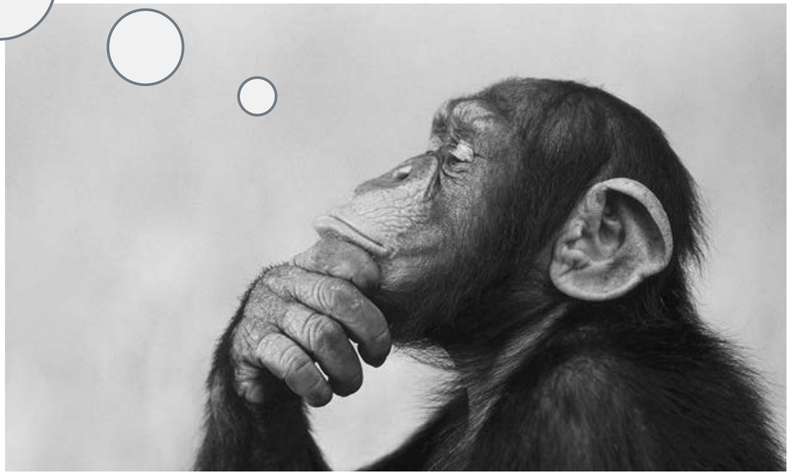
`while` Repetition Structure

› General format

```
while (condition) {
     loop body
}
```

Hints on Constructing Repetition Structures

Understanding the syntaxes of *for* and *while* is not difficult, but how about formulating the algorithm using the structures when given a problem?



Formulating Algorithms (Counter-Controlled Repetition)

- Counter-controlled repetition
  - Loop repeated until counter reaches certain value
- Definite repetition
  - Number of repetitions known
- For example,

A class of ten students took a quiz. The marks (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.

Formulating Algorithms (Counter-Controlled Repetition)

› Pseudocode

```
Set total to zero
Set counter to one
While counter is less than or equal to ten
        Input the next mark
        Add the mark into the total
        Add one to the counter
Set the class average to the total divided by ten
Print the class average
```

Formulating Algorithms (Counter-Controlled Repetition)

› Another version (less wordy)

```
total $\leftarrow$ 0
counter $\leftarrow$ 1
While counter $\leq$ 10
        Input mark
        total $\leftarrow$ total + mark
        counter $\leftarrow$ counter + 1
average $\leftarrow$ total/10
Print average
```

while2.cpp (Page 1 of 2)

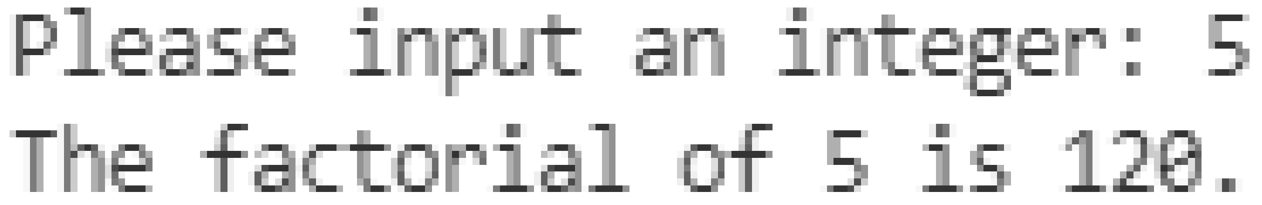
```
// Class average program with counter-controlled repetition.
#include <iostream>

using namespace std;

int main() {

    double total;          // sum of marks input by user
    int counter;           // keep track of the number of marks entered
    int mark;              // mark value
    double average;        // average of marks

    // initialization phase
    total = 0;          // initialize total
    counter = 1;        // initialize loop counter

    // processing phase
    while (counter <= 10) {      // loop 10 times
        cout << "Enter a mark: ";  // prompt for input
        cin >> mark;               // read mark from user
        total = total + mark;      // add mark to total
        counter = counter + 1;     // increment counter
    }
    // termination phase
    average = total / 10;          // integer division

    // display result
    cout << "Class average is " << average << endl;

    // indicate program ended successfully
    return 0;
}
```

while2.cpp (Page 2 of 2)

```
Enter a mark: 98
Enter a mark: 76
Enter a mark: 71
Enter a mark: 87
Enter a mark: 83
Enter a mark: 90
Enter a mark: 57
Enter a mark: 79
Enter a mark: 82
Enter a mark: 84
Class average is 80.7
```

Exercise

- In mathematics, the factorial of a non-negative integer n is the product of all positive integers less than or equal to n, denoted by **n!**.
- For example, 5! is equal to 1 x 2 x 3 x 4 x 5 = 120.
- Also, there is a special case where 0! = 1.
- Write a program that uses the **while** repetition structure to perform the factorial calculation for an input integer n.
- Here is a sample input and output:



```
Please input an integer: 5
The factorial of 5 is 120.
```

Exercise

› Write down your code here:

Formulating Algorithms (Counter-Controlled Repetition)

- 3 phases
  - Initialization
    - Initializes the program variables
  - Processing
    - Input data, adjusts program variables
  - Termination
    - Calculate and print the final results
- Helps break down programs for top-down refinement

Formulating Algorithms (Sentinel-Controlled Repetition)

- Suppose problem becomes

  Develop a class-averaging program that will process an arbitrary number of marks 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

    - For example, -1 (because the regular input range is [0, 100])

Formulating Algorithms (Sentinel-Controlled Repetition)

- Refine the initialization phase
  - *Initialize variables*

```
total ← 0
counter ← 0
```

- Processing
  - *Input, sum and count the quiz marks*

```
Input mark
While mark ≠ sentinel
   total ← total + mark
   counter ← counter + 1
   Input mark
```

Formulating Algorithms (Sentinel-Controlled Repetition)

- Termination

  - *Calculate and print the class average*

```
If counter $\neq$ 0
   average $\leftarrow$ total/counter
   Print average
Else
   Print “No marks were entered.”
```

while3.cpp (Page 1 of 2)

```cpp
// Class average program with sentinel-controlled repetition.
#include <iostream>

using namespace std;

int main() {

    double total;         // sum of marks input by user
    int counter;          // keep track of the number of marks entered
    int mark;            // mark value
    double average;       // average of marks

    // initialization phase
    total = 0;                                // initialize total
    counter = 0;                              // initialize counter

    cout << "Enter a mark, -1 to end: ";  // prompt for input
    cin >> mark;                              // read mark from user

    // processing phase
    while (mark != -1) {
        total = total + mark;                     // add mark to total
        counter = counter + 1;                    // increment counter
        cout << "Enter a mark, -1 to end: ";  // prompt for input
        cin >> mark;                              // read mark from user
    }
    // termination phase
    if (counter != 0) {
        average = total / counter;                         // integer division
        cout << "Class average is " << average << endl;    // display result
    }
    else {
        cout << "No marks were entered." << endl;       // display another result
    }

    // indicate program ended successfully
    return 0;
}
```

while3.cpp (Page 2 of 2)

```
Enter a mark, -1 to end: 75
Enter a mark, -1 to end: 94
Enter a mark, -1 to end: 97
Enter a mark, -1 to end: 88
Enter a mark, -1 to end: 70
Enter a mark, -1 to end: 64
Enter a mark, -1 to end: 83
Enter a mark, -1 to end: 89
Enter a mark, -1 to end: -1
Class average is 82.5
```

Constants

› Some values in the world are fixed. E.g.,

```
∏ = 3.14159
```

› Tedious to type every time in the program

› Unlike variables, the value of a constant must be assigned once declared

› It cannot be altered anywhere in the program

Constants

› To declare a constant, e.g.,

```
const double PI = 3.14159;
```

› Use it like a variable

```
area = PI * r * r;
```

› But,

```
PI = 10;
// Wrong!! It is a constant!
```

› For constant names, we usually use **CAPITAL** letters. Again, not compulsory, but recommended.

constant.cpp

```cpp
// Demonstrating how constant works
#include <iostream>

using namespace std;

int main() {

    const double PI = 3.14159;
    double r;

    cout << "Please enter the radius: ";

    cin >> r;

    cout << "Area of Circle is " << PI * r * r << endl;

    return 0;
}
```

```
Please enter the radius: 123
Area of Circle is 47529.1
```

Nested Control Structures

- Problem
  A class 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 "Very Good!". Otherwise, print "Very Bad!"
- Note that
  - Program processes 10 results
    - Fixed number, so we should use ***counter-controlled*** loop
  - Two counters should be used
    - One for counting number of passes
    - Another for counting number of fails
  - Assume 1 means PASS and 2 means FAIL

Nested Control Structures

- Top level outline
  - Analyze test results and comment on the overall result
- First refinement

```
Initialize variables
Input the ten test results and count passes and fails
Print a summary of the test results and a comment on the students'
    performance
```

- Refine
  - *Initialize variables*

```
passes ← 0
fails ← 0
counter ← 1
```

Nested Control Structures

› Refine

– *Input the ten test results and count passes and fails*

```
While counter $\leq 10$
   Input the next test result
   If the student passed
      passes $\leftarrow$ passes + 1
   Else
      fails $\leftarrow$ fails + 1
   counter $\leftarrow$ counter + 1
```

Nested Control Structures

- Refine
  - *Print a summary of the test results and a comment on the students' performance*

```
Print number of passes
Print number of fails
If passes > 8
   Print "Very Good!"
Else
   Print "Very Bad!"
```

Nested Control Structures

nested.cpp (Page 1 of 2)

```
// Demonstrating Nested Control Structures
#include <iostream>

using namespace std;

int main() {

    // initialize variables in declarations
    int passes = 0;            // number of passes
    int fails = 0;             // number of fails
    int studentCounter = 1;    // student counter
    int input;                 // one test result
    const int PASS = 1;
    const int FAIL = 2;

    // process 10 students using counter-controlled loop
    while (studentCounter <= 10) {

        // prompt user for input and obtain value from user
        cout << "Enter result (1 = PASS, 2 = FAIL): ";
        cin >> input;
```

```cpp
        // add to the correct counter
        if (input == PASS) {
                passes = passes + 1;
        }
        else {
                fails = fails + 1;
        }

        // increment studentCounter so loop eventually terminates
        studentCounter = studentCounter + 1;
    }

    // termination phase; display number of passes and failures
    cout << "Passed " << passes << endl;
    cout << "Failed " << fails << endl;

    // decide what to print based on the number of pass and fails
    if (passes > 8) {
        cout << "Very Good!" << endl;
    }
    else {
        cout << "Very Bad!" << endl;
    }

    return 0;
}
```

nested.cpp (Page 2 of 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): 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
Very Good!
```

```
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): 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): 2
Enter result (1 = PASS, 2 = FAIL): 2
Passed 6
Failed 4
Very Bad!
```

Summary

- Repetition Structures
  - `while`
- Hints on Constructing Repetition Structures
- Constants
- Nested Control Structures