Lecture #5
Loops

chapter: 1.4.1-1.4.3

Motivations

Suppose that you need to print a string (e.g., "Welcome to C++!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
cout  << "Welcome to C++!" << endl;
```

So, how do you solve this problem?

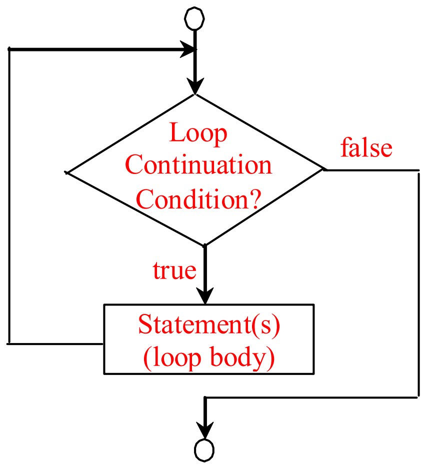
Objectives

- To write programs that execute statements repeatedly using a **while** loop.
- To control a loop with the user confirmation.
- To obtain input from a file using input redirection rather than typing from the keyboard.
- To read data from and write to a file.
- To write loops using **do-while** statements.
- To write loops using **for** statements.
- To discover the similarities and differences of three types of loop statements.
- To write nested loops .
- To learn the techniques for minimizing numerical errors.
- To implement program control with **break** and **continue**.
- To write a program that tests palindromes .
- To write a program that displays prime numbers.

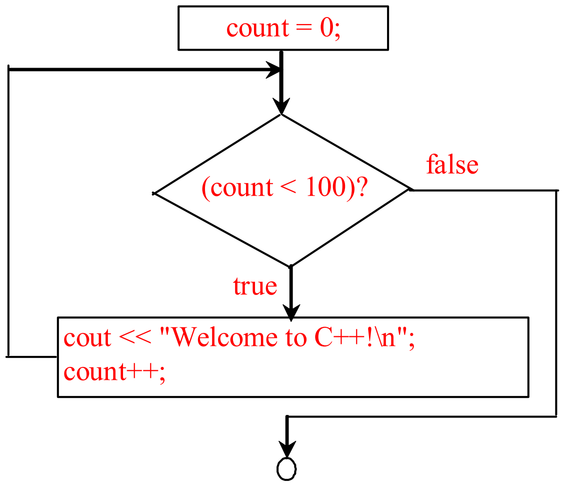
`while` Loop Flow Chart

```
while (loop-continuation-condition)
{
 // loop-body;
 Statement(s);
}
```

```
int count = 0;
while (count < 100)
{
  cout << "Welcome to C++!“<<endl;
  count++;
}
```



(a)



(b)

Case Study: Guessing Numbers

Write a program that randomly generates an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently. Here is a sample run:

GuessNumber

Controlling a Loop with User Confirmation

```
char continueLoop = 'Y';
while (continueLoop == 'Y')
{
  // Execute body once
  // Prompt the user for confirmation
  cout << "Enter Y to continue and N to quit: ";
  cin >> continueLoop;
}
```

Ending a Loop with a specific (Sentinel) Value

Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*.

Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

SentinelValue

Caution

Don’t use floating-point values for equality checking in a loop control. Since floating-point values are approximations, using them could result in imprecise counter values and inaccurate results. This example uses int value for data. If a floating-point type value is used for data, (data != 0) may be true even though data is 0.

```
double data = pow(sqrt(2.0), 2) - 2;
if (data == 0)
  cout << "data is zero";
else
   cout << "data is not zero";
```

Reading Data from a File

If you have many numbers to read, you will need to write a loop to read all these numbers even 0. If you don’t know how many numbers are in the file and want to read them all, how do you know the end of file? You can invoke the **eof()** function on the input object to detect it.

We write a program that reads all data from a file (scores.txt) and prints them out until eof() reached.

ReadAllData

Writing Data to a File

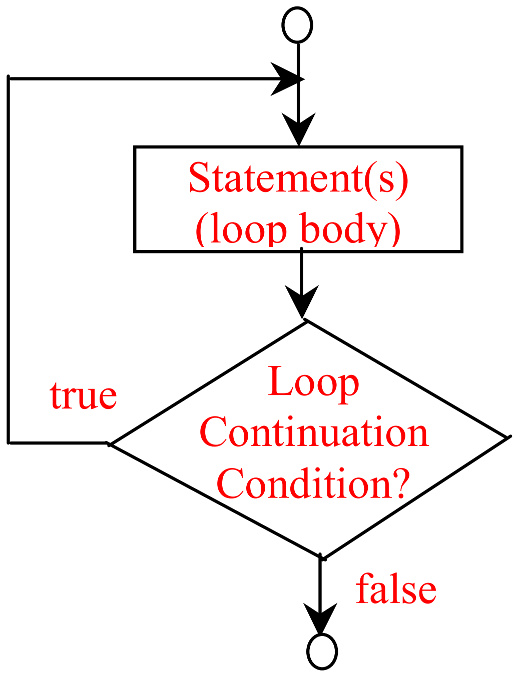
The example in previous slide reads some data from the a file. You may want to write the data you read from a stream into a file. We rewrite the previous program to write all data into a new file instead of standard out.

WriteAllData

do-while Loop

SentinelValue

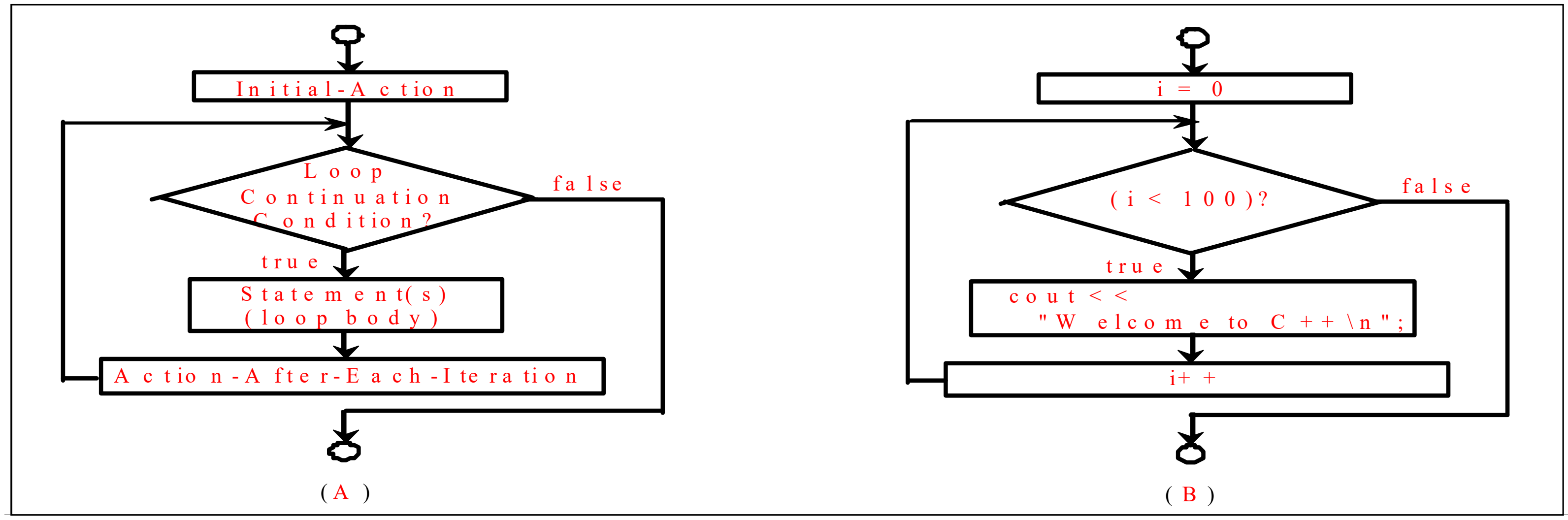
```
do
{
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```



for Loops

```
for (initial-action; loop-continuation-condition; action-after-each-iteration)
{
   // loop body;
  Statement(s);
}
```

```
int i;
for (i = 0; i < 100; i++)
{
  cout << "Welcome to C++!";
}
```


Initial-Action
Loop Continuation Condition?
false
true
Statement(s) (loop body)
Action-After-Each-Iteration
(A)
i = 0
(i < 100)?
false
true
cout << "Welcome to C++\n";
i++
(B)

for Loops

Declare i

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

Declare and initiate i

```
for (int i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

Note

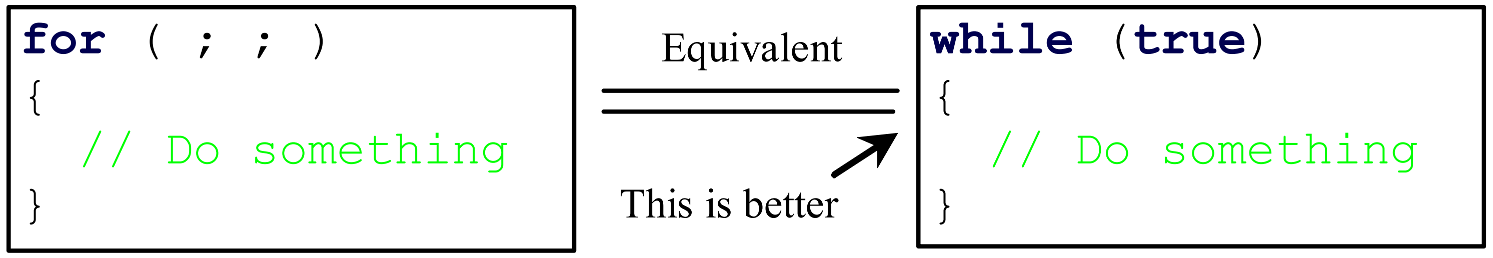
The initial-action in a for loop can be a list of zero or more comma-separated expressions. The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements. Therefore, the following two for loops are correct. They are rarely used in practice, however.

```
for ( int i = 1; i < 100; cout << (i++) );


for ( int i = 0, j = 0; (i + j < 10); i++, j++ ) {
  // Do something
}
```

Note

If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:



```
for ( ; ; )
{
  // Do something
}
```

(a)

```
while (true)
{
  // Do something
}
```

(b)

Example: Using for Loops

Problem: Write a program that sums a series that starts with 0.01 and ends with 1.0. The numbers in the series will increment by 0.01, as follows: 0.01 + 0.02 + 0.03 and so on.

TestSum

Nested Loops

Problem: Write a program that uses nested for loops to print a multiplication table.

TestMultiplicationTable

Using break and continue

Examples for using the break and continue keywords:

TestBreak

TestContinue

Exercise: Do it at home : Checking Palindromes

Problem: Write a program that tests whether a string is a palindrome.

A string is a palindrome if it reads the same forward and backward. The words “mom,” “dad,” and “noon,” for example, are all palindromes.

How do you write a program to check whether a string is a palindrome? One solution is to check whether the first character in the string is the same as the last character. If so, check whether the second character is the same as the second-last character. This process continues until a mismatch is found or all the characters in the string are checked, except for the middle character if the string has an odd number of characters.

Thank you