PROGRAMMING WITH C++

DTS102TC

JIANJIA WANG

JIANJIA.WANG@XJTLU.EDU.CN

SCHOOL OF AI AND ADVANCED COMPUTING

Xi'an Jiaotong-Liverpool University

西交利物浦大学

CHAPTER 5

LOOPS

Xi'an Jiaotong-Liverpool University

西交利物浦大學

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?

OPENING PROBLEM

Problem:

100 times

```
cout << "Welcome to Java!" << endl;
cout << "Welcome to Java!" << endl;
cout << "Welcome to Java!" << endl;
cout << "Welcome to Java!" << endl;
cout << "Welcome to Java!" << endl;

…
…
…
cout << "Welcome to Java!" << endl;
cout << "Welcome to Java!" << endl;
cout << "Welcome to Java!" << endl;
cout << "Welcome to Java!" << endl;
cout << "Welcome to Java!" << endl;
```

INTRODUCING WHILE LOOPS

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

OBJECTIVES

To write programs that execute statements repeatedly using a while loop (§5.2).

To follow the loop design strategy to develop loops (§§5.2.1–5.2.3).

To control a loop with the user confirmation (§5.2.4).

To control a loop with a sentinel value (§5.2.5).

To obtain input from a file using input redirection rather than typing from the keyboard (§5.2.6).

To read all data from a file (§5.2.7).

To write loops using do-while statements (§5.3).

To write loops using for statements (§5.4).

To discover the similarities and differences of three types of loop statements (§5.5).

To write nested loops (§5.6).

To learn the techniques for minimizing numerical errors (§5.7).

To learn loops from a variety of examples (GCD, FutureTuition, MonteCarloSimulation, Dec2Hex) (§5.8).

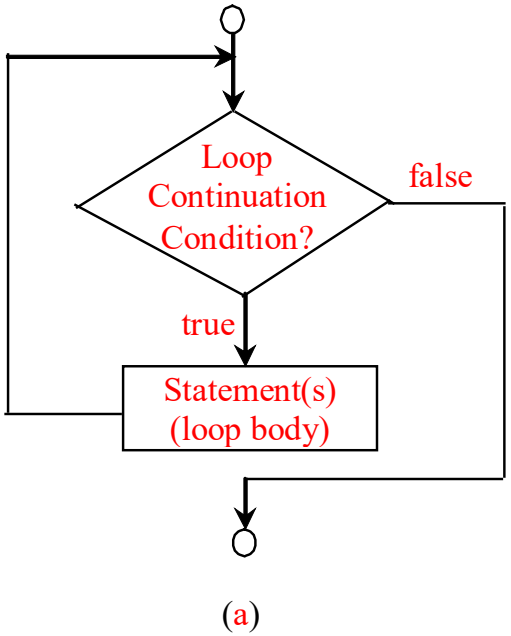
To implement program control with break and continue (§5.9).

To write a program that tests palindromes (§5.10).

To write a program that displays prime numbers (§5.11).

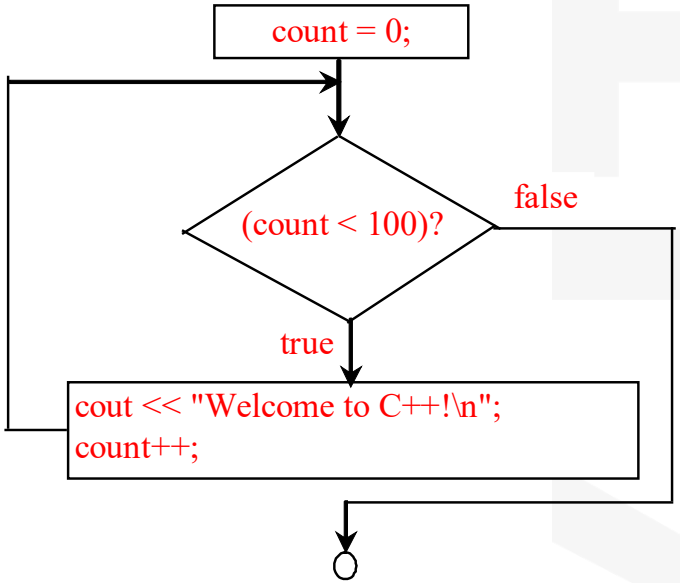
WHILE LOOP FLOW CHART

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



(a)

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



(b)

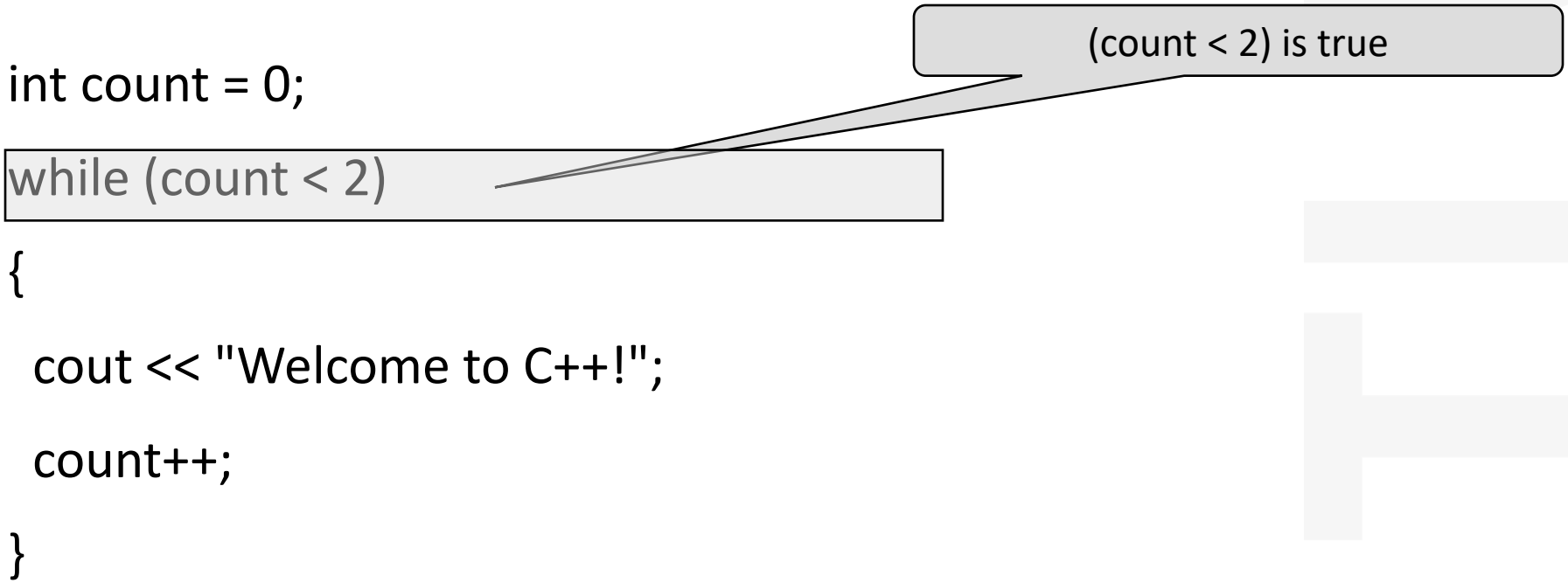
TRACE WHILE LOOP

```
int count = 0;
```

Initialize count

```
while (count < 2)
{
  cout << "Welcome to C++!";
  count++;
}
```

TRACE WHILE LOOP, CONT.

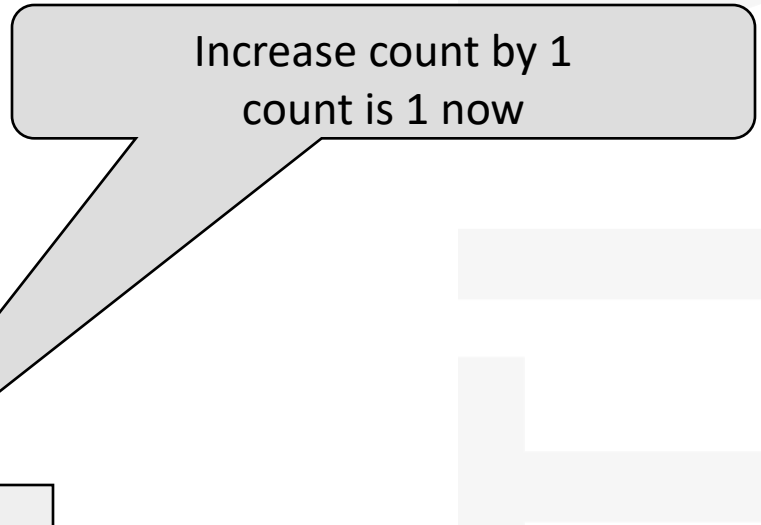


TRACE WHILE LOOP, CONT.


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

TRACE WHILE LOOP, CONT.

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



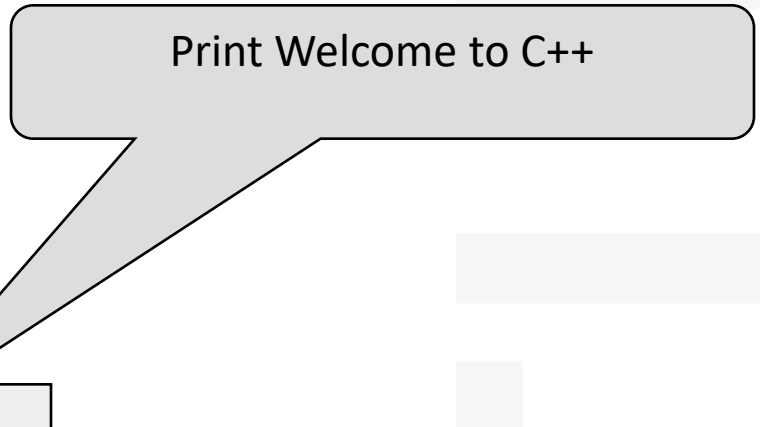
TRACE WHILE LOOP, CONT.

```
int count = 0;

while (count < 2)
{
  cout << "Welcome to C++!";
  count++;
}
```

(count < 2) is still true since count is 1

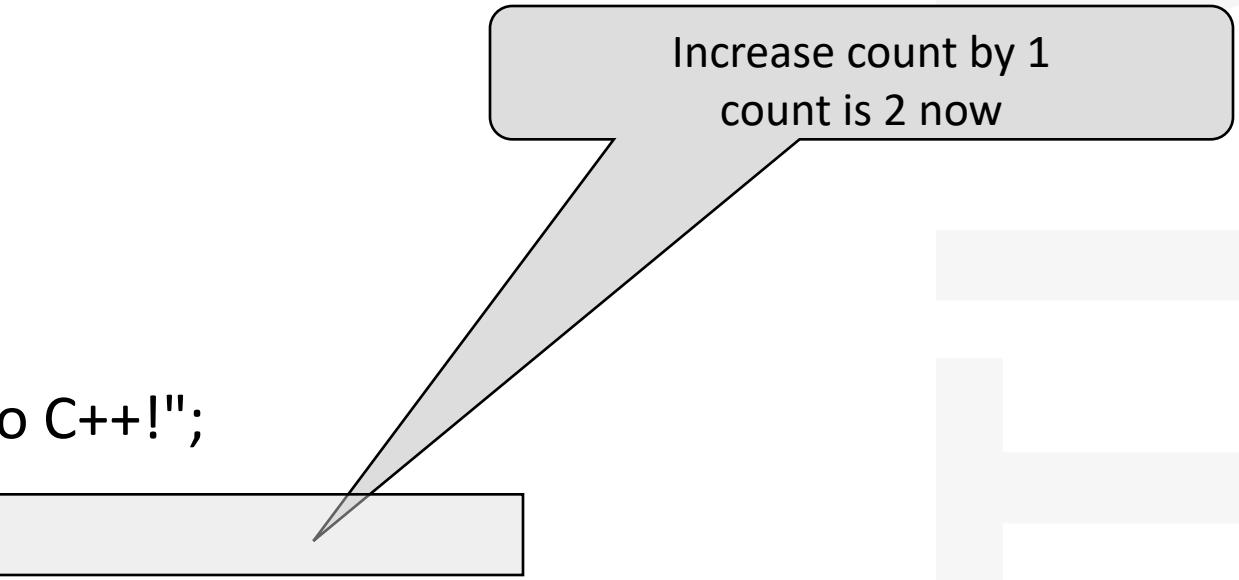
TRACE WHILE LOOP, CONT.

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



TRACE WHILE LOOP, CONT.

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



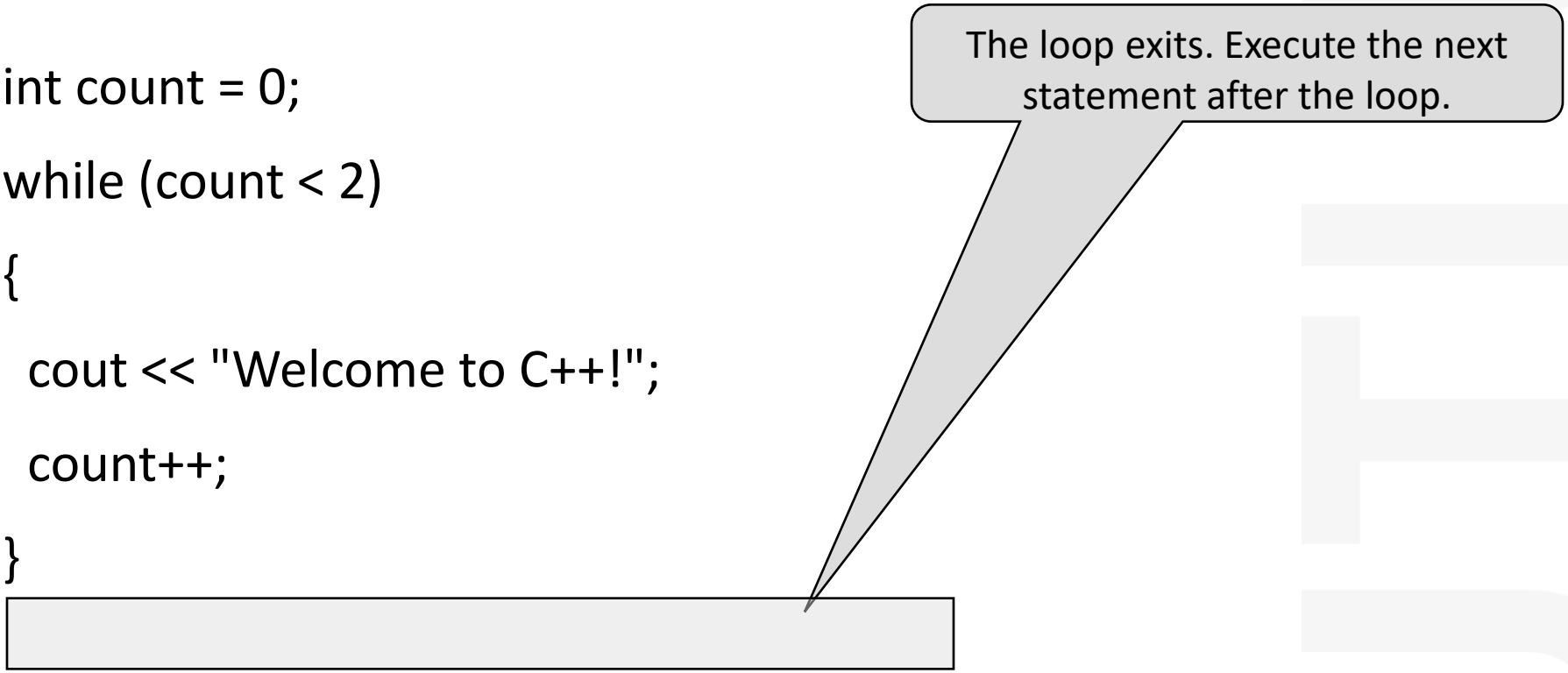
TRACE WHILE LOOP, CONT.

```
int count = 0;

while (count < 2)

{

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

  count++;

}
```

(count < 2) is false since count is 2 now

TRACE WHILE LOOP

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



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:

```
#include <iostream>
#include <cstdlib>
#include <ctime> // Needed for the time function
using namespace std;
int main()
{
  // Generate a random number to be guessed
  srand(time(0));
  int number = rand() % 101;
  cout << "Guess a magic number between 0 and 100";
  // Prompt the user to guess the number
  cout << "\nEnter your guess: ";
  int guess;
  cin >> guess;
  if (guess == number)
    cout << "Yes, the number is " << number << endl;
  else if (guess > number)
    cout << "Your guess is too high" << endl;
  else
    cout << "Your guess is too low" << endl;
  return 0;
}
```

```
#include <iostream>
#include <cstdlib>
#include <ctime> // Needed for the time function
using namespace std;
int main() {
  // Generate a random number to be guessed
  srand(time(0));
  int number = rand() % 101;
  cout << "Guess a magic number between 0 and 100";
  int guess = -1;
  while (guess != number)
  {
    // Prompt the user to guess the number
    cout << "\nEnter your guess: ";
    cin >> guess;
    if (guess == number)
      cout << "Yes, the number is " << number << endl;
    else if (guess > number)
      cout << "Your guess is too high" << endl;
    else
      cout << "Your guess is too low" << endl;
  } // End of loop
  return 0; }
```

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 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.

```
#include <iostream>
using namespace std;
int main()
{
  cout << "Enter an integer (the input ends " <<
    "if it is 0): ";
  int data;
  cin >> data;
  // Keep reading data until the input is 0
  int sum = 0;
  while (data != 0)
  {
    sum += data;
    // Read the next data
    cout << "Enter an integer (the input ends " <<
      "if it is 0): ";
    cin >> data;
  }
  cout << "The sum is " << sum << endl;
  return 0;
}
```

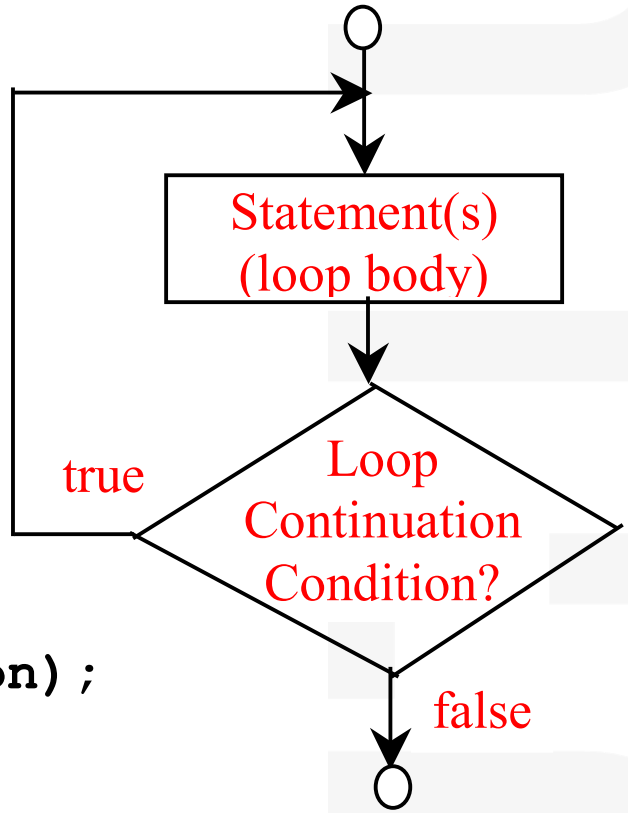
CAUTION

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";
```

DO-WHILE LOOP

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

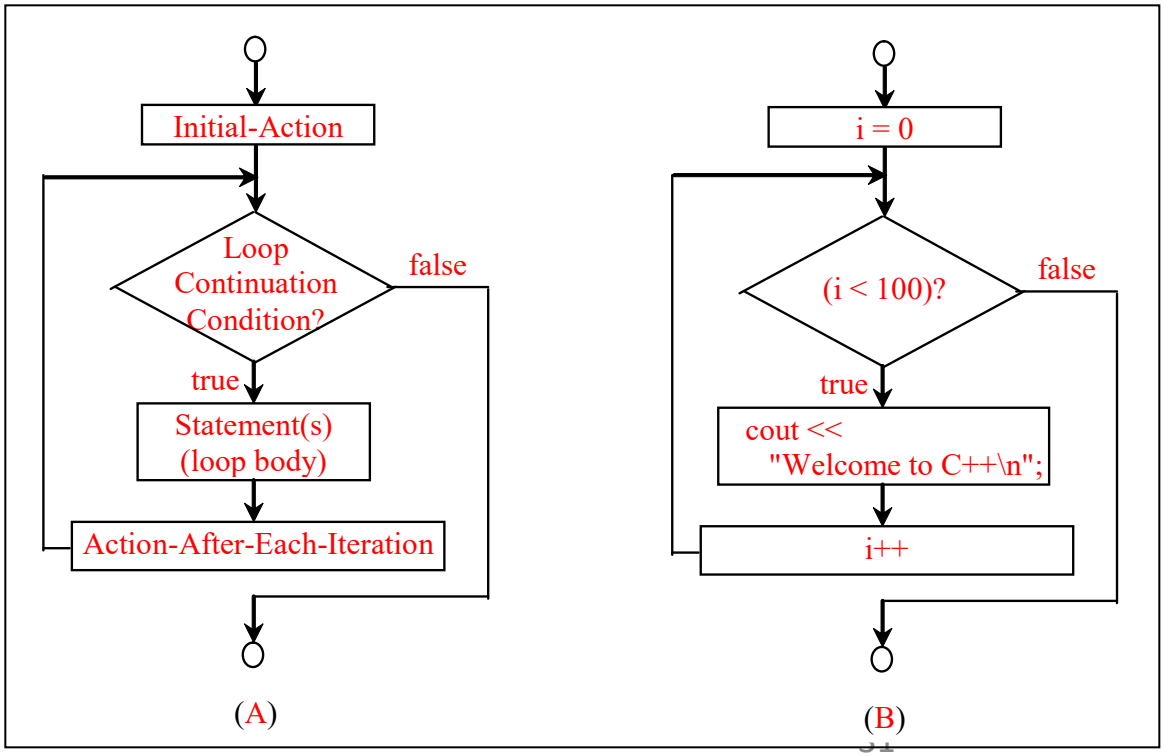


```
#include <iostream>
using namespace std;

int main()
{
  // Initialize data and sum
  int data = 0;
  int sum = 0;

  do
  {
    sum += data;

    // Read the next data
    cout << "Enter an integer (the input ends " <<
      "if it is 0): ";
    cin >> data; // Keep reading data until the input is 0
  }
  while (data != 0);

  cout << "The sum is " << sum << endl;

  return 0;
}
```

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++!\n";
}
```



TRACE FOR LOOP

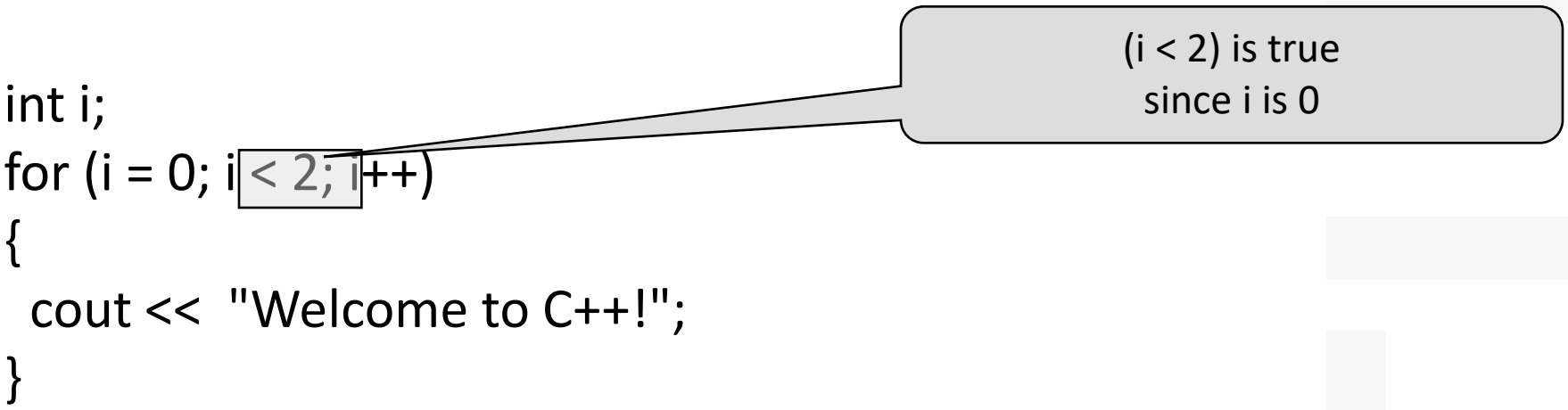

```
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

TRACE FOR LOOP, CONT.

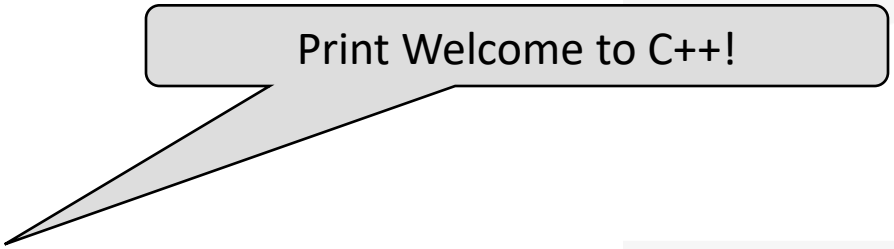
```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

Execute initializer
i is now 0

TRACE FOR LOOP, CONT.



TRACE FOR LOOP, CONT.




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

TRACE FOR LOOP, CONT.

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

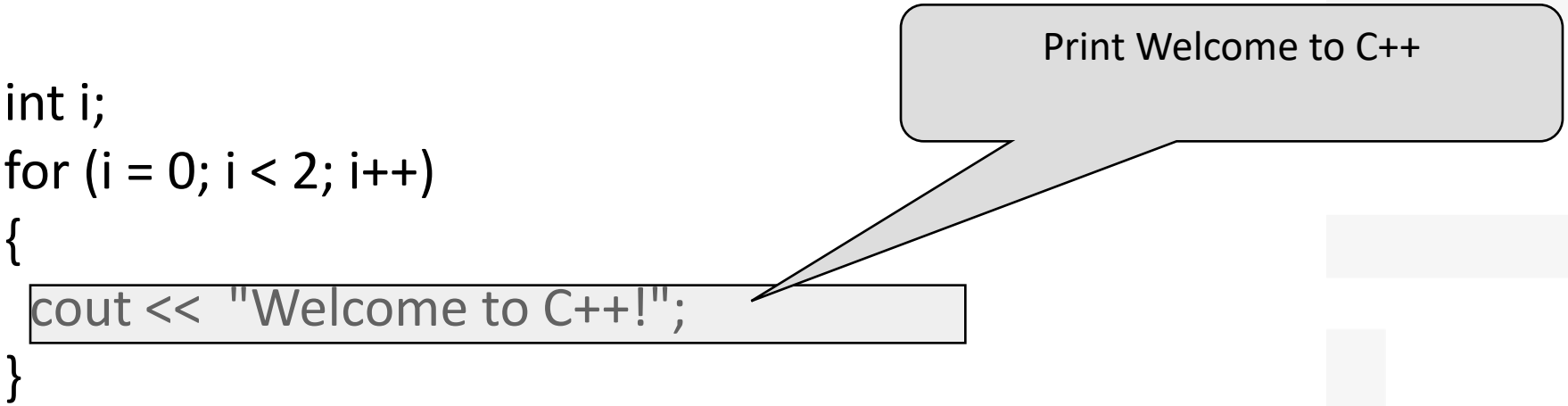
Execute adjustment statement
i now is 1

TRACE FOR LOOP, CONT.

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

(i < 2) is still true
since i is 1

TRACE FOR LOOP, CONT.



TRACE FOR LOOP, CONT.

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

Execute adjustment statement
i now is 2

TRACE FOR LOOP, CONT.

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

(i < 2) is false
since i is 2

TRACE FOR LOOP, CONT.

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

Exit the loop. Execute the next statement after the loop

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)

Equivalent

This is better

```
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.

```
#include <iostream>
using namespace std;

int main()
{
  // Initialize sum
  double sum = 0;

  // Add 0.01, 0.02, ..., 0.99, 1 to sum
  for (double i = 0.01; i <= 1.0; i = i + 0.01)
    sum += i;

  // Display result
  cout << "The sum is " << sum << endl;

  return 0;
}
```

WHICH LOOP TO USE?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):

```
while (loop-continuation-condition)
{
  // Loop body
}
```
(a)

Equivalent

```
for ( ; loop-continuation-condition; )
{
  // Loop body
}
```
(b)

A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases (see Review Question 3.19 for one of them):

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

Equivalent

```
initial-action;
while (loop-continuation-condition)
{
  // Loop body;
  action-after-each-iteration;
}
```
(b)

RECOMMENDATIONS

Use the one that is most intuitive and comfortable for you.

In general, a for loop may be used if the number of repetitions is counter-controlled, as, for example, when you need to print a message 100 times.

A while loop may be used if the number of repetitions is sentinel-controlled, as in the case of reading the numbers until the input is 0.

A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

NESTED LOOPS

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

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(){
 cout << "      Multiplication Table\n";
 // Display the number title
 cout << "  | ";
 for (int j = 1; j <= 9; j++)
  cout << setw(3) << j;
 cout << "\n";
 cout << "------------------------------\n";
 // Display table body
 for (int i = 1; i <= 9; i++)
 {   cout << i << " | ";
  for (int j = 1; j <= 9; j++)
  {     // Display the product and align properly
    cout << setw(3) << i * j;    }
  cout << "\n";  }
 return 0;
}
```

CASE STUDY: FINDING THE GREATEST COMMON DIVISOR

Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor.

Solution:

Suppose you enter two integers 4 and 2, their greatest common divisor is 2.

Suppose you enter two integers 16 and 24, their greatest common divisor is 8.

So, how do you find the greatest common divisor? Let the two input integers be n1 and n2.

You know number 1 is a common divisor, but it may not be the greatest commons divisor.

So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.

```
#include <iostream>
using namespace std;
int main()
{
  // Prompt the user to enter two integers
  cout << "Enter first integer: ";
  int n1;
  cin >> n1;
  cout << "Enter second integer: ";
  int n2;
  cin >> n2;
  int gcd = 1;
  int k = 2;
  while (k <= n1 && k <= n2)
  {
    if (n1 % k == 0 && n2 % k == 0)
      gcd = k;
    k++;
  }
  cout << "The greatest common divisor for " << n1 << " and "
       << n2 << " is " << gcd << endl;

  return 0;
}
```

USING BREAK AND CONTINUE

TestBreak

```
#include <iostream>
using namespace std;
int main()
{
  int sum = 0;
  int number = 0;

  while (number < 20)
  {
    number++;
    sum += number;
    if (sum >= 100)
      break;
  }
  cout << "The number is " << number << endl;
  cout << "The sum is " << sum << endl;
  return 0;
}
```

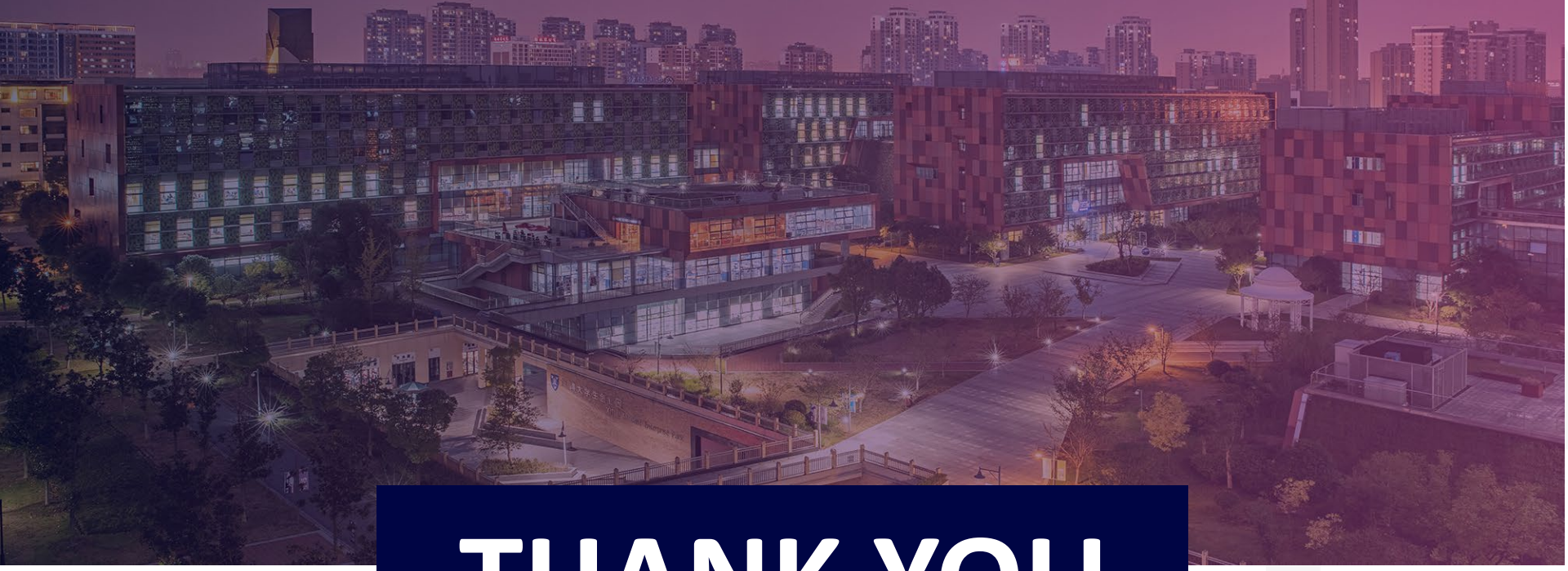
USING BREAK AND CONTINUE

TestContinue

```
#include <iostream>
using namespace std;
int main()
{
  int sum = 0;
  int number = 0;
  while (number < 20)
  {
    number++;
    if (number == 10 || number == 11)
      continue;
    sum += number;
  }
  cout << "The sum is " << sum << endl;
  return 0;
}
```

SUMMARIZATION

- While Loops
- IO Redirections
- Do While
- For Loops
- Break & Continue



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

Xi'an Jiaotong-Liverpool University
西交利物浦大学