Computer Science I

Chapter 5 - Loops and Files

Hanoi University of Science and Technology

November 19, 2025

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Six Edition, Pearson, 2009
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

The for Loop

- Useful for counter-controlled loop
- General Format:

```
for(initialization; test; update)
  statement; // or block in { }
```

- No semicolon after the update expression or after the)

The for Loop

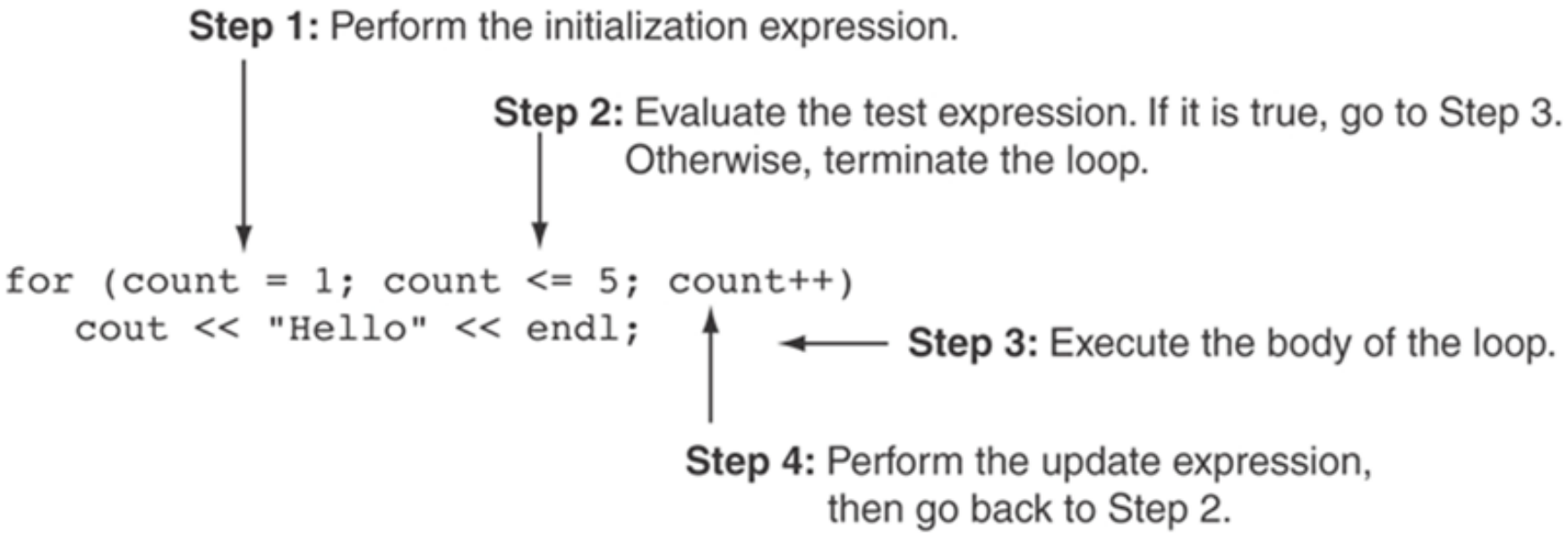
- `for(initialization; test; update)`
  `statement; // or block in { }`

1. Perform `initialization`
2. Evaluate `test` expression
   - If `true`, execute `statement`
   - If `false`, terminate loop execution
3. Execute `update`, then re-evaluate `test` expression

The for Loop

```
int count;
for (count = 1; count <= 5; count++)
  cout << "Hello" << endl;
```

A Closer Look at the Previous Example



A for Loop in Program 5-9

Program 5-9

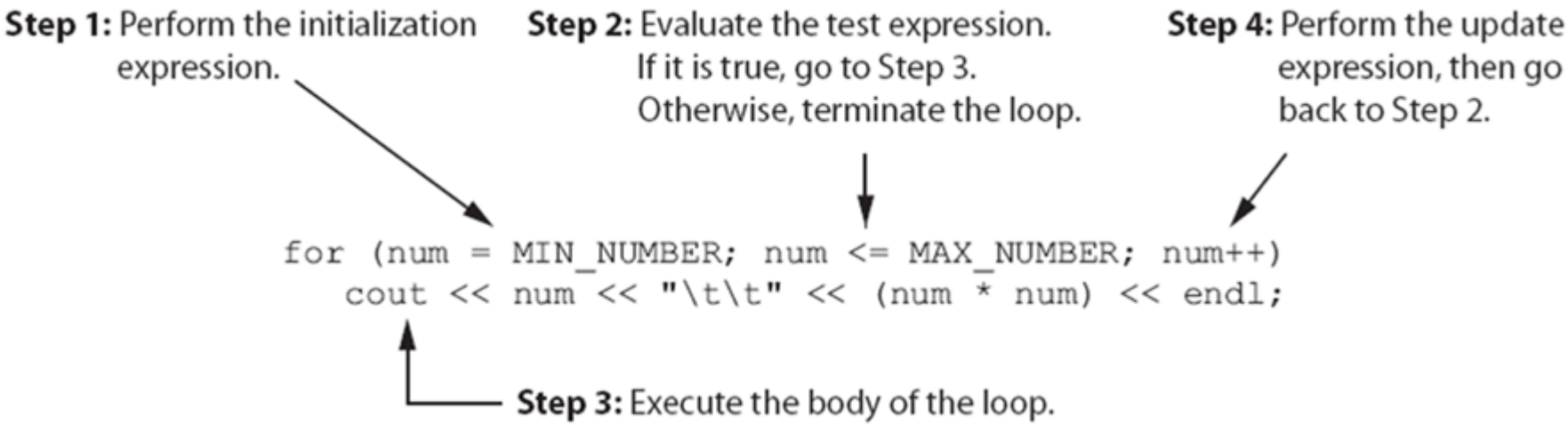
```
1  // This program displays the numbers 1 through 10 and
2  // their squares.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     const int MIN_NUMBER = 1,    // Starting value
9               MAX_NUMBER = 10;   // Ending value
10    int num;
11
12    cout << "Number Number Squared\n";
13    cout << "-------------------------\n";
14
15    for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
16       cout << num << "\t\t" << (num * num) << endl;
17
18    return 0;
19 }
```

A for Loop in Program 5-9

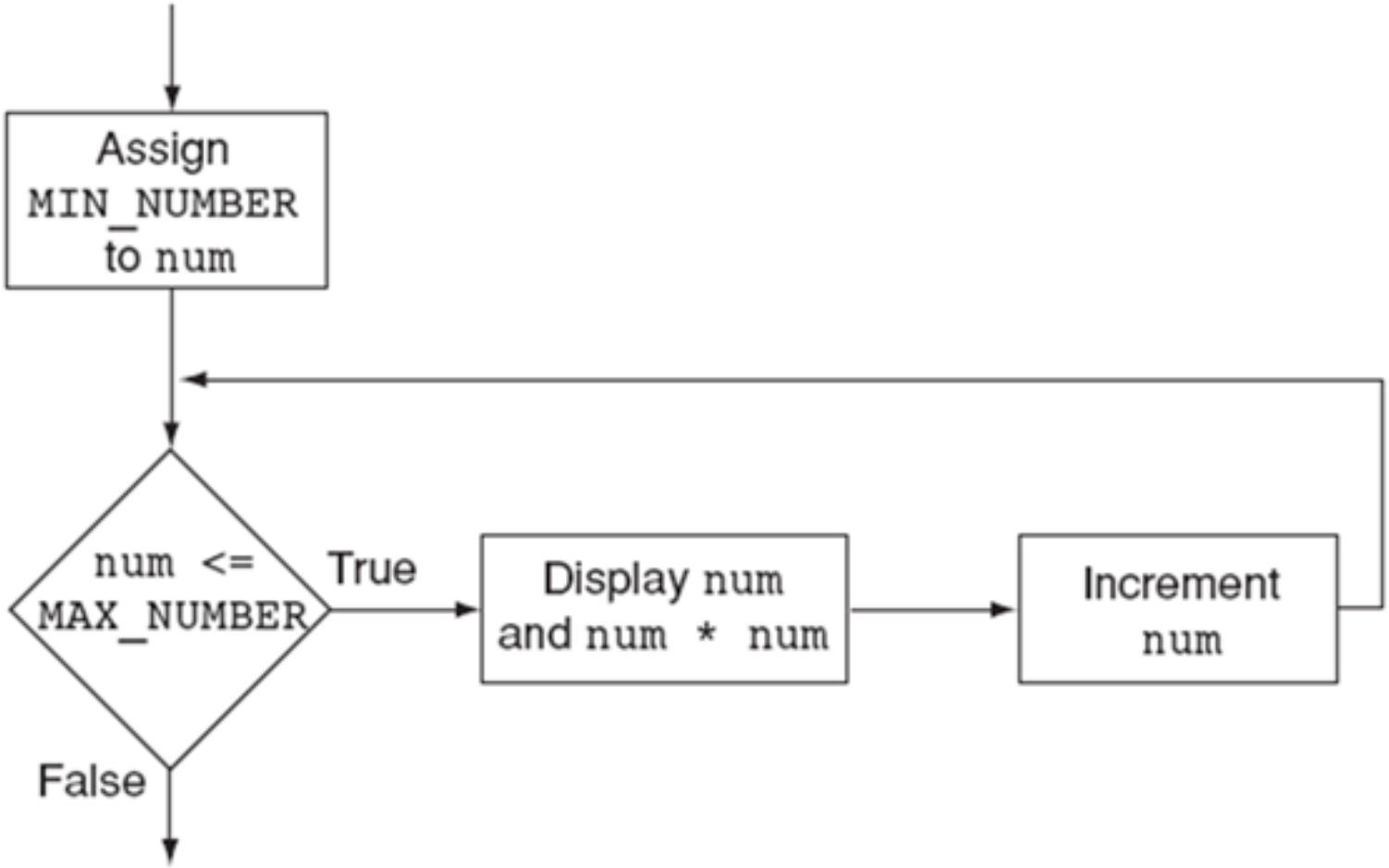
Program Output

```
Number Number Squared
---------------------
1              1
2              4
3              9
4              16
5              25
6              36
7              49
8              64
9              81
10             100
```

A Closer Look at Program 5-9



A Closer Look at Program 5-9



When to Use the `for` Loop

In any situation that clearly requires:

- An initialization
- A false condition to stop the loop
- An update to occur at the end of each iteration

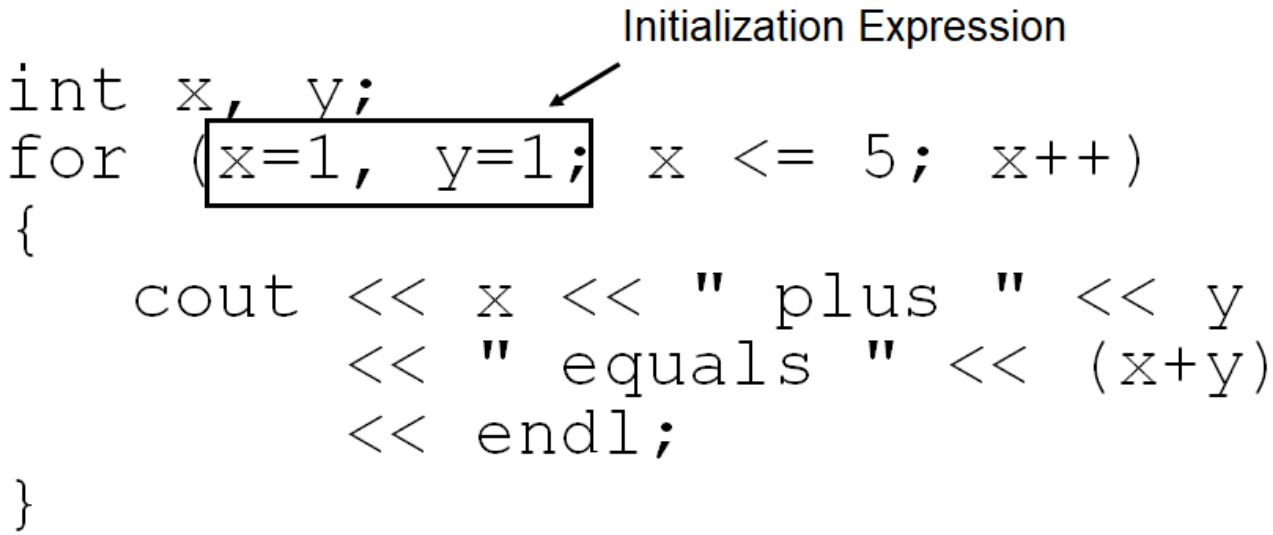
The `for` Loop is a Pretest Loop

- The for loop tests its test expression before each iteration, so it is a pretest loop.
- The following loop will never iterate:

```
for (count = 11; count <= 10; count++)
   cout << "Hello" << endl;
```

`for` Loop - Modifications

- You can have multiple statements in the `initialization` expression. Separate the statements with a comma:



`for` Loop - Modifications

- You can also have multiple statements in the `test/update` expression. Separate the statements with a comma:

Update Expression

```
int x, y;
for (x=1, y=1; x <= 5; x++, y++)
{
    cout << x << " plus " << y
         << " equals " << (x+y)
         << endl;
}
```

`for` Loop - Modifications

- You can omit the `initialization` expression if it has already been done:

```
int sum = 0, num = 1;
for (; num <= 10; num++)
  sum += num;
```

`for` Loop - Modifications

- You can declare variables in the `initialization` expression:

```
int sum = 0;
for (int num = 0; num <= 10; num++)
  sum += num;
```

- The scope of the variable num is the `for` loop.

Nested Loops

- A nested loop is a loop inside the body of another loop
- Inner (inside), outer (outside) loops:

```
for (row=1; row<=3; row++) //outer
  for (col=1; col<=3; col++)//inner
    cout << row * col << endl;
```

Nested Loops

```
26    // Determine each student's average score.
27    for (int student = 1; student <= numStudents; student++)
28    {
29       total = 0;       // Initialize the accumulator.
30       for (int test = 1; test <= numTests; test++)
31       {
32          double score;
33          cout << "Enter score " << test << " for ";
34          cout << "student " << student << ": ";
35          cin >> score;
36          total += score;
37       }
38       average = total / numTests;
39       cout << "The average score for student " << student;
40       cout << " is " << average << ".\n\n";
41    }
```

Inner Loop

Outer Loop

Nested Loops - Notes

- Inner loop goes through all repetitions for each repetition of outer loop
- Inner loop repetitions complete sooner than outer loop
- Total number of repetitions for inner loop is product of number of repetitions of the two loops.

Deciding Which Loop to Use

- The `while` loop is a conditional pretest loop:
  - Iterates as long as a certain condition exists
  - Validating input
  - Reading lists of data terminated by a sentinel
- The `do-while` loop is a conditional posttest loop
  - Always iterates at least once
  - Repeating a menu
- The `for` loop is a pretest loop
  - Built-in expressions for initializing, testing, and updating
  - Situations where the exact number of iterations is known

Breaking and Continuing a Loop

Breaking Out of a Loop:

- Can use `break` to terminate execution of a loop
- When used in an inner loop, terminates that loop only and goes back to outer loop

The `continue` Statement:

- Can use continue to go to end of loop and prepare for next repetition
  - `while, do-while` loops: go to test, repeat loop if test passes
  - `for` loop: perform update step, then test, then repeat loop if test passes

Using Files for Data Storage

- Can use files instead of keyboard, monitor screen for program input, output
- Allows data to be retained between program runs
- Steps:
  - Open the file
  - Use the file (read from, write to, or both)
  - Close the file

Files: What is Needed

- Use `fstream` header file for file access
- File stream types:
  - `ifstream` for input from a file
  - `ofstream` for output to a file
  - `fstream` for input from or output to a file
- Define file stream objects:

```
ifstream infile;
ofstream outfile;
```

Opening Files

- Create a link between file name (outside the program) and file stream object (inside the program)
- Use the `open` member function:

```
infile.open("inventory.dat");
outfile.open("report.txt");
```

- Filename may include drive, path info.
- Output file will be created if necessary; existing file will be erased first
- Input file must exist for `open` to work

Testing for File Open Errors

- Can test a file stream object to detect if an open operation failed:

```
infile.open("test.txt");
if (!infile)
{
  cout << "File open failure!";
}
```

Using Files

- Can use output file object and << to send data to a file:

```
outfile << "Inventory report";
```

- Can use input file object and >> to copy data from file to variables:

```
infile >> partNum;
infile >> qtyInStock >> qtyOnOrder;
```

Reading data from file

Program 5-19

```
1  // This program reads data from a file.
2  #include <iostream>
3  #include <fstream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9       ifstream inputFile;
10      string name;
11
12      inputFile.open("Friends.txt");
13      cout << "Reading data from the file.\n";
14
15      inputFile >> name;         // Read name 1 from the file
16      cout << name << endl;      // Display name 1
17
18      inputFile >> name;         // Read name 2 from the file
19      cout << name << endl;      // Display name 2
20
21      inputFile >> name;         // Read name 3 from the file
22      cout << name << endl;      // Display name 3
23
24      inputFile.close();         // Close the file
25      return 0;
26 }
```

Program Output

```
Reading data from the file.
Joe
Chris
Geri
```

Writing to file

Program 5-15

```
1  // This program writes data to a file.
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int main()
7  {
8       ofstream outputFile;
9       outputFile.open("demofile.txt");
10
11      cout << "Now writing data to the file.\n";
12
13      // Write four names to the file.
14      outputFile << "Bach\n";
15      outputFile << "Beethoven\n";
16      outputFile << "Mozart\n";
17      outputFile << "Schubert\n";
18
19      // Close the file
20      outputFile.close();
21      cout << "Done.\n";
22      return 0;
23 }
```

Program Screen Output

```
Now writing data to the file.
Done.
```

Using Loops to Process Files

- The stream extraction operator >> returns `true` when a value was successfully read, `false` otherwise
- Can be tested in a while loop to continue execution as long as values are read from the file:

```
// read data separated by space, tab, newline
while (inputFile >> myString)
...
// read data separated by newline
while (getline(inputFile, myString))
...
```

Closing Files

- Use the `close` member function:

```
infile.close();
outfile.close();
```

- Don't wait for operating system to close files at program end:
  - may be limit on number of open files
  - may be buffered output data waiting to send to file

Closing Files

- Use the `close` member function:

```
infile.close();
outfile.close();
```

- Don't wait for operating system to close files at program end:
  - may be limit on number of open files
  - may be buffered output data waiting to send to file

Exercises

- Programming Challenges: 17, 18, 20, 23, 24
- Write a program to read a text file and find the longest word it contains.
- Write a program to read all numbers from a file and display their sum.
- Write a program that asks users to enter a positive integer (n) and lists all prime numbers less than n.
- Write a program that asks users to enter a positive integer n and calculates the factorial of n (n!)
- Write a program that asks users to enter a positive integer n and a positive floating-point number x. Calculates the sum $S(n) = x + x^2 + x^3 + ... + x^n$