CSci 127: Introduction to Computer Science

hunter.cuny.edu/csci

Today's Topics

- C++ and g++
- Hello World!
  - Libraries
  - Basic Input/Output (I/O)
- Datatypes
- Variables
- Assignment Statements
- Operators
- Decision Making / Loops
- Loops
- Examples

C++

- What is it?
  - Powerful programming language that gives you incredible speed and low-level access to memory and hardware!
- I heard about this language called C?
  - C is one of the most popular programming languages of all time. It can be used on almost every platform.
  - C++ was created as an extension to the C programming language.
  - C++ is like *C with Classes*.
    - Classes are a high-level programming construct that allows a program to be an interaction of objects.

C++ Continued

- C++ is a **compiled** not interpreted language.
  - **Compiler**: a program that understands a language and is used to create executable programs.
    - Translates all of the code once and allows the program to be executed many times thereafter.
  - **Interpreter**: a program that understands a language and is used to create programs.
    - Translates and executes the program line by line for each execution.
- Some examples:
  - Compiled languages: C/C++, Java, C#
  - Interpreted Languages: Python, Matlab, PHP
- Ok, so C++ is a compiled language…
  - But which compiler are we using?

g++

- What is it?
  - Compiler developed for the GNU operating system.
    - GNU: "**G**NU's **N**ot **U**nix"
    - Supports C, C++, Objective-C, and more.
- How do we get it?
    - Cygwin: linux-like environment for windows.
    - Xcode: Mac programming package found on the OSX disc.
    - (Refer to lab12 for assistance)

Getting Started

- C++ programs are stored in .cpp files.
  - Example: homework1**.cpp**, main**.cpp**, etc.
- Create the programs in Notepad (or Notepad++).
  - Mac users: you can use TextEdit.
- Compiling programs using g++:
  - Make sure the file is saved!
  - g++ homework1.cpp
    - Creates an executable (by default) called:
      - a.exe on Windows
      - a.out on Linux
    - Run the executable program with: **./**a.exe

Hello World!

- File: helloworld.cpp

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

int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Hello World Dissected

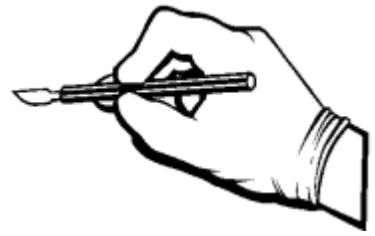
```
#include <iostream>
```

- `iostream` is a *library*.
  - **Library**: a collection of code that provides functionality to our programs.
- `iostream` provides us with basic input and output.
  - i.e., the ability to print and retrieve information to and from the console.
- We add this functionality to our program by referring to the library using the `#include` directive.

```
using namespace std;
```

- `std` is the standard C++ namespace.
- We're skipping this!
- Just include it in each of your programs!

Hello World Dissected



```
int main()
{}
```

- The main *function* of our program! Every C++ program needs this!
  - **Function**: grouping of code to solve a particular task.
- The `int` says that this function called main returns some integer.

```
cout << "Hello World!" << endl;
```

- `cout` is a C++ keyword that specifies that we should output to the standard console stream.
- The pair of arrow brackets `<<` is a C++ operator that controls the reading (`>>`) or writing (`<<`).
- Endl is a keyword that behaves like a carriage return (newline).

```
return 0;
```

- The value 0 gets returned to signal the end of the program.

cout

- What is it?
  - A reserved keyword in C++ that allows us to output text, variables, and expressions to the screen/console.
- How do we use it?
  - Include the code that allows us to print to the screen.

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

  - Use the << operator!
- Examples:

```
cout << "Know anything besides Hello World?";
cout << "We get the point already!";
```

More `cout` Stuff

- Breaking to a **new line**:
  - 2 ways
    - `\n` escape sequence
    - `endl` keyword
  - Examples:

```
cout << "Wow, a new line! \n";
cout << "Wow another new line!" << endl;
```

Comments

- This is one of the most important topics!
  - And also the simplest!
- What is it?
  - A way to annotate code that is ignored by the compiler.
- Two types of comments:
  - Single line: `//This is a single-line comment!`
    - Everything (on the same line) after the `//` is ignored!
  - Multi-Line: Triggered by the `/*` and `*/` sequences.

```
/* <- This opens the comment block…
     Dude, there‟s just too much to say!
This closes the comment block -> */
```

  - Rule: Use comments often to describe what your code is doing.

Datatypes

- C++ allows you to store and manipulate different types of data.
- Examples:

Type Name	Memory Used
`int`	4 bytes
`float`	4 bytes
`char`	1 byte
`bool`	1 byte

```
int number = 10001;
char middleInitial = 'E';   //Single Quotes
bool amIAwesome = true;
float teachingSalary = 0.50; //Cents!
```

- So we know the types of data that we can play around with. But how do we hold data in our program?

Escape Sequences

- Triggered by a backslash \
- The most common escape sequences are:

Sequence	Meaning
\n	Break the output to a new line. This is also achieved with the `endl` keyword!
\\	Allows you to put a backslash in the output.
\"	Allows you to put a double quote in the output.

- Examples:

```
cout << "This is awesome!\n";
cout << "A\\S\\L" << endl;
cout << "He said \"FAIL\"\n";
```

Variables

- What are they?
  - A facility for storing and manipulating data in a computer program.
  - Examples:
    `temp, windSpeed, position, weight, height, eyeColor`
  - Basic Rules:
    - **Must be declared** prior to use!
    - Case sensitive: `num` is not the same as `Num`
    - Don't start with a number! `1stTemp` is illegal!
    - No spaces! `wind Speed` is illegal!
    - No *reserved words*!
      - **Reserved word**: variable name with a predefined meaning in C++.

Statements

- What is it?
  - A line of code that indicates an action or performs some purpose.
- Examples:

```
num = 4;   x = y;   windspeed = 0.55;
```

- You might have noticed this semicolon business!
  - **Rule**: Statements in C++ are always terminated by a semicolon!
  - What does it do?
    - The semicolon indicates where the end of the statement lies.
    - Without the semicolon, we don't know where one statement begins and another one ends.

- Semicolons ruin lives!
- Let's see the most common type of statement!

Assignment Statements

i = i + 1

- Syntax:

```
datatype variable = expression;
```

- Examples:

```
float rate = 0.05;
float time = 1.0;
float distance = rate * time;
int count = count + 2;
```

- How does it work?
  - Expression on the right of the equal sign gets evaluated.
  - Then the result is stored in the variable on the left of the equal sign.

Assignment Compatibility

- Variables need to have a datatype when declared!
- Rule: Store the proper type of data in that variable!
- Example:

```
int number = 0.665;  //What gets stored in num?
//The number 0 gets stored in num!
//An integer is a whole number (positive or negative).
//We"re basically truncating the fractional part!
//Solution: Declare the variable as a float!
```

- Some datatypes can be automatically converted to another form.
  - Integers to floats: 1 to 1.0
  - Integers to bools: 1 to true; 0 to false
  - Chars to integers: 'a' to 97

Operators

- Arithmetic: Order of operations! PEMDAS!
  - Addition: `z = x + y;`
  - Subtraction: `z = x - y;`
  - Multiplication: `z = x * y;`
  - Division: `z = x / y;`
- Two types of division:
  - Integer (whole number): 1/2 = 0, 2/1 = 2, etc.
  - Floating point: 1.0/2 = 0.5, 1.0/3 = 0.3333, etc.
  - **Rule**: If both numbers are integers, then integer division is triggered. If either of the number is a floating point, then floating point division occurs!
- Integer division ruins lives!

More Operators

- Similar to division, we have the **modulus** operator!
  - The modulus operator `%` gives the remainder of the division.

```
3 % 4=  3;     //  3/4 = 0 so R = 3

4 % 1=  0;     //  4/1 = 4 so R = 0

2 % 2=  0;     //  2/2 = 1 so R = 0
```

- C++ has a bunch of nice math operators, but no exponentiation operator!
  - We can't easily express x^2!
  - To solve this, we can use a function known as `pow()`.

```
float xSquared = pow(x, 2); // x to the power of 2
```

  - We'll see function usage later…

Combined Operators

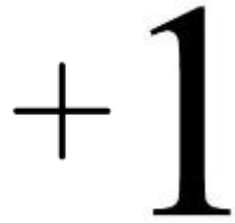
- We know how to do simple arithmetic operations.
- We can combine operations acting on a variable:
  - If we have these types of statements

```
x  =  x  +  10;

y  =  y  -  5;

x  =  x  /  2;

y  =  y  *  2;
```

  - We can rewrite these statements using combined operators:

```
x += 10
y -= 5
etc..
```

Increment Operator



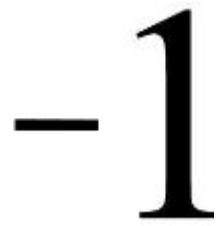
- What it is?
  - A concise way of adding one to a variable.
- Why do we use it?

– Two ways to basically say `x = x + 1`

```
y = x++;          //Increment x after  storing
y = ++x;          //Increment x first  then
                                       store
```

- Who cares?
  - Used a lot!
    - We'll see specific uses (looping) later on.
- Understand the difference between the two ways!

Decrement Operator



- What is it?
  - A cool way of subtracting one from a variable.
- Why do we use it?

– Two ways to basically say `x = x - 1`

```
y = x--;        //Decrement x  after storing

y = --x;        //Decrement x  first then
                               store
```

- Who cares?
  - It's the same argument as increment!
- Some languages do not have increment and decrement operators, and they get used quite often!

Type Casting

- We spoke about the automatic changing (**casting**) of data types:
  - `float` to `int`, `int` to `float`, `char` to `int`, etc…
- We can explicitly cast variables to different types:

```
float x = 5.34;
int y = (int) x;
```

Constants

- What are they?
  - Variables that can't change value!
- Why do we use them?
  - Sometimes you want to make sure that a variable retains the same value.
    - We'll get a compiler error if we try to change a constant!
- Syntax:
  - *`const datatype variable = Constant;`*
  - *Note*: We usually capitalize the variable name!
- Examples:
  - `const float PI = 3.14159265;`
  - `const float NYC_TAX = 8.875;`

cin

- What is it?
  - Another reserved keyword (part of `iostream` and the `std` namespace) in C++ that allows us to grab user input from the screen/console.
- When do we use it?
  - Basically, whenever we expect some input from the user.
  - Note, the opposite direction of the arrow operator >>.
- Examples:

```
int age;
cout << "Please enter your age: ";
cin >> age;
```

- Cool Stuff
  - `cin` take input in the type specified by the holder variable.
  - Example, in `cin >> age`, the input is automatically captured as an `int`!

DECISION MAKING

- Relational Operators
- `if` Statement
- `if/else` Statement

Relational Operators

- In C++ we have statements that evaluate to a boolean (truth) value. Known as *conditions*.
  - Statements that test for:
    - Equality `A == B //is A equal to B?`
    - Not Equal `A != B`
    - Greater Than `A > B`
    - Greater or Equal `A >= B`
    - Less Than `A < B`
    - Less or Equal `A <= B`
- Example:

```
int A = 4, B = 5;
bool isEqual = A == B; //Remember double equal signs!
bool isGreater = A > B;
```

Boolean Expressions

- We can combine relational statements into complicated expressions.
  - Boolean operators!
    - AND = &&
    - OR = ||
    - NOT = !
- Format:

  (*Boolean_Exp_1*) **&&** (*Boolean_Exp_2*)

  (*Boolean_Exp_1*) **||** (*Boolean_Exp_2*)

  (**!***Boolean_Exp_1*)

AND

- What is it?
  - A boolean operation that evaluates to true only when all parts of the expression are true.
- Format:

  (*Boolean_Exp_1*) && (*Boolean_Exp_2*)
- Example:

```
int timeFree;
cin >> timeFree;
if(timeFree >= 9 && timeFree <= 12)
   cout << "Dude, you're a bum!\n";
```

AND if it helps…

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

- Look at this *truth table* for `&&`.
- Both A and B are boolean expressions.
- Imagine that 0 is `false` and 1 is `true`.
- How to make sense of the table:
  - Line 1: If A is false and B is false, then A && B is false.
  - Line 2: If A is false and B is true, then A && B is false.
  - Line 3: If A is true and B is false, then A && B is false.
  - Line 4: If A is true and B is true, then A && B is true.

OR

- What is it?
  - A boolean operation that evaluates to true when **any** part of the expression is true!
- Format:

  (*Boolean_Exp_1*) || (*Boolean_Exp_2*)
- Example:

```
if(hoursHome >= 10 || status == “Hacker”)
   cout << “Get Outside!\n”;
```

OR if it helps…

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

- Look at this *truth table* for `||`.
- Both A and B are boolean expressions.
- Imagine that 0 is `false` and 1 is `true`.
- How to make sense of the table:
  - Line 1: If A is false and B is false, then A || B is false.
  - Line 2: If A is false and B is true, then A || B is true.
  - Line 3: If A is true and B is false, then A || B is true.
  - Line 4: If A is true and B is true, then A || B is true.

NOT

A	!A
0	1
1	0

- What is it?
  - A boolean expression that evaluates to `true` only when the expression is `false`.
  - It's the opposite of the expression's truth value.
- How do we use it?
  - `!Boolean_Exp`
- Examples:

```
if(!(status == "PhD Student"))
    cout << "You've got more money than Joel!\n";
```

`if` Statement

- What is it?
  - A branching construct whereby a set of code is executed based on the truth value of a condition.
- Syntax:

```
if (condition == true)
{
    //Do the code in here!
}
```

- What does it mean?
  - If the boolean condition/expression is true, then we execute the instructions within the curly braces.
    - The branching condition can be any statement that evaluates to a boolean value.
  - Otherwise, we skip all of the code between the curly braces.
- Example:

```
int x = 15;
if(x < 20)
   cout << "Yay!\n";
```

`if…else` Statement

- We need to understand how the code gets executed.
  - Each if statement gets visited regardless of whether or not its body gets executed…
  - But what if we have 20 consecutive if statements in our program?
    - If the first condition evaluated to true, then we would still check the conditions for the other 19…
  - Isn't there a way to skip all other conditions once we have a true condition?
- If/Else Statements to the rescue!

if…else continued!

- Easy Example:

```
if(x == y)
{
    cout << x << " is equal to " << y << endl;
}
else
{
    cout << x << " is not equal to " << y << endl;
}
```

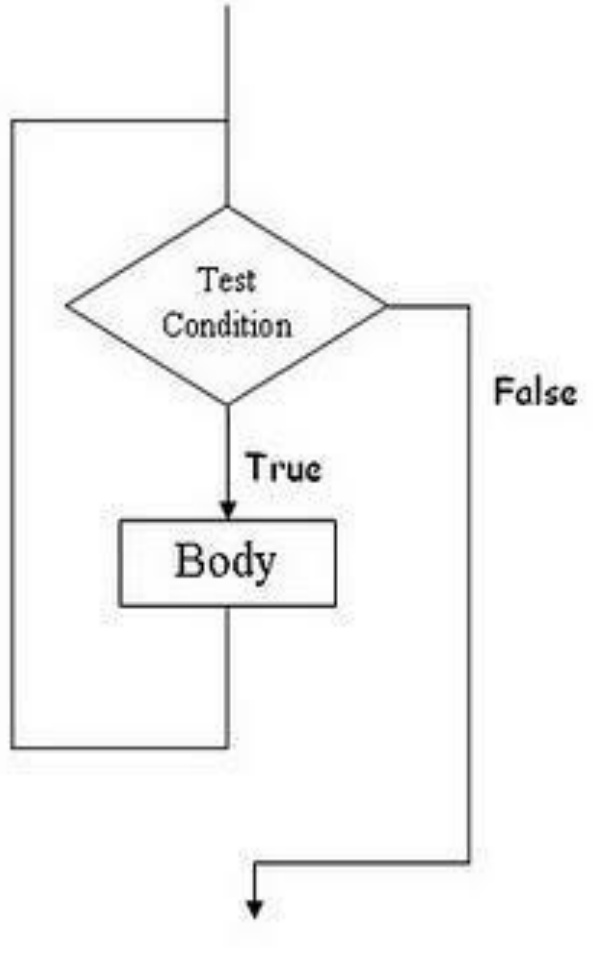
- More involved example: Combining Else and If

```
if(x == y)
{
    cout << x << " is equal to " << y << endl;
}
else if (x != y)
{
    cout << x << " is not equal to " << y << endl;
}
```

LOOPS

Loop

- **Loop**: a coding construct that allows for some code to be run repeatedly.
- A loop, like a branch:
  - Also has a condition (**loop condition**).
  - Can execute a chunk of code (the **loop body**) if the condition is true.



What does this look like in C++?

The `for` Loop

Useful for counter-controlled loop

General Format:

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

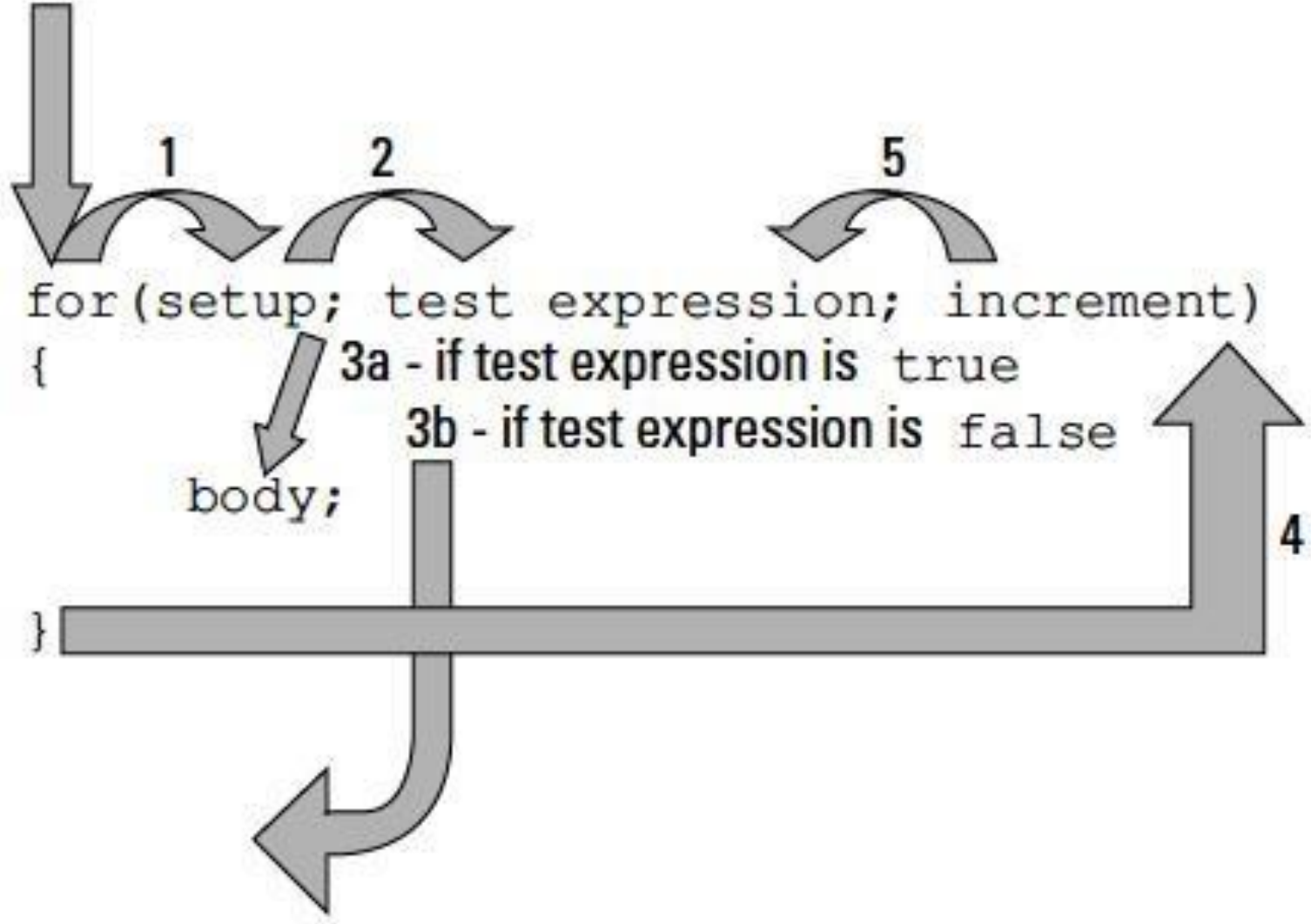
No semicolon after the `update` expression or after the `)`

`for` Loop - Mechanics

```
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

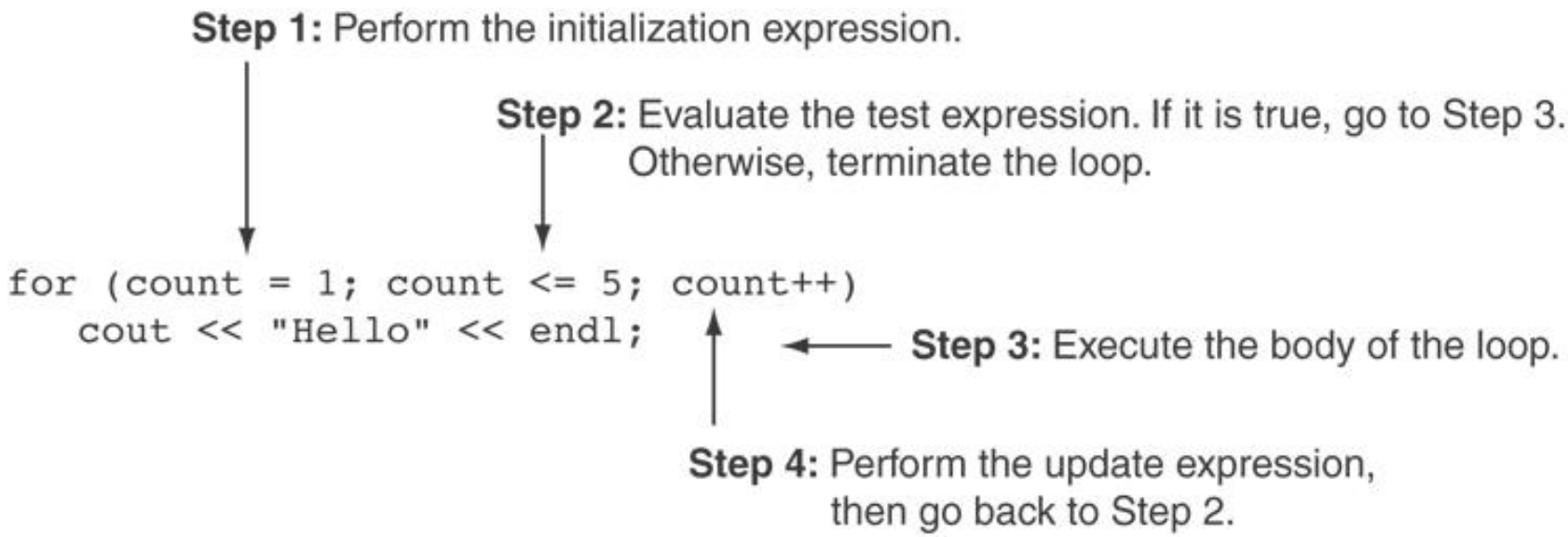
`for` loop parts!



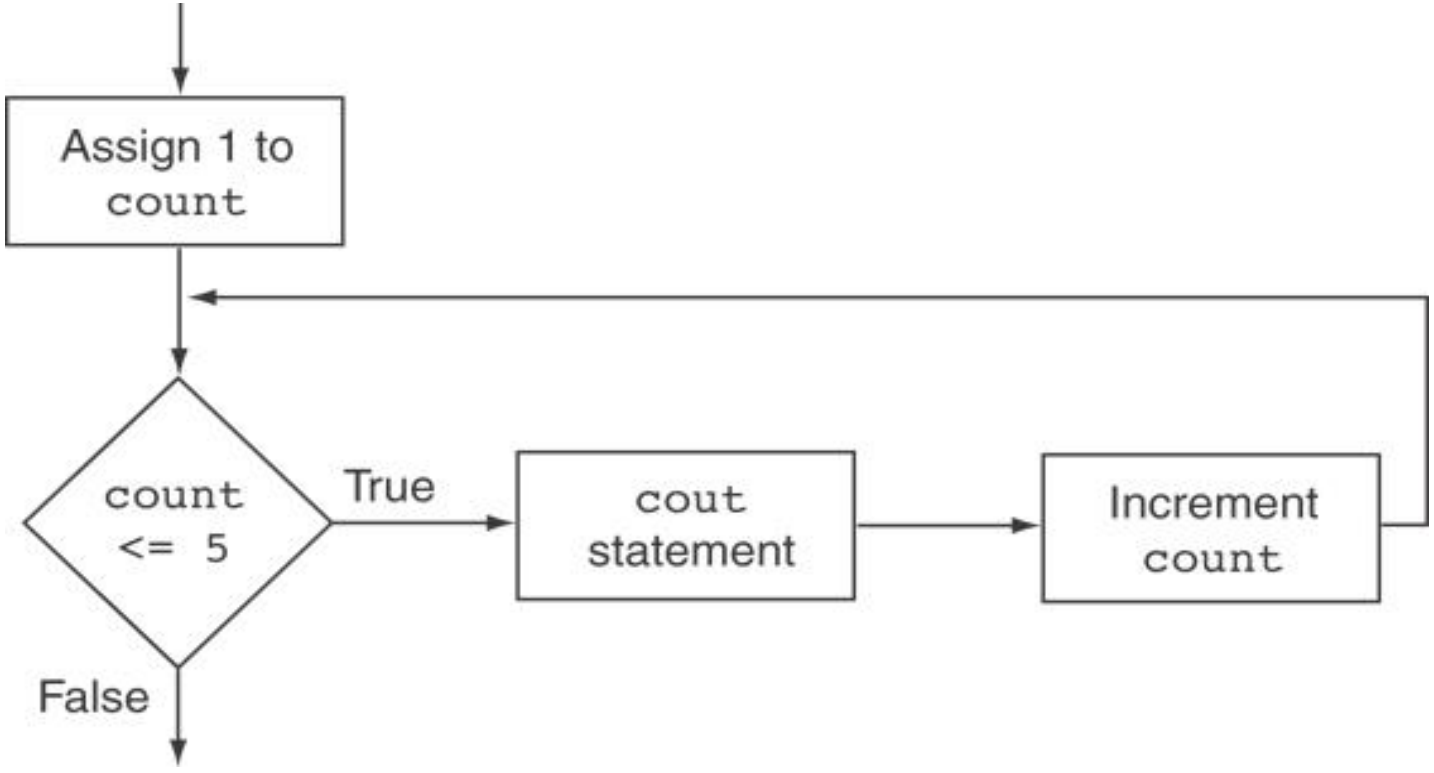
`for` Loop - Example

```
int count;

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

A Closer Look at the Previous Example



Flowchart for the Previous Example



A `for` Loop example

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 }
```

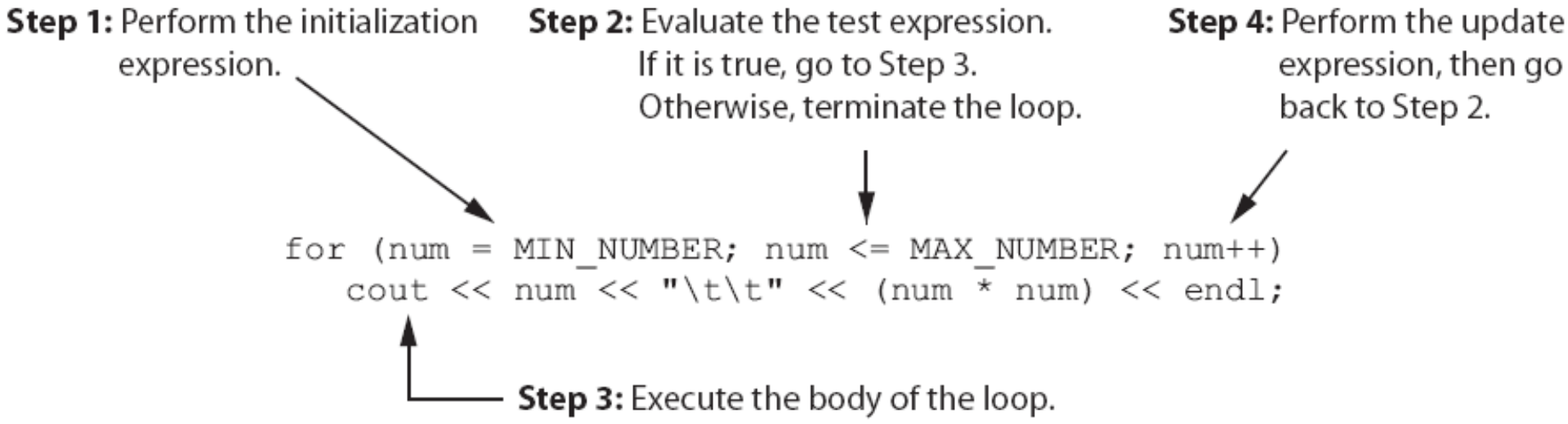
Continued…

A `for` Loop example

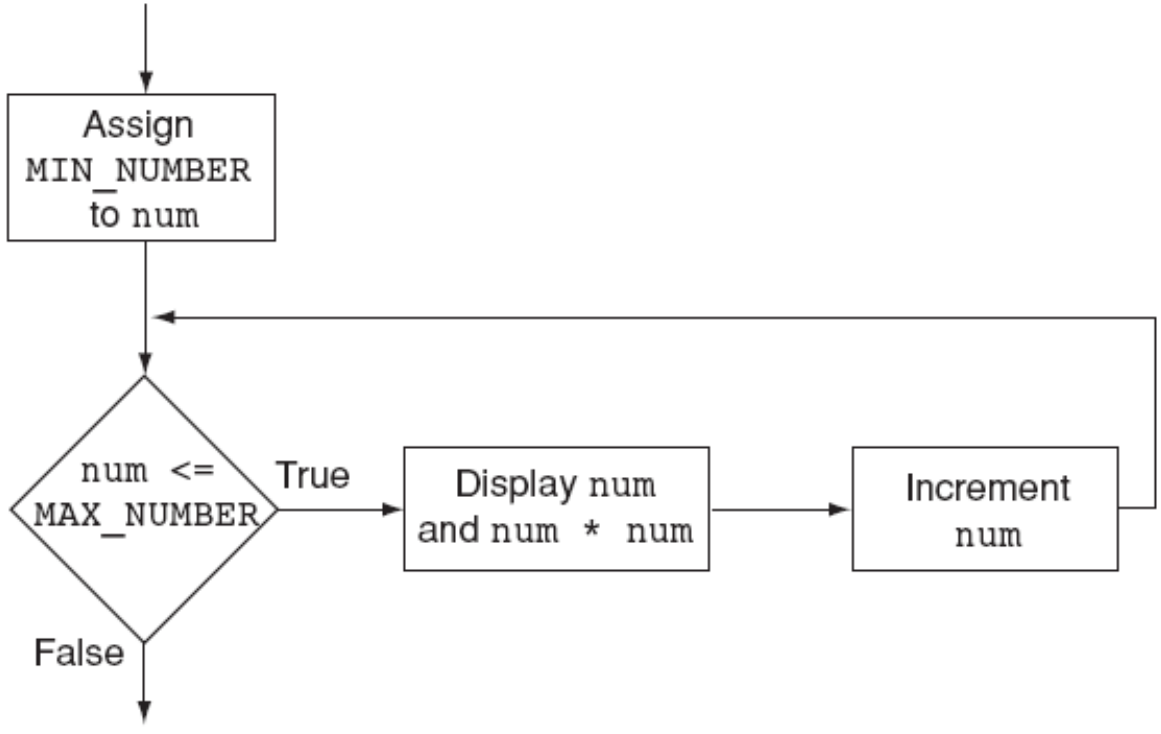
Program Output

```
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 Lines 15 through 16 example



Flowchart for Lines 15 through 16 in example



When to Use the `for` Loop

In any situation that clearly requires

1. an initialization
2. a false condition to stop the loop
3. 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;
```

The `while` Loop

Loop: a control structure that causes a statement or statements to repeat

General format of the `while` loop:

```
while (expression)
    statement;
```

`statement;` can also be a block of statements enclosed in `{ }`

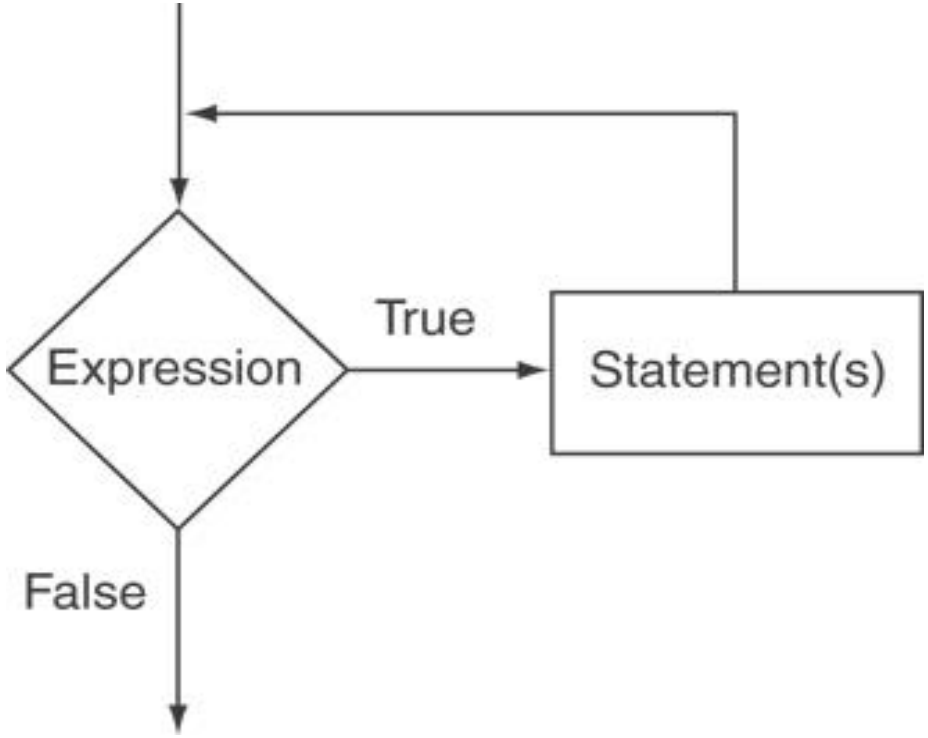
The `while` Loop – How It Works

```
while (expression)
      statement;
```

`expression` is evaluated

- if `true`, then *`statement`* is executed, and *`expression`* is evaluated again
- if `false`, then the loop is finished and program statements following *`statement`* execute

The Logic of a `while` Loop



The `while` loop in Program 5-3

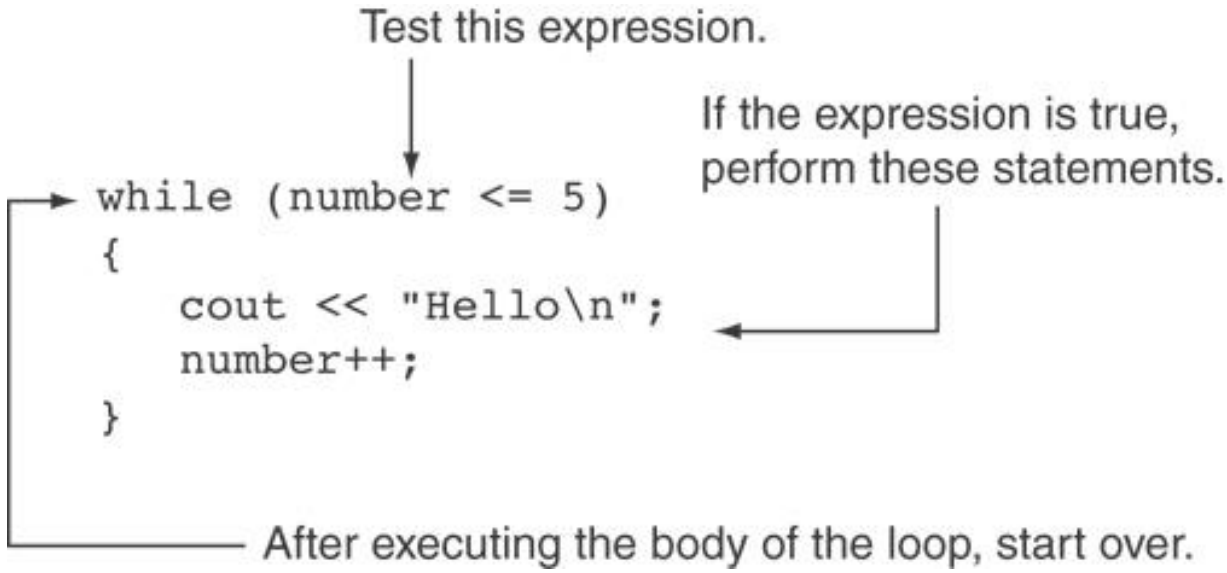
Program 5-3

```
1  // This program demonstrates a simple while loop.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int number = 1;
8
9     while (number <= 5)
10    {
11       cout << "Hello\n";
12       number++;
13    }
14    cout << "That's all!\n";
15    return 0;
16 }
```

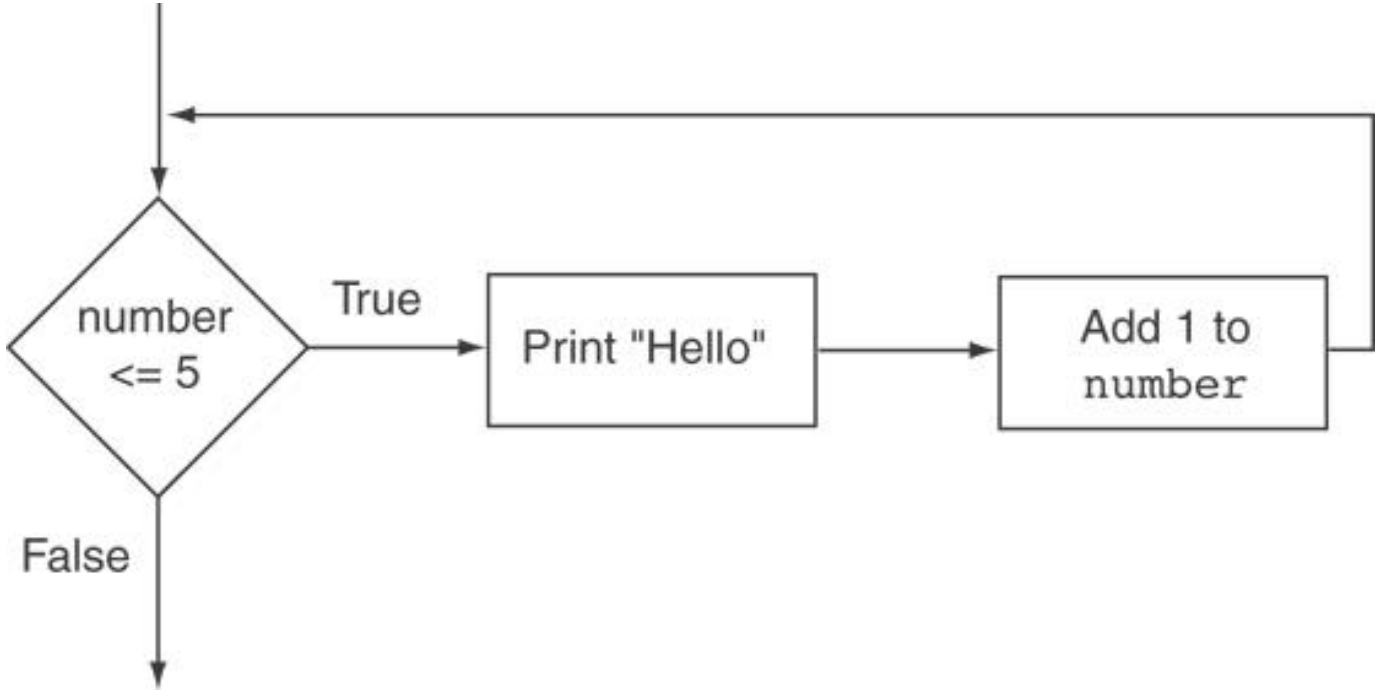
Program Output

```
Hello
Hello
Hello
Hello
Hello
That's all!
```

How the `while` Loop in example Lines 9 through 13 Works



Flowchart of the `while` Loop in Program 5-3



The `while` Loop is a Pretest Loop

`expression` is evaluated *before* the loop executes. The following loop will never execute:

```
int number = 6;
while (number <= 5)
{
   cout << "Hello\n";
   number++;
}
```

Watch Out for Infinite Loops

The loop must contain code to make *`expression`* become `false`

Otherwise, the loop will have no way of stopping

Such a loop is called an *infinite loop*, because it will repeat an infinite number of times

Example of an Infinite Loop

```
int number = 1;
while (number <= 5)
{
    cout << "Hello\n";
}
```

In Pairs or Triples:

- *Predict what the C++ code will do:*

```
1  //Another C++ program, demonstrating variables
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int year;
8    cout << "Enter a number: ";
9    cin >> year;
10   cout << "Hello " << year << "!!\n\n";
11   return 0;
12 }
```

onlinegdb demo

```cpp
//Another C++ program, demonstrating variables
#include <iostream>
using namespace std;

int main ()
{
  int year;
  cout << "Enter a number: ";
  cin >> year;
  cout << "Hello " << year << "!!\n\n";
  return 0;
}
```

(Demo with `onlinegdb`)

In Pairs or Triples:

Predict what the following pieces of code will do:

```
//Another C++ program, demonstrating I/O & arithmetic
#include <iostream>
using namespace std;

int main ()
{
  float kg, lbs;
  cout << "Enter kg: ";
  cin >> kg;
  lbs = kg * 2.2;
  cout << endl << "Lbs: " << lbs << "\n\n";
  return 0;
}
```

C++ Demo

```cpp
//Another C++ program, demonstrating I/O & arithmetic
#include <iostream>
using namespace std;

int main ()
{
  float kg, lbs;
  cout << "Enter kg: ";
  cin >> kg;
  lbs = kg * 2.2;
  cout << endl << "Lbs: " << lbs << "\n\n";
  return 0;
}
```

(Demo with `onlinegdb`)

In Pairs or Triples:

Predict what the following pieces of code will do:

```
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j;
  for (i = 0; i < 4; i++)
  {
      cout << "The world turned upside down...\n";
  }

  for (j = 10; j > 0; j--)
  {
      cout << j << " ";
  }
  cout << "Blast off!!" << endl;

  return 0;
}
```

C++ Demo

```cpp
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j;
  for (i = 0; i < 4; i++)
  {
      cout << "The world turned upside down...\n";
  }

  for (j = 10; j > 0; j--)
  {
      cout << j << " ";
  }
  cout << "Blast off!!" << endl;

  return 0;
}
```

(Demo with `onlinegdb`)

Definite loops

```
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j;
  for (i = 0; i < 4; i++)
  {
      cout << "The world turned upside down...\n";
  }

  for (j = 10; j > 0; j--)
  {
      cout << j << " ";
  }
  cout << "Blast off!!" << endl;

  return 0;
}
```

General format:

`for` (*initialization* ; *test* ; *updateAction*)
{

command1;
command2;
command3;
...

}

In Pairs or Triples:

Predict what the following pieces of code will do:

```
//Growth example
#include <iostream>
using namespace std;

int main ()
{
  int population = 100;
  cout << "Year\tPopulation\n";
  for (int year = 0; year < 100; year= year+5)
  {
      cout << year << "\t" << population << "\n";
      population = population * 2;

  }
  return 0;
}
```

C++ Demo

```cpp
//Growth example
#include <iostream>
using namespace std;

int main ()
{
  int population = 100;
  cout << "Year\tPopulation\n";
  for (int year = 0; year < 100; year= year+5)
  {
      cout << year << "\t" << population << "\n";
      population = population * 2;

  }
  return 0;
}
```

(Demo with C++)

In Pairs or Triples:

Predict what the following pieces of code will do:

```
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j,size;
  cout << "Enter size: ";
  cin >> size;
  for (i = 0; i < size; i++)
  {
    for (j = 0; j < size; j++)
      cout << "*";
    cout << endl;
  }
  cout << "\n\n";
  for (i = size; i > 0; i--)
  {
    for (j = 0; j < i; j++)
      cout << "*";
    cout << endl;
  }
  return 0;
}
```

C++ Demo

```cpp
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j,size;
  cout << "Enter size: ";
  cin >> size;
  for (i = 0; i < size; i++)
  {
    for (j = 0; j < size; j++)
      cout << "*";
    cout << endl;
  }
  cout << "\n\n";
  for (i = size; i > 0; i--)
  {
    for (j = 0; j < i; j++)
      cout << "*";
    cout << endl;
  }
  return 0;
}
```

(Demo with C++)

C++

```
//Growth example
#include <iostream>
using namespace std;

int main ()
{
  int population = 100;
  cout << "Year\tPopulation\n";
  for (int year = 0; year < 100; year= year+5)
  {
      cout << year << "\t" << population << "\n";
      population = population * 2;

  }
  return 0;
}
```