CIS 22A – Lecture 10

Manish Goel

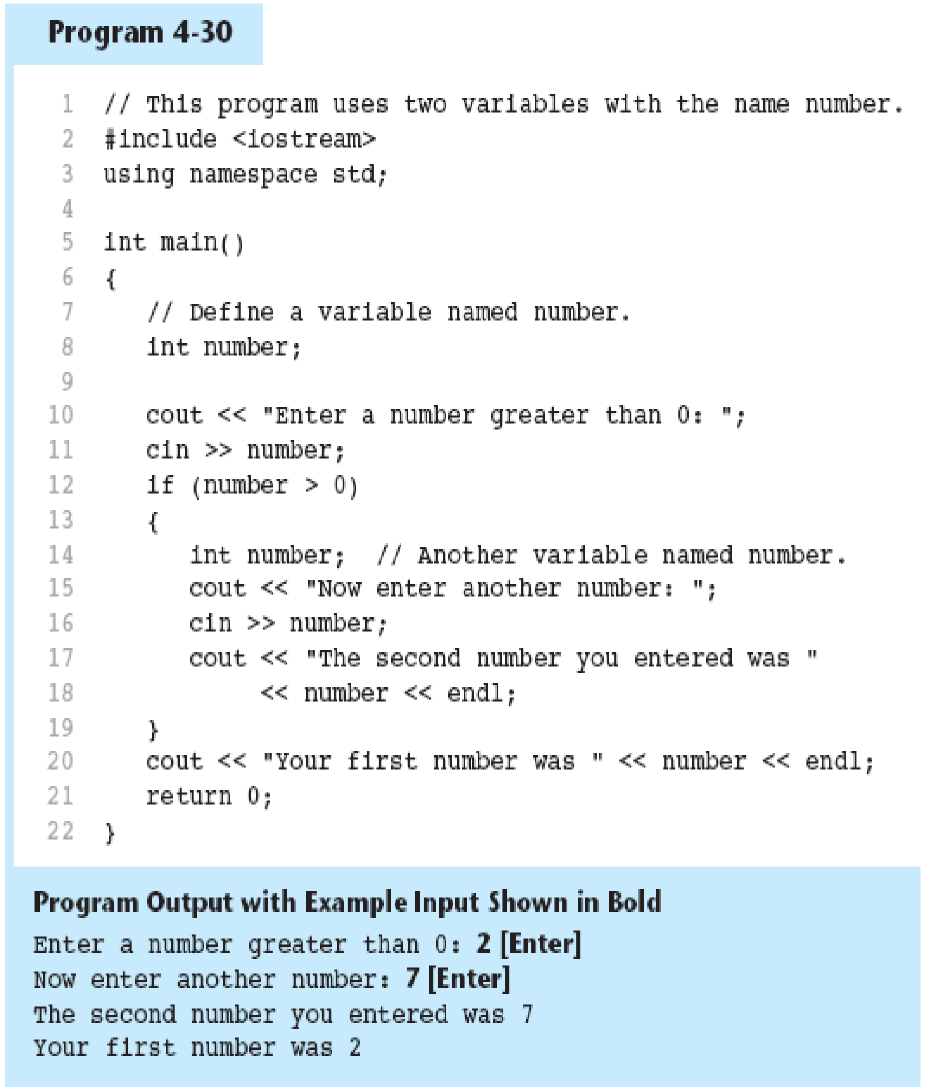
More About Blocks and Scope

- Scope of a variable is the block in which it is defined, from the point of definition to the end of the block
- Usually defined at beginning of function
- May be defined close to first use

```
if (income >= MIN_INCOME)
{
   // Get the number of years at the current job.
   cout << "How many years have you worked at "
        << "your current job? ";
   int years;     // Variable definition
   cin >> years;

   if (years > MIN_YEARS)
      cout << "You qualify.\n";
   else
   {
      cout << "You must have been employed for\n"
           << "more than " << MIN_YEARS
           << " years to qualify.\n";
   }
}
```

Variables with the Same Name

- Variables defined inside { } have <u>local</u> or <u>block</u> scope
- When inside a block within another block, can define variables with the same name as in the outer block.
  - When in inner block, outer definition is not available
  - Not a good idea



Program 4-30

```
// This program uses two variables with the name number.
#include <iostream>
using namespace std;

int main()
{
   // Define a variable named number.
   int number;

   cout << "Enter a number greater than 0: ";
   cin >> number;
   if (number > 0)
   {
      int number;  // Another variable named number.
      cout << "Now enter another number: ";
      cin >> number;
      cout << "The second number you entered was "
           << number << endl;
   }
   cout << "Your first number was " << number << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter a number greater than 0: 2 [Enter]
Now enter another number: 7 [Enter]
The second number you entered was 7
Your first number was 2
```

More Mathematical Library Functions

- These require `cstdlib` header file
- `rand()`: returns a random number (`int`) between `0` and the largest int the compute holds. Yields same sequence of numbers each time program is run
- `srand(x)`: initializes random number generator with `unsigned int x`
- `srand(time(0))`: initializes random generator with different values to generate truly random numbers. Include `ctime` header file

The Increment and Decrement Operators

- `++` is the increment operator. It adds one to a variable.
  `val++;` is the same as `val = val + 1;`

- `--` is the decrement operator. It subtracts one from a variable.
  `val--;` is the same as `val = val - 1;`

- `++` and `--` can be used before (prefix) or after (postfix) a variable:

```
 ++val;  val++;              --val;     val--;
```

- In prefix mode (`++val, --val`), the operator increments or decrements, *then* returns the value of the variable
- In postfix mode (`val++, val--`), the operator returns the value of the variable, *then* increments or decrements

Increment / Decrement with Prefix / Postfix

Program 5-2

```cpp
// This program demonstrates the prefix and postfix
// modes of the increment and decrement operators.
#include <iostream>
using namespace std;

int main()
{
   int num = 4;

   cout << num << endl;    // Displays 4
   cout << num++ << endl;  // Displays 4, then adds 1 to num
   cout << num << endl;    // Displays 5
   cout << ++num << endl;  // Adds 1 to num, then displays 6
   cout << endl;           // Displays a blank line

   cout << num << endl;    // Displays 6
   cout << num-- << endl;  // Displays 6, then subtracts 1 from num
   cout << num << endl;    // Displays 5
   cout << --num << endl;  // Subtracts 1 from num, then displays 4

   return 0;
}
```

Program Output

```
4
4
5
6

6
6
5
4
```

Notes on Increment and Decrement

- Can be used in expressions:

```
result = num1++ + num2++;
result = ++num1 + ++num2;
result = num1++ + ++num2;
```

  pre- and post-operations will cause different results

- Must be applied to something that has a location in memory. Following cause errors:

```
result = (num1 + num2)++;
```

- Can be used in relational expressions:

```
if (++num > limit)
```

  pre- and post-operations will cause different comparisons

Loops – Parts of programs that repeat

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

- Three loop-ing structures in C++
  - the `while` loop
  - the `do-while` loop
  - the `for` loop

- Loop constructs differ in their control of repetition

The `while` Loop

- General format of the `while` loop:

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

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

- *`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

- `while` Loop is a Pretest Loop - *`expression`* is evaluated *before* the loop executes.

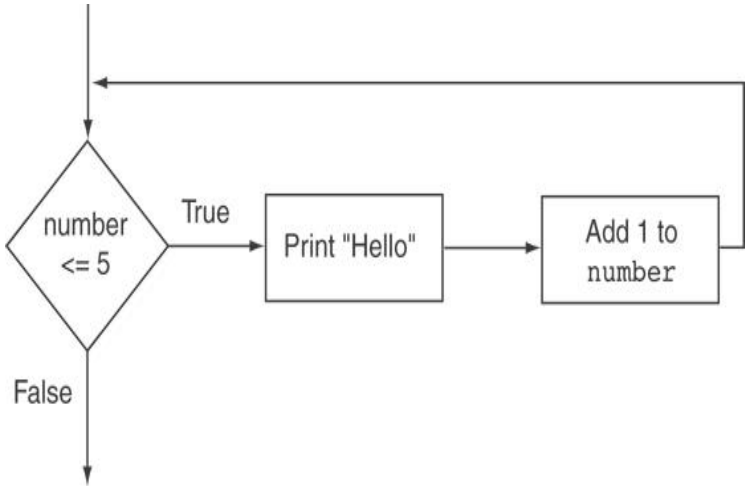
The `while` loop in Program 5-3

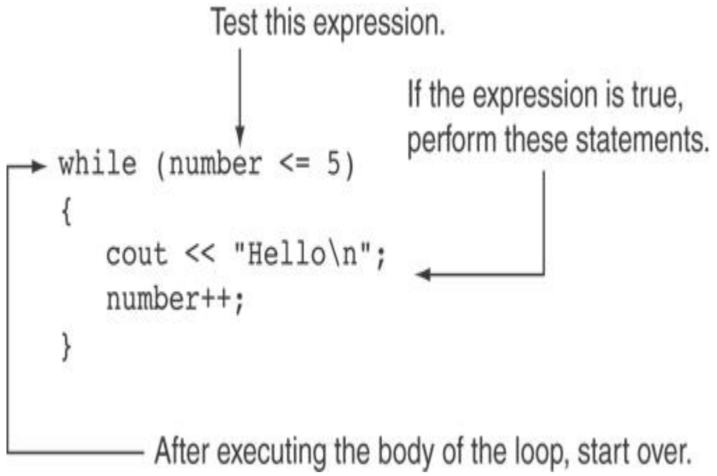
Program 5-3

```
// This program demonstrates a simple while loop.
#include <iostream>
using namespace std;

int main()
{
   int number = 1;

   while (number <= 5)
   {
      cout << "Hello\n";
      number++;
   }
   cout << "That's all!\n";
   return 0;
}
```

Program Output

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




Test this expression.
If the expression is true, perform these statements.
After executing the body of the loop, start over.

```
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

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

Using the `while` Loop for Input Validation

- The while loop can be used to create input routines that reject invalid data, and repeat until valid data is entered.

```
Read an item of input.
While the input is invalid
    Display an error message.
    Read the input again.
End While
```

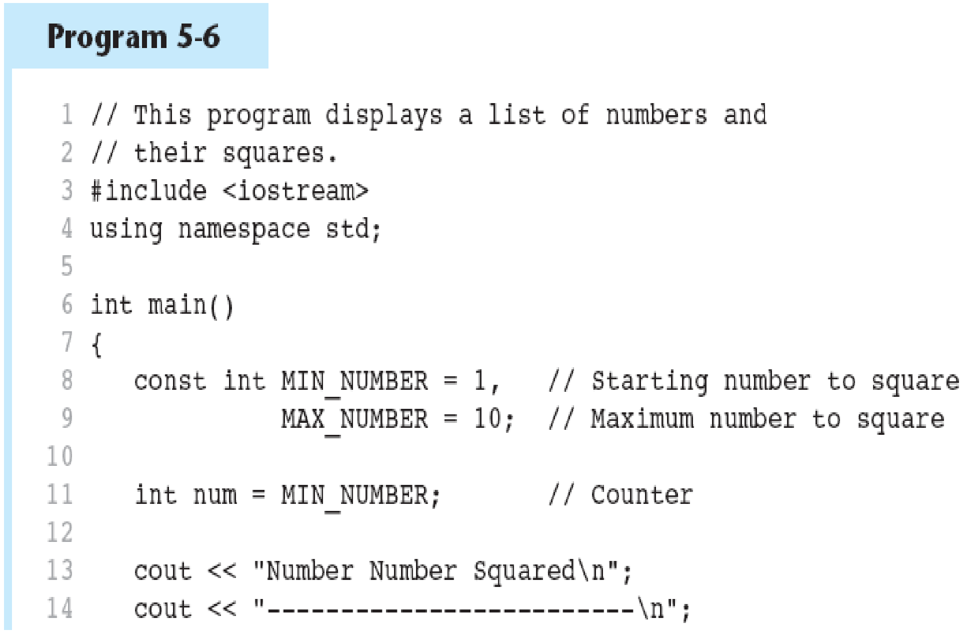
```
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
   cout << "Invalid Entry!"
        << "Enter a number less than 10: ";
   cin >> number;
}
```

Input Validation in Program 5-5

```
// Get the number of players per team.
cout << "How many players do you wish per team? ";
cin >> teamPlayers;

// Validate the input.
while (teamPlayers < MIN_PLAYERS || teamPlayers > MAX_PLAYERS)
{
   // Explain the error.
   cout << "You should have at least " << MIN_PLAYERS
        << " but no more than " << MAX_PLAYERS << " per team.\n";

   // Get the input again.
   cout << "How many players do you wish per team? ";
   cin >> teamPlayers;
}

// Get the number of players available.
cout << "How many players are available? ";
cin >> players;

// Validate the input.
while (players <= 0)
{
   // Get the input again.
   cout << "Please enter 0 or greater: ";
   cin >> players;
}
```

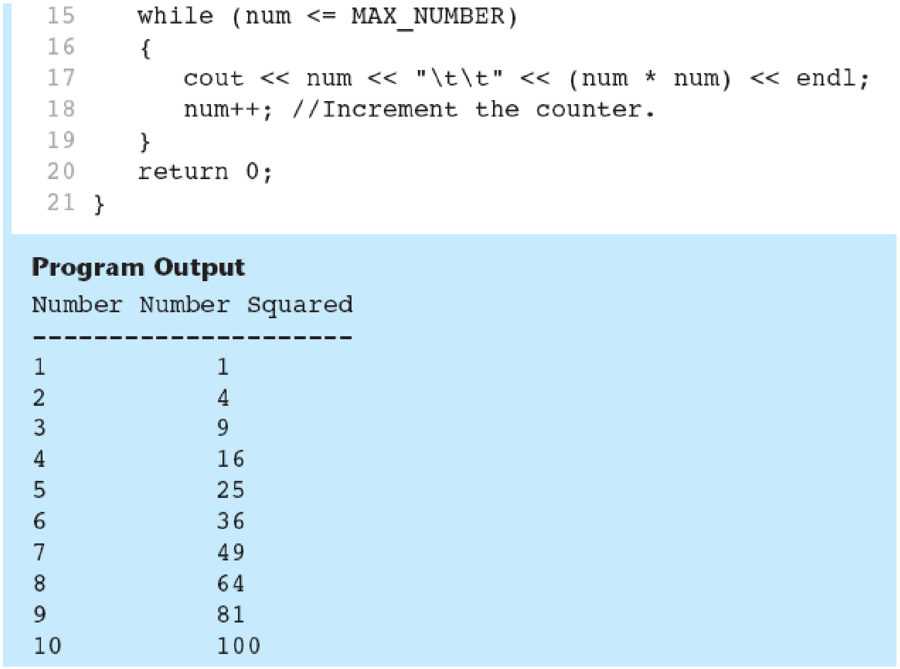
Counters

- Counter: a variable that is incremented or decremented each time a loop repeats
- Can be used to control execution of the loop (also known as the *loop control variable*)
- Must be initialized before entering loop



Program 5-6

```cpp
// This program displays a list of numbers and
// their squares.
#include <iostream>
using namespace std;

int main()
{
   const int MIN_NUMBER = 1,    // Starting number to square
             MAX_NUMBER = 10;   // Maximum number to square

   int num = MIN_NUMBER;        // Counter

   cout << "Number Number Squared\n";
   cout << "-------------------------\n";
```



```cpp
   while (num <= MAX_NUMBER)
   {
      cout << num << "\t\t" << (num * num) << endl;
      num++; //Increment the counter.
   }
   return 0;
}
```

Program Output

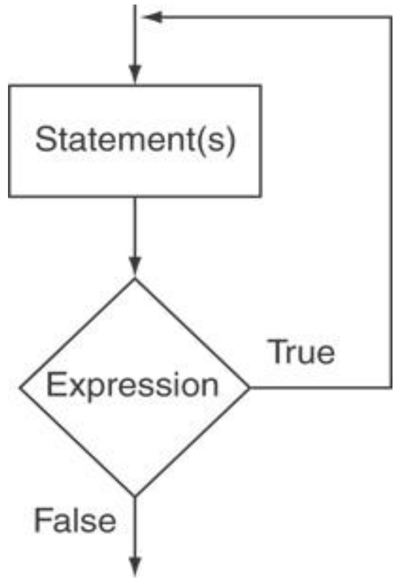
```
Number Number Squared
-------------------------
1               1
2               4
3               9
4               16
5               25
6               36
7               49
8               64
9               81
10              100
```

The `do-while` Loop

- A post-test loop – execute the loop, then test the `expression`
- General Format:

```
     do
            statement;    // or block in { }
     while (expression); // ; required after expression
```

- Always executes at least once; repeatedly executes as long as *`expression`* is `true`, stops repetition when *`expression`* becomes `false`



```
int x = 1;
do
{
    cout << x << endl;
} while(x < 0);
```

Although the test expression is false, this loop will execute one time because `do-while` is a post-test loop.