CSE 1062 **Fundamentals of Programming**

Lecture #5

Spring 2016

Computer Science & Engineering Program
The School of EE & Computing
Adama Science & Technology University

I/O statements and functions

- Using Library Functions
  - Formatting Numbers for Program Output
  - Mathematical Library Functions
- Input using the **`cin`** object
- Symbolic Constants
- Writing Functions
  - Function and parameter declarations
  - Returning a value
  - Pass by Value and Pass by Reference
- Variable Scope and Lifetime

Case Study: (Chemistry) Acid Rain
(General Math) Length Conversion
(General Math) Rectangular to Polar Coordinate Conversion

Reading assignment

- Chapter 3 of the textbook
- Chapter 6 of the textbook
- Read about
  - **Function Overloading**
  - **Default Arguments**

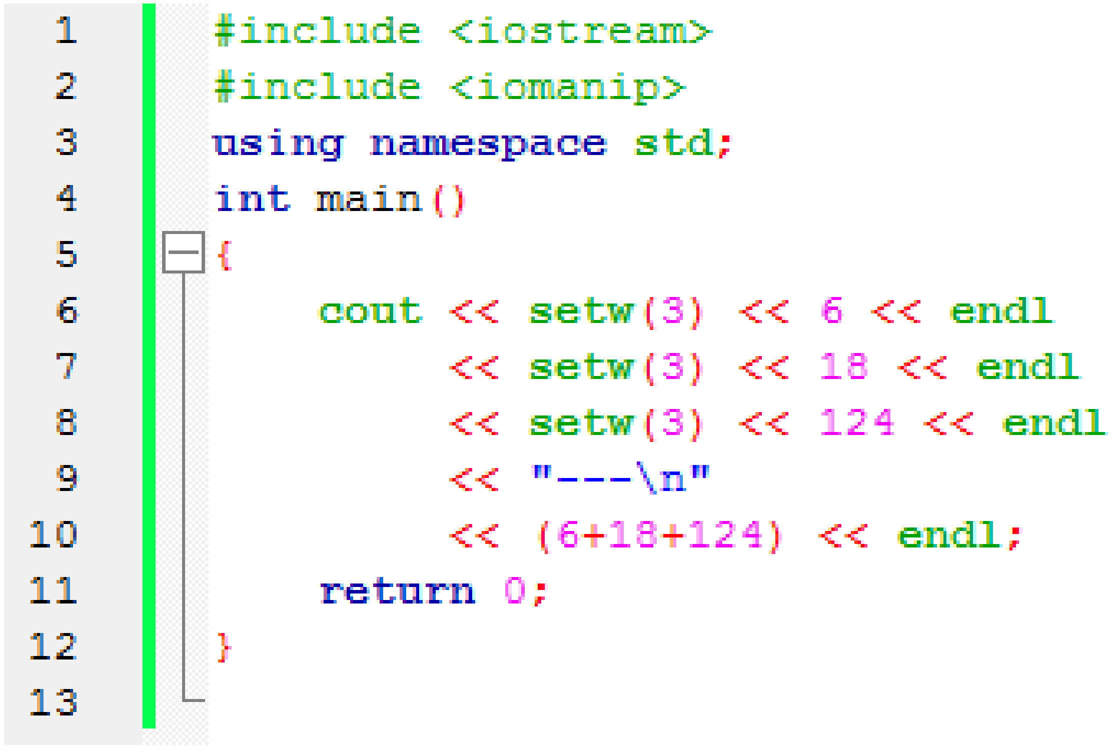
Formatting Numbers for Program Output

ASTU

- Proper output formatting contributes to ease of use and user satisfaction
- **cout** with stream manipulators can cont rol output formatting

Manipulator	Action
setw (n)	Set the field width to n
scientific	Set the output to display real numbers in scientific notation
endl	Output a newline character and display all characters in the buffer
fixed	Always show a decimal point and use a default of six digits after the decimal point. Fill with trailing zeros, if necessary.
setprecision(n)	Set the floating-point precision to n places.

Formatting Numbers for Program Output



```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    cout << setw(3) << 6 << endl
         << setw(3) << 18 << endl
         << setw(3) << 124 << endl
         << "---\n"
         << (6+18+124) << endl;
    return 0;
}
```

Formatting Numbers for Program Output

- The field width manipulator must be incl uded for each value in the data stream s ent to **`cout`**
- Other manipulators remain in effect until they are changed
- **`iomanip`** header file must be included to use manipulators requiring arguments

Formatting Numbers for Program Output

- Formatting floating-point numbers requires three field-width manipulators to:
  - Set the total width of the display
  - Force a decimal place
  - Set the number of significant digits after th e decimal point
- Example:

```
 cout << "|" << setw(10) << fixed
        << setprecision(3) << 25.67 << "|";
```

produces this output: `|    25.670|`

Using Mathematical Library Functions

- C++ has preprogrammed mathematical functi ons that can be included in a program
- You must include the **`cmath`** header file:

```
#include <cmath>
```

  - Math functions require one or more arguments as i nput, but will return only one value
  - All functions are **overloaded**, and can be used with integer and real arguments

Using Mathematical Library Functions

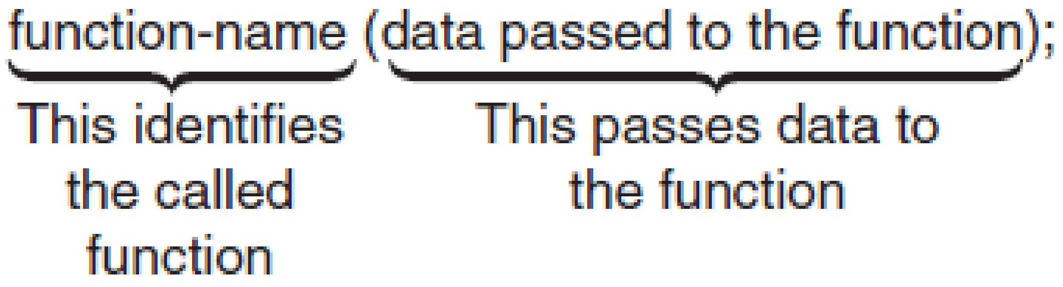
ASTU

Function Name	Description	Returned Value
`abs(a)`	Absolute value	Same data type as argument
`pow(a1,a2)`	`a1` raised to the `a2` power	Same data type as argument `a1`
`sqrt(a)`	Square root of a real number	Double-precision
`sin(a)`	Sine of `a` (`a` in radians)	Double
`cos(a)`	Cosine of `a` (`a` in radians)	Double
`tan(a)`	Tangent of `a` (`a` in radians)	Double
`log(a)`	Natural logarithm of `a`	Double
`log10(a)`	Common log (base 10) of `a`	Double
`exp(a)`	`e` raised to the `a` power	Double

Using Mathematical Library Functions

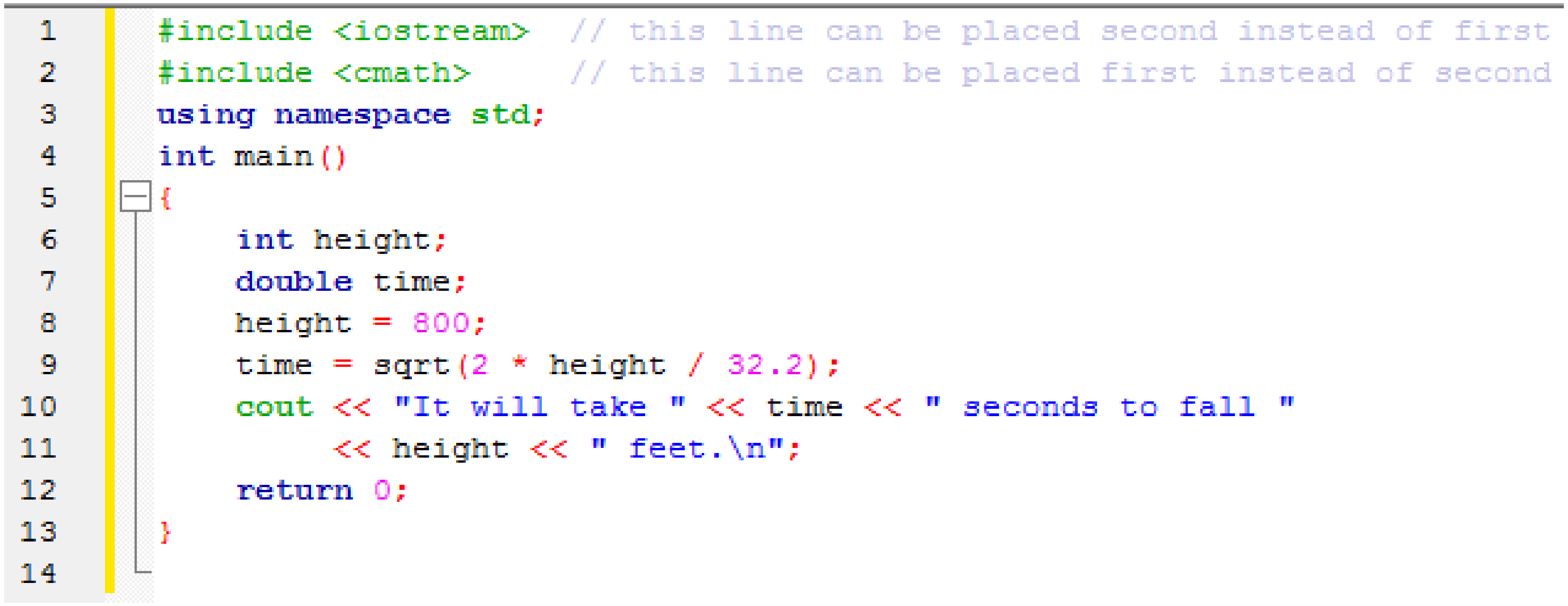
ASTU

- To use a math function, give its name and pass the input arguments within parentheses
- Expressions that can be evaluated to a value can be passed as arguments


function-name (data passed to the function);
This identifies the called function
This passes data to the function

Using Mathematical Library Functions

ASTU



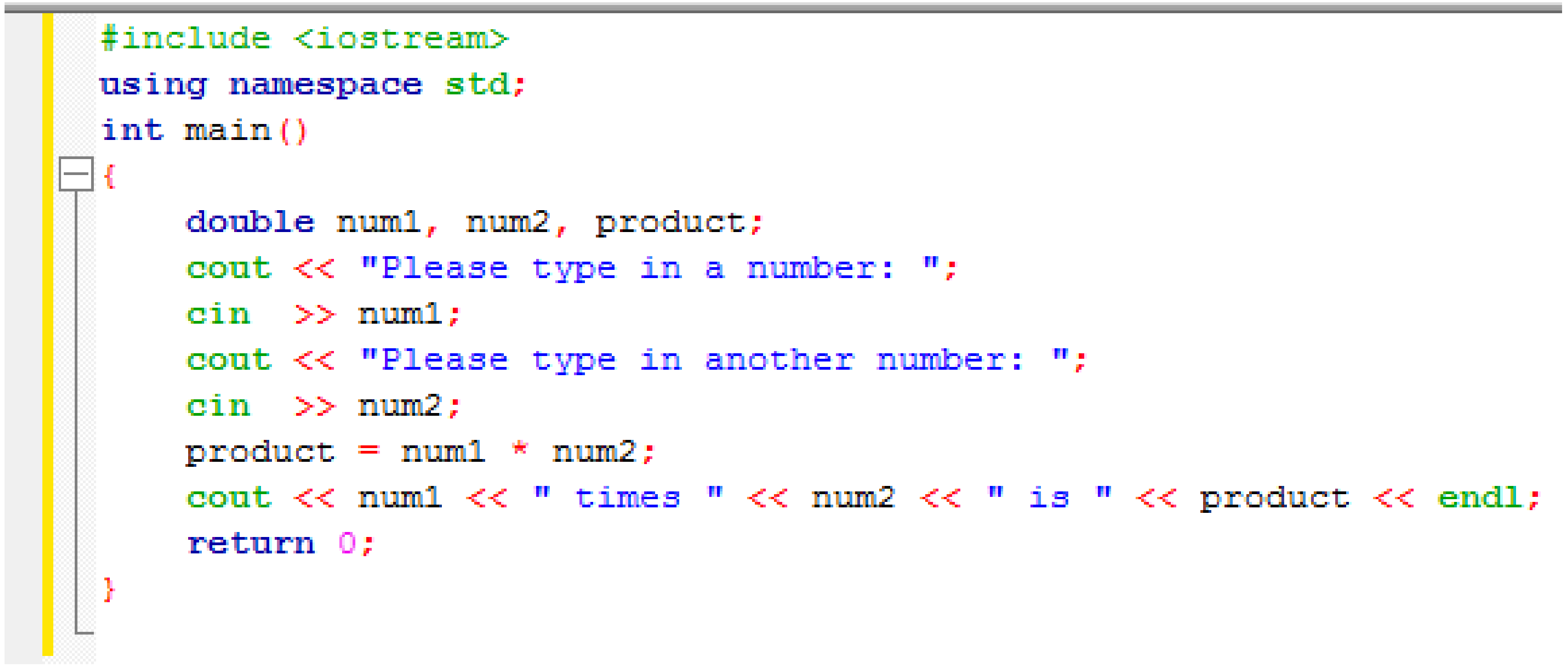
```cpp
#include <iostream>  // this line can be placed second instead of first
#include <cmath>     // this line can be placed first instead of second
using namespace std;
int main()
{
    int height;
    double time;
    height = 800;
    time = sqrt(2 * height / 32.2);
    cout << "It will take " << time << " seconds to fall "
         << height << " feet.\n";
    return 0;
}
```

Program Input Using cin

- **`cin` Object:** Allows data entry to a runni ng program
- Use of the **`cin`** object causes the progra m to wait for input from the keyboard
- When keyboard entry is complete, the pr ogram resumes execution, using the ent ered data
- An output statement preceding the **`cin`** object statement provides a **prompt** to t he user

Program Input Using cin

ASTU



```
#include <iostream>
using namespace std;
int main()
{
    double num1, num2, product;
    cout << "Please type in a number: ";
    cin  >> num1;
    cout << "Please type in another number: ";
    cin  >> num2;
    product = num1 * num2;
    cout << num1 << " times " << num2 << " is " << product << endl;
    return 0;
}
```

Program Input Using cin

- **`cin`** can accept multiple input values to be stored in different variables
- Multiple numeric input values must be s eparated by spaces

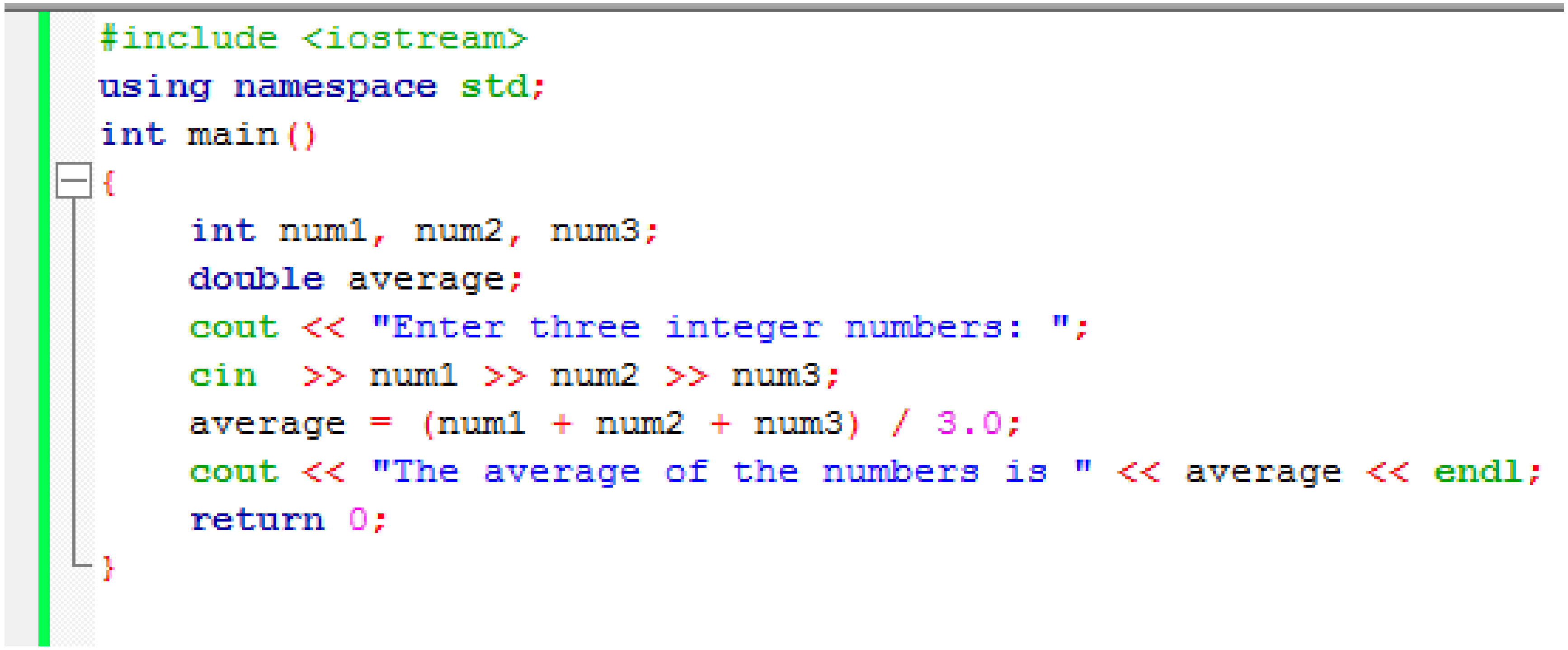
  Example:

  ```
  cin >> num1 >> num2
  ```

  with keyboard entry: **`0.052 245.79`**

Program Input Using cin

ASTU



```cpp
#include <iostream>
using namespace std;
int main()
{
    int num1, num2, num3;
    double average;
    cout << "Enter three integer numbers: ";
    cin  >> num1 >> num2 >> num3;
    average = (num1 + num2 + num3) / 3.0;
    cout << "The average of the numbers is " << average << endl;
    return 0;
}
```

Program Input Using cin

- **User-input validation:** The process of ens uring that data entered by the user matc hes the expected data type
- **Robust program:** One that detects and h andles incorrect user entry

Case Study: Acid Rain

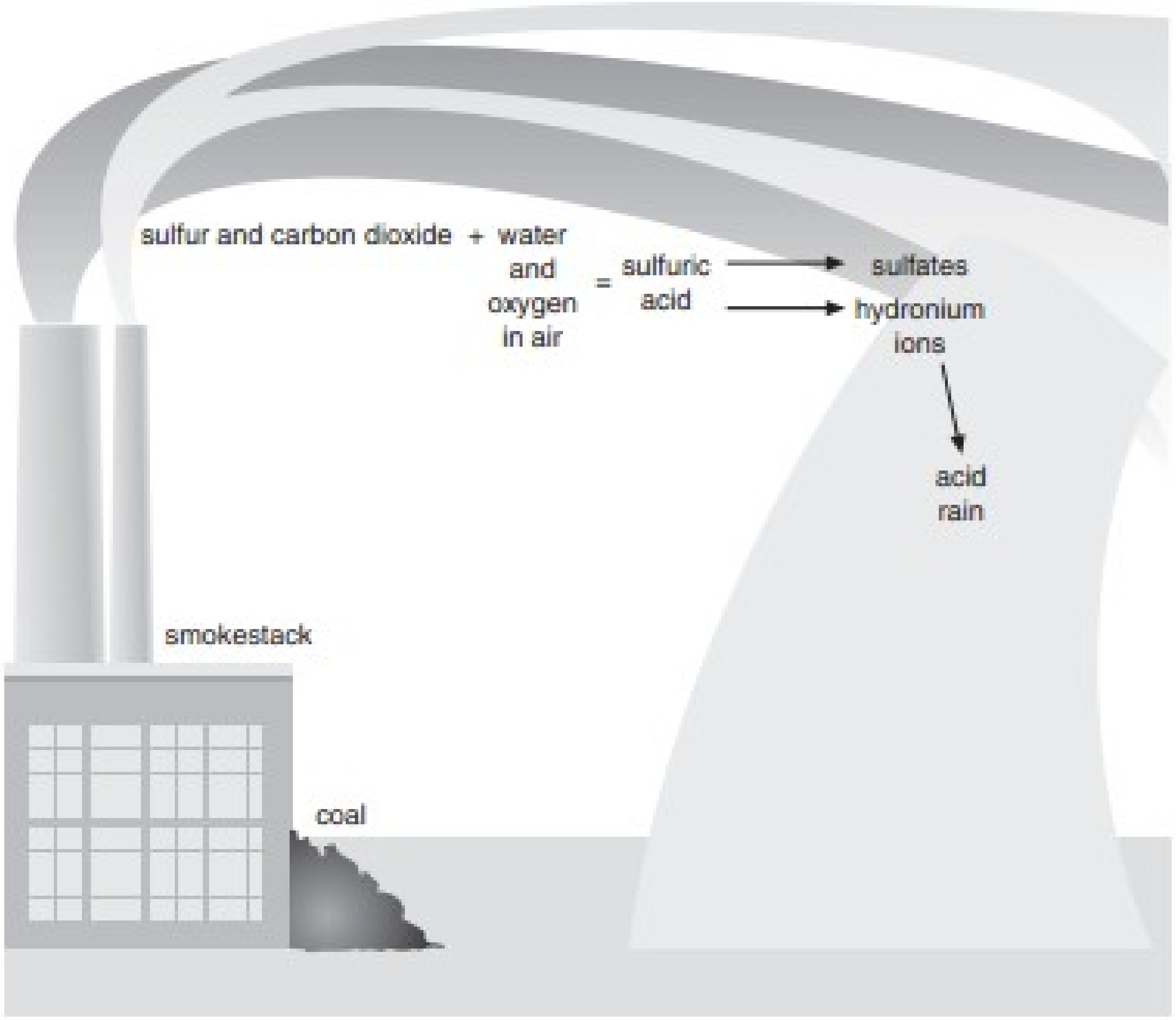
- Currently, coal is one of the principal sou rces of electrical power generation in ma ny industrialized countries.
  - it is known that the oxygen used in the bur ning process combines with the carbon and sulfur in coal to produce carbon dioxide an d sulfur dioxide.

Case Study: Acid Rain

- When these gases are released into the atm osphere, sulfur dioxide com-bines with wat er and oxygen in the air to form sulfuric aci d
  - Then it is transformed into separate hydronium ions a nd sulfates see figure.
- The hydronium ions in the atmosphere that fall to Earth as components of rain are wha t change the acidity levels of lakes and fore sts

Case Study: Acid Rain

ASTU



Case Study: Acid Rain

ASTU

- The acid level of rain and lakes is measur ed on a pH scale by using this formula:

$$pH = -\log_{10}(concentration\ of\ \ hyrodnium\ ions)$$

Case Study: Acid Rain

- The concentration of hydronium ions is measured in units of moles/liter.
- pH value 7 =>neutral value
- pH below 7 =>acid,
- pH above 7=>alkaline
- Marine life usually can’t survive in water with a pH level below 4.

Case Study: Acid Rain

- Using the formula for pH, we will write a C++ program, using the software develo pment procedure described in Lecture 1,

Case Study: Acid Rain

- **Acid Rain:** Develop a program to calculate the pH level of a substance based on user input of the co ncentration of hydronium ions
  - Step 1: Analyze the Problem
  - Step 2: Develop a Solution
  - Step 3: Code the Solution
  - Step 4: Test and Correct the Program

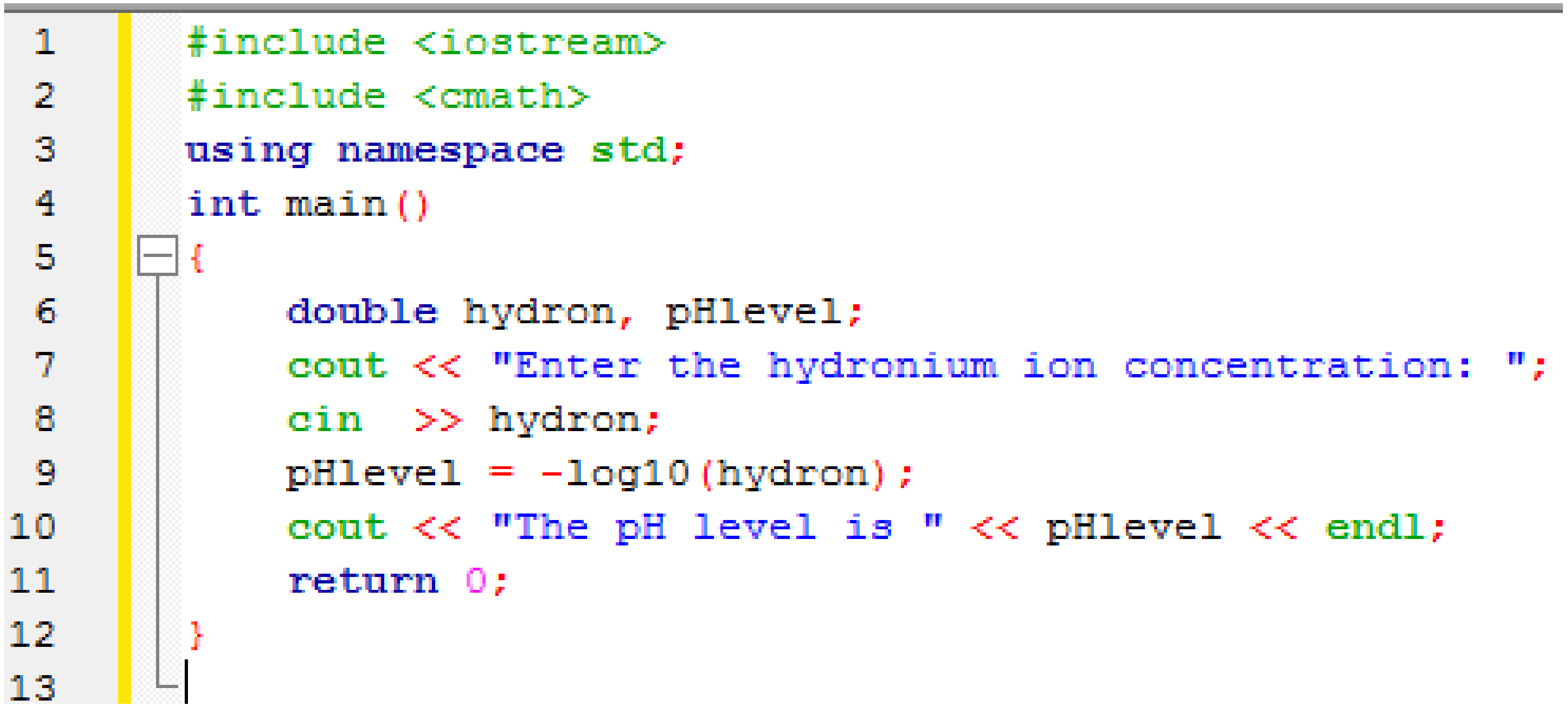
Step 1: Analyze the Problem

- Although the problem statement provides technical information on the composition of acid rain, from a programming viewpoint, this problem is rather simple.
- You have only one required output (a pH level) and one input (the concentration of hydronium ions).

Step 2: Develop a Solution

- Display a prompt to enter an ion concent ration level
- Read a value for the concentration level
- Calculate a pH level, using the given for mula
- Display the calculated value

Step 3: Code the Solution

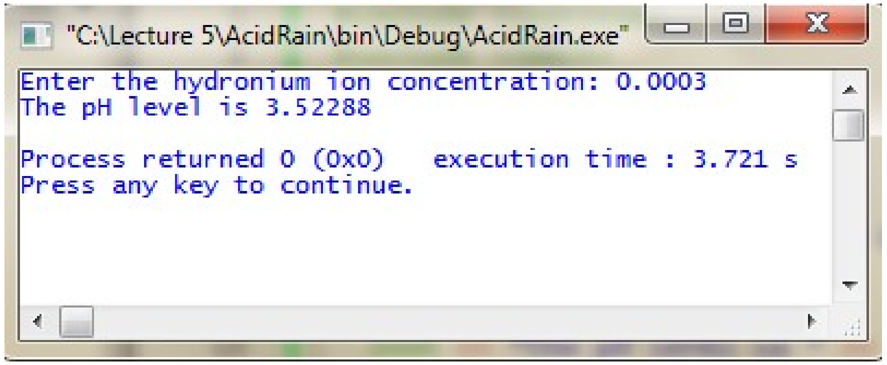


```cpp
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double hydron, pHlevel;
    cout << "Enter the hydronium ion concentration: ";
    cin  >> hydron;
    pHlevel = -log10(hydron);
    cout << "The pH level is " << pHlevel << endl;
    return 0;
}
```

Step 4: Test and Correct the Program

ASTU

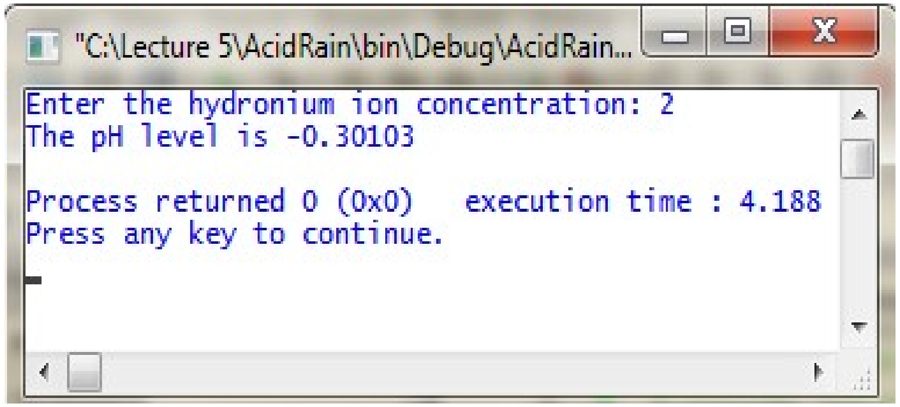
- First test



"C:\Lecture 5\AcidRain\bin\Debug\AcidRain.exe"

```
Enter the hydronium ion concentration: 0.0003
The pH level is 3.52288

Process returned 0 (0x0)   execution time : 3.721 s
Press any key to continue.
```

- Second test
    - Think about validating the user input so that the pH level becomes between 0 and 14
        - We need selection statements



"C:\Lecture 5\AcidRain\bin\Debug\AcidRain...

```
Enter the hydronium ion concentration: 2
The pH level is -0.30103

Process returned 0 (0x0)   execution time : 4.188
Press any key to continue.
```

Symbolic (Named) Constants

- **Symbolic constant:** Constant value that is declared with an identifier using the **`const`** keyword
- A constant's value may not be changed

  Example:

  ```
  const int MAXNUM = 100;
  ```
- Good programming places statements in appropriate order

Case Study: Length Conversion

ASTU

- **Length Conversion:** Write a program that takes as input given lengths expressed in feet and inches. The program should then convert and output the lengths in centimeters. Assume that the given len gths in feet and inches are integers.
  - Step 1: Analyze the Problem
  - Step 2: Develop a Solution
  - Step 3: Code the Solution
  - Step 4: Test and Correct the Program

Step 1: Analyze the Problem

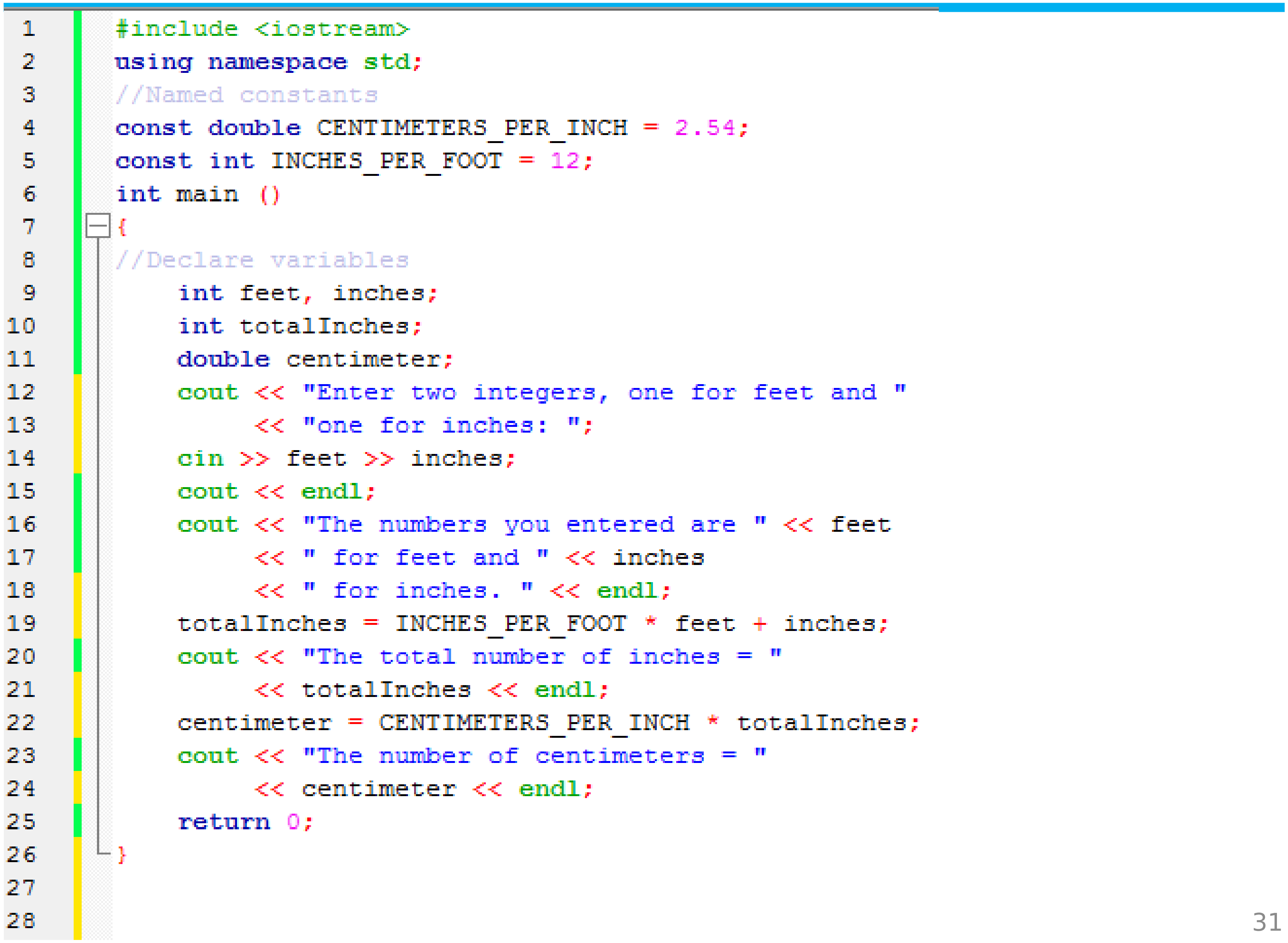
- The lengths are given in feet and inches, and we need to find the equivalent lengt h in centimeters.
- **Input**: Length in feet and inches.
  - **Output**: Equivalent length in centimeters.

Step 2: Develop a Solution

1. Get the length in feet and inches.
2. Convert the length into total inches.
   - Convert the feet into inches
     - Multiply the number of feet by 12
   - Add the given inches
3. Convert total inches into centimeters.
   - Multiply the total inches by 2.54
4. Output centimeters.

Step 3: Coding

ASTU



```cpp
#include <iostream>
using namespace std;
//Named constants
const double CENTIMETERS_PER_INCH = 2.54;
const int INCHES_PER_FOOT = 12;
int main ()
{
//Declare variables
    int feet, inches;
    int totalInches;
    double centimeter;
    cout << "Enter two integers, one for feet and "
         << "one for inches: ";
    cin >> feet >> inches;
    cout << endl;
    cout << "The numbers you entered are " << feet
         << " for feet and " << inches
         << " for inches. " << endl;
    totalInches = INCHES_PER_FOOT * feet + inches;
    cout << "The total number of inches = "
         << totalInches << endl;
    centimeter = CENTIMETERS_PER_INCH * totalInches;
    cout << "The number of centimeters = "
         << centimeter << endl;
    return 0;
}
```

Function and Parameter Declarations

ASTU

- Interaction with a function includes:
  - Passing data to a function correctly when its called
  - Returning values from a function when it ceases o peration
- A function is called by giving the function’s name and passing arguments in the parentheses followi ng the function name

$$\underbrace{\textit{function-name}}_{\substack{\text{This identifies the}\\ \text{called function}}}\ (\underbrace{\textit{data passed to function}}_{\substack{\text{This passes data}\\ \text{to the function}}});$$

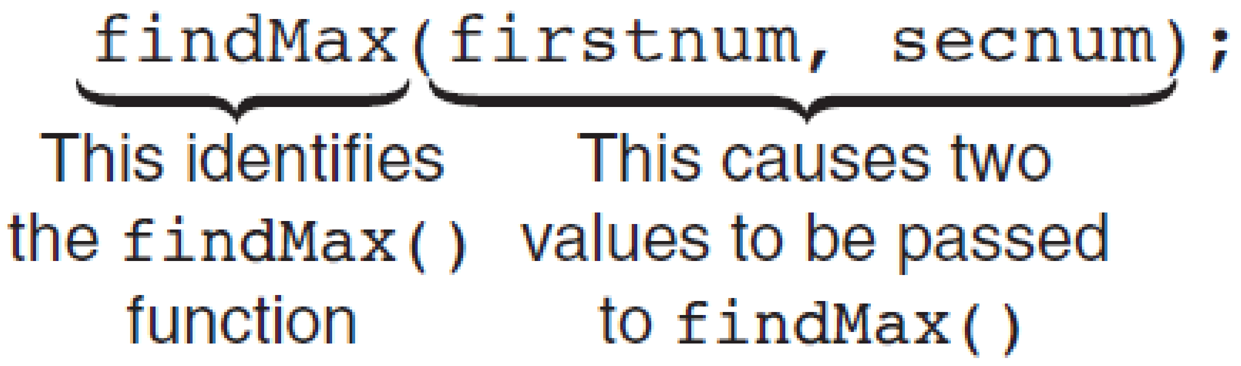
- Before a function is called, it must be declared to functio n that will do calling
- Declaration statement for a function is referred to as func tion prototype
- Function prototype tells calling function:
  - Type of value that will be formally returned, if any
  - Data type and order of the values the calling f unction should transmit to the called function
- Function prototypes can be placed with the variable decla ration statements above the calling function name or in a separate header file

Calling a Function

- Requirements when calling a function include:
  - Using the name of the function
  - Enclosing any data passed to the function in the p arentheses following the function name, using the same order and type declared in the function prot otype

Calling a Function

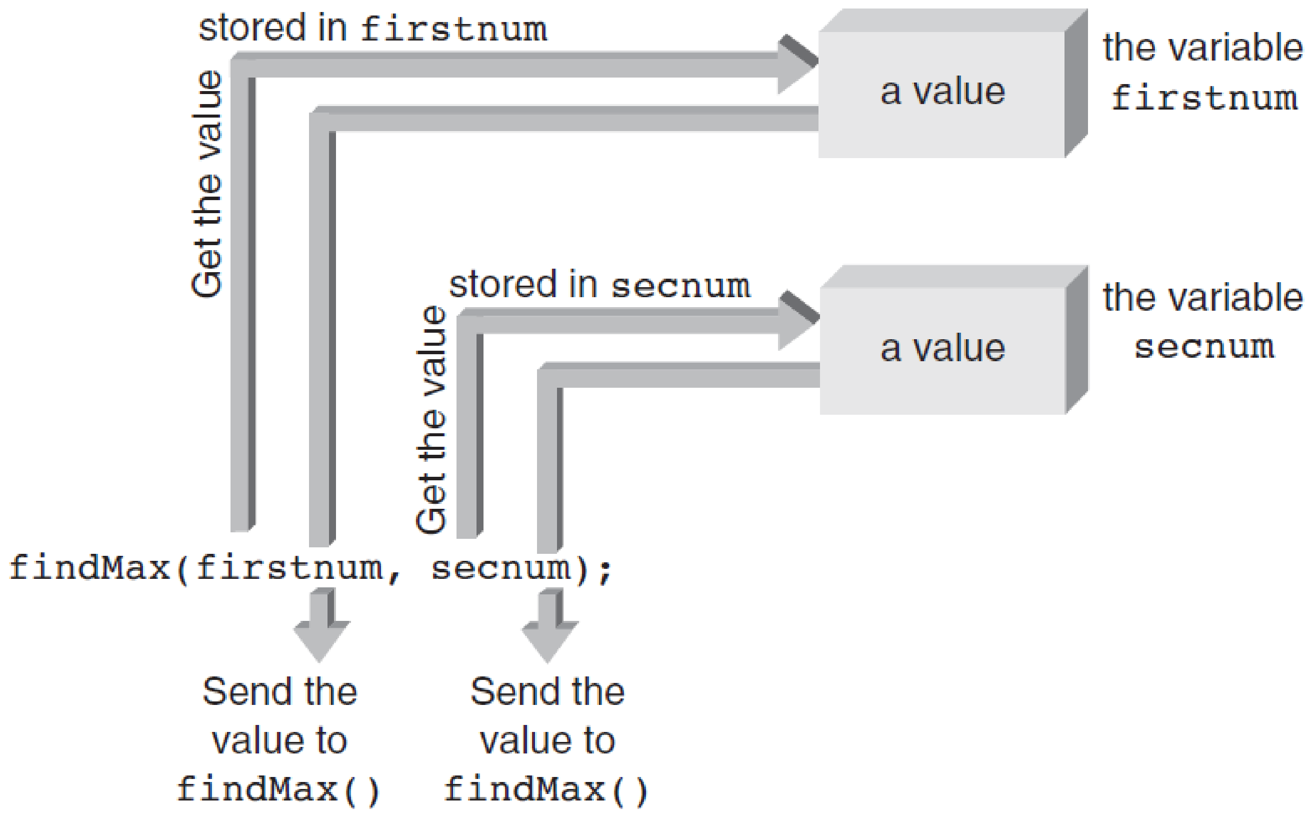
- The items enclosed in the parentheses ar e called **arguments** of the called function



$$\underbrace{\texttt{findMax}}_{\substack{\text{This identifies}\\ \text{the } \texttt{findMax()}\\ \text{function}}}\underbrace{\texttt{(firstnum, secnum)}}_{\substack{\text{This causes two}\\ \text{values to be passed}\\ \text{to } \texttt{findMax()}}}\texttt{;}$$

Calling a Function

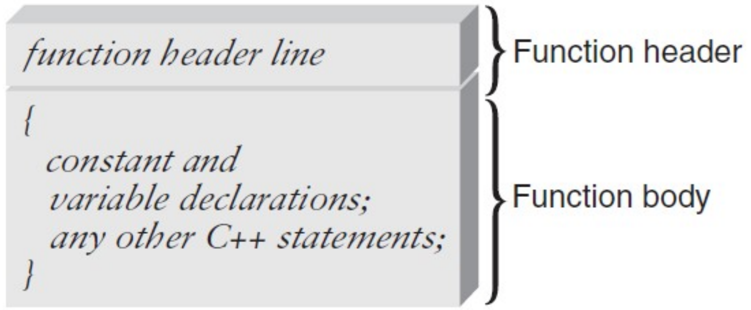
ASTU

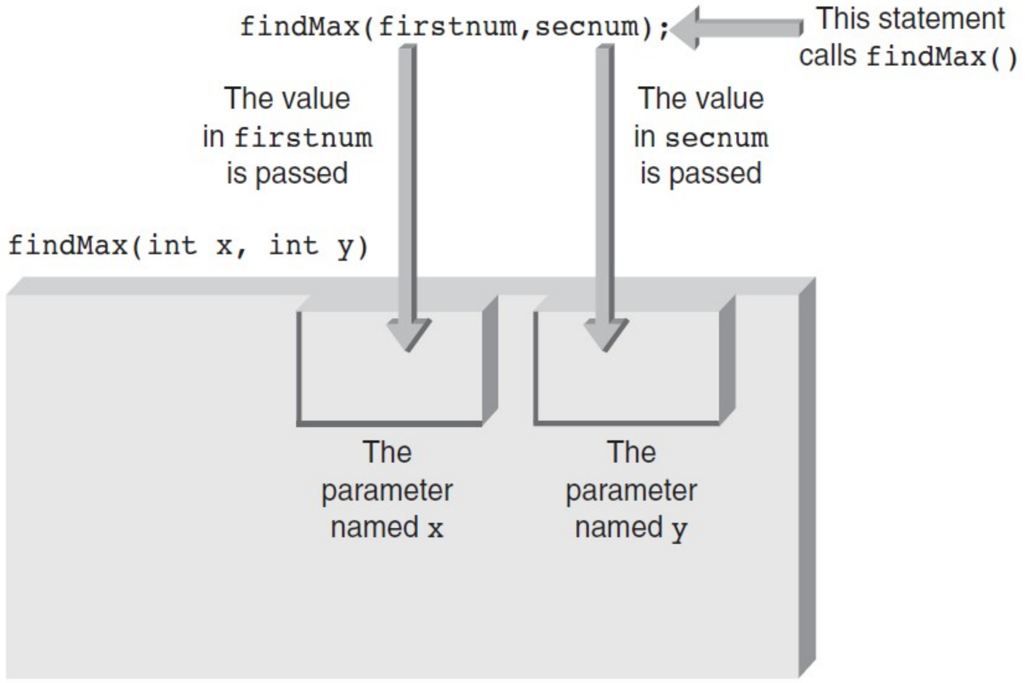


Defining a Function

- Every C++ function consists of two parts:
  - **Function header**
  - **Function body**
- Function header's purpose:
  - Identify data type of value function returns, provid e function with name, and specify number, order, and type of arguments function expects
- Function body's purpose:
  - To operate on passed data and return, at most, on e value directly back to the calling function

Defining and Calling Functions




findMax(firstnum,secnum);
This statement calls findMax()
The value in firstnum is passed
The value in secnum is passed
findMax(int x, int y)
The parameter named x
The parameter named y

Placement of Statements

ASTU

- General rule for placing statements in a C++ prog ram:
  - All preprocessor directives, named constants, varia bles, and functions must be declared or defined be fore they can be used
  - Although this rule permits placing both preprocess or directives and declaration statements througho ut the program, doing so results in poor program s tructure

Empty Parameter List

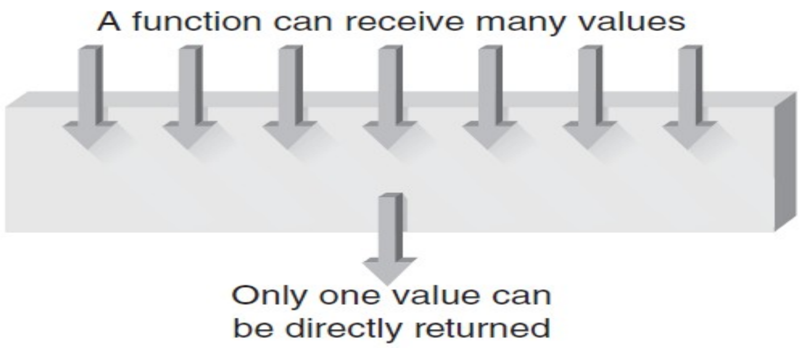
- Although useful functions having an empty param eter list are extremely limited, they can occur
- Function prototype for such a function requires w riting the keyword void or nothing at all between the parentheses following the function’s name
- Examples:

```
int display();
int display(void);
```

Returning a Value

ASTU

- Function receiving an argument passed by value cannot inadvertently alter value stored in the variable used for the argum ent
- Function receiving passed by value argu ments can process the values sent to it in any fashion and return one, and only on e, “legitimate” value directly to the callin g function

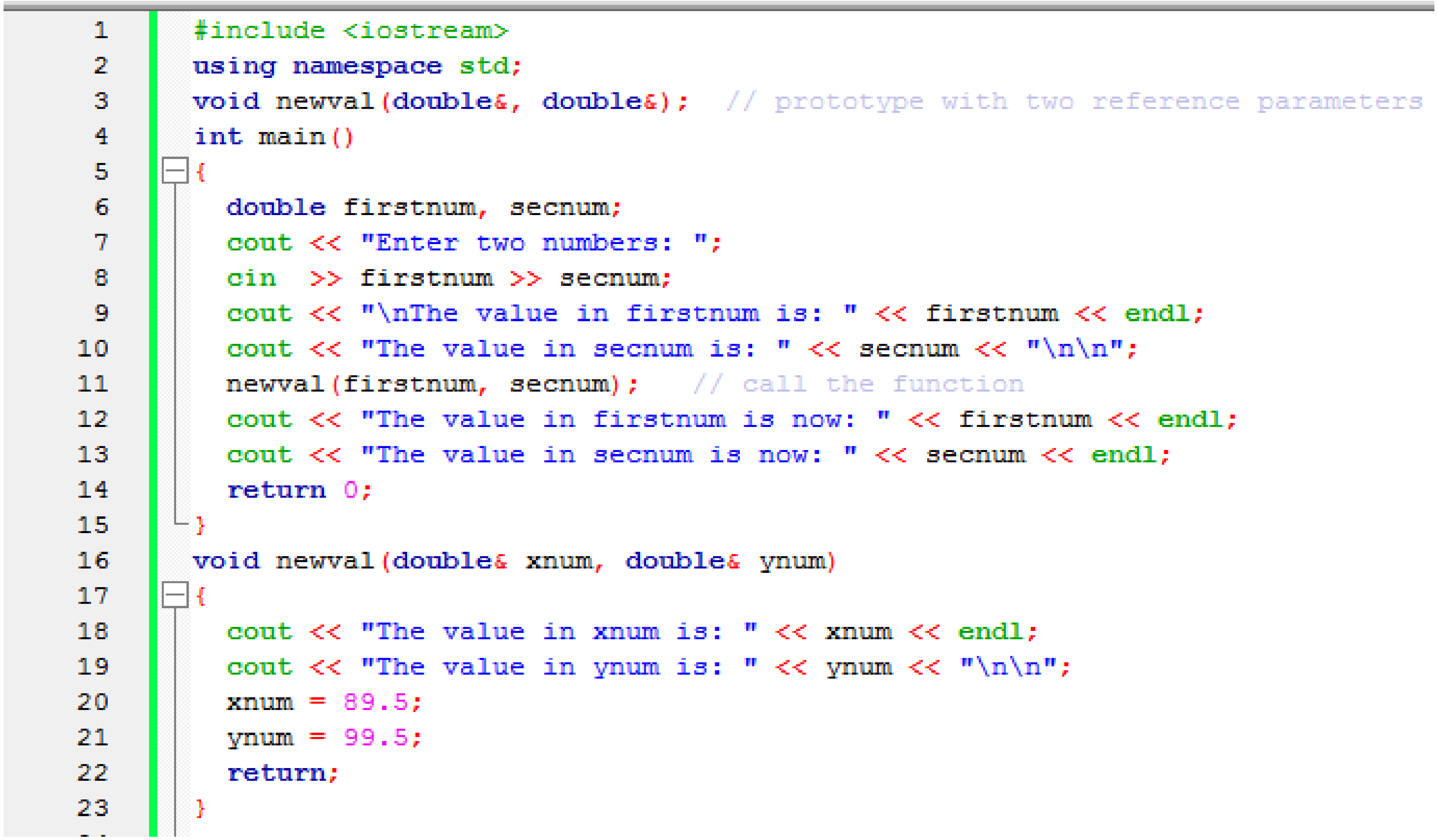

A function can receive many values
Only one value can be directly returned

Pass By Value and Pass By Reference

- In a typical function invocation, called function receives values from its calling function, stores and manipulates the passed values, and directl y returns at most one value
  - **Pass by value**: When data is passed in this manner
  - **Pass by reference**: Giving a called function direct ac cess to its calling function's variables is referred to as
    - The called function can reference, or access, the variable whose address has been passed as a pass by reference a rgument

Pass By Value and Pass By Reference

ASTU

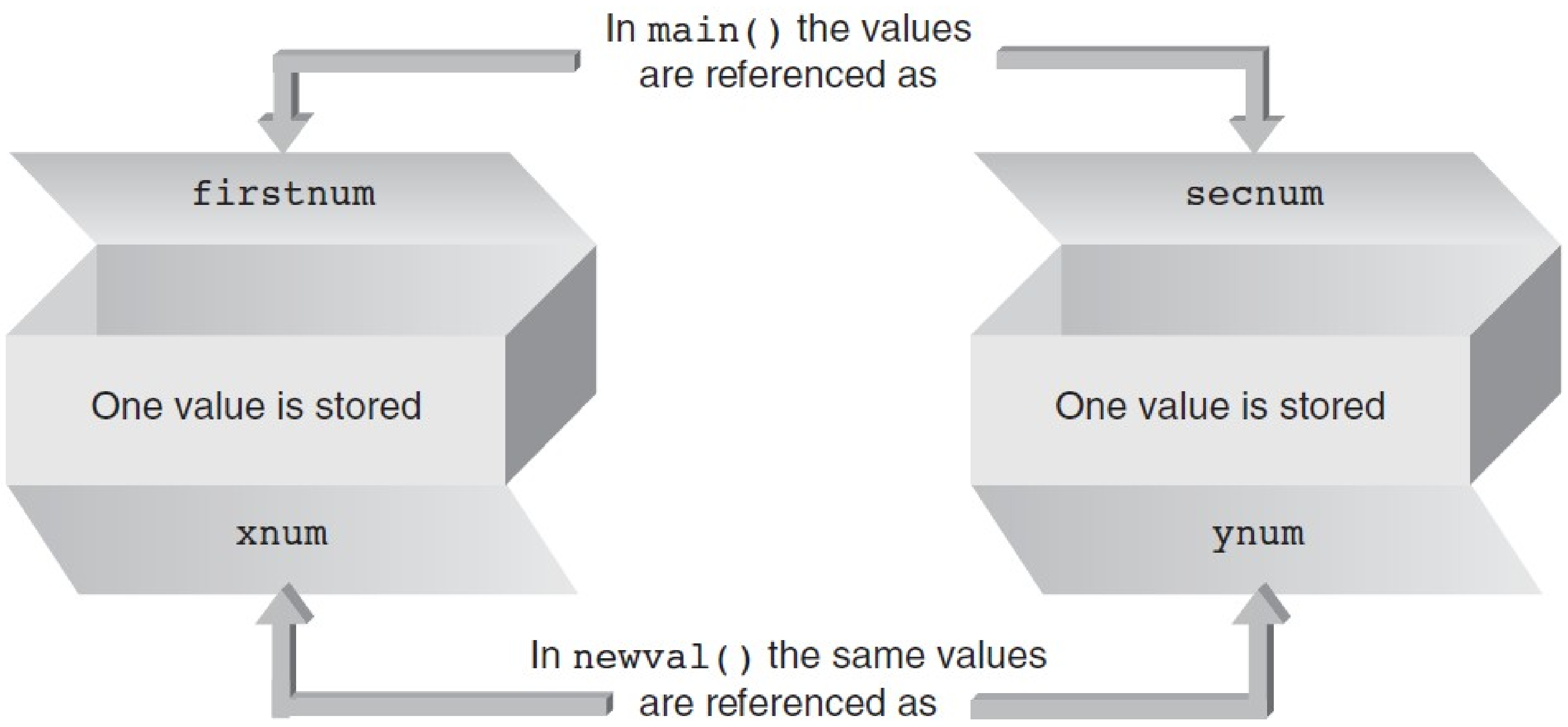


```cpp
#include <iostream>
using namespace std;
void newval(double&, double&);   // prototype with two reference parameters
int main()
{
   double firstnum, secnum;
   cout << "Enter two numbers: ";
   cin  >> firstnum >> secnum;
   cout << "\nThe value in firstnum is: " << firstnum << endl;
   cout << "The value in secnum is: " << secnum << "\n\n";
   newval(firstnum, secnum);    // call the function
   cout << "The value in firstnum is now: " << firstnum << endl;
   cout << "The value in secnum is now: " << secnum << endl;
   return 0;
}
void newval(double& xnum, double& ynum)
{
   cout << "The value in xnum is: " << xnum << endl;
   cout << "The value in ynum is: " << ynum << "\n\n";
   xnum = 89.5;
   ynum = 99.5;
   return;
}
```

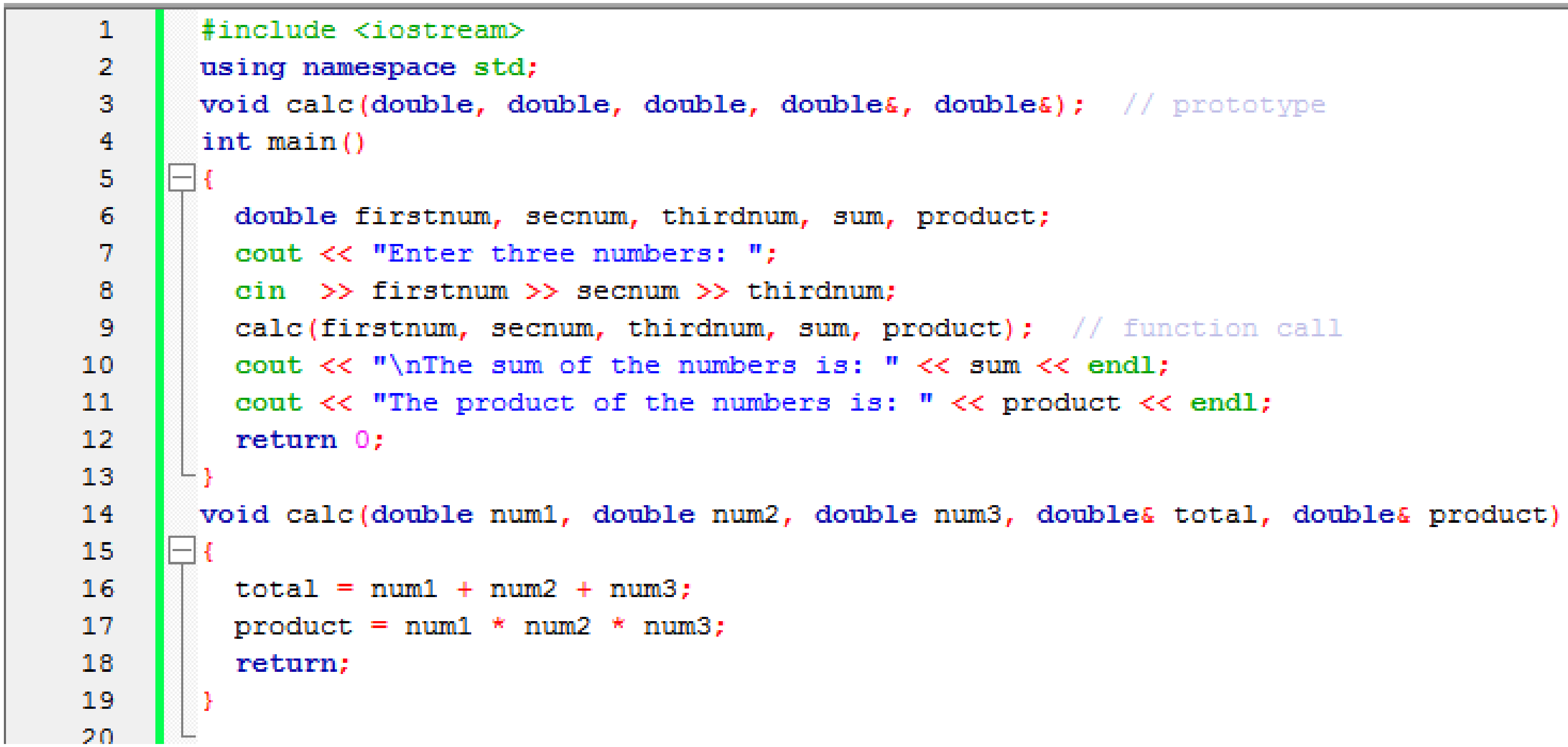
Pass By Value and Pass By Reference

- From the sending side(caller)
  - calling a function and passing an address as an argument that's accepted as a reference parameter is the same as calling a function and passing a value
  - Whether a value or an address is actually pa ssed **depends** on the **parameter** types decla red
- Address operator (**&**) provides the variab le's address

Pass By Value and Pass By Reference

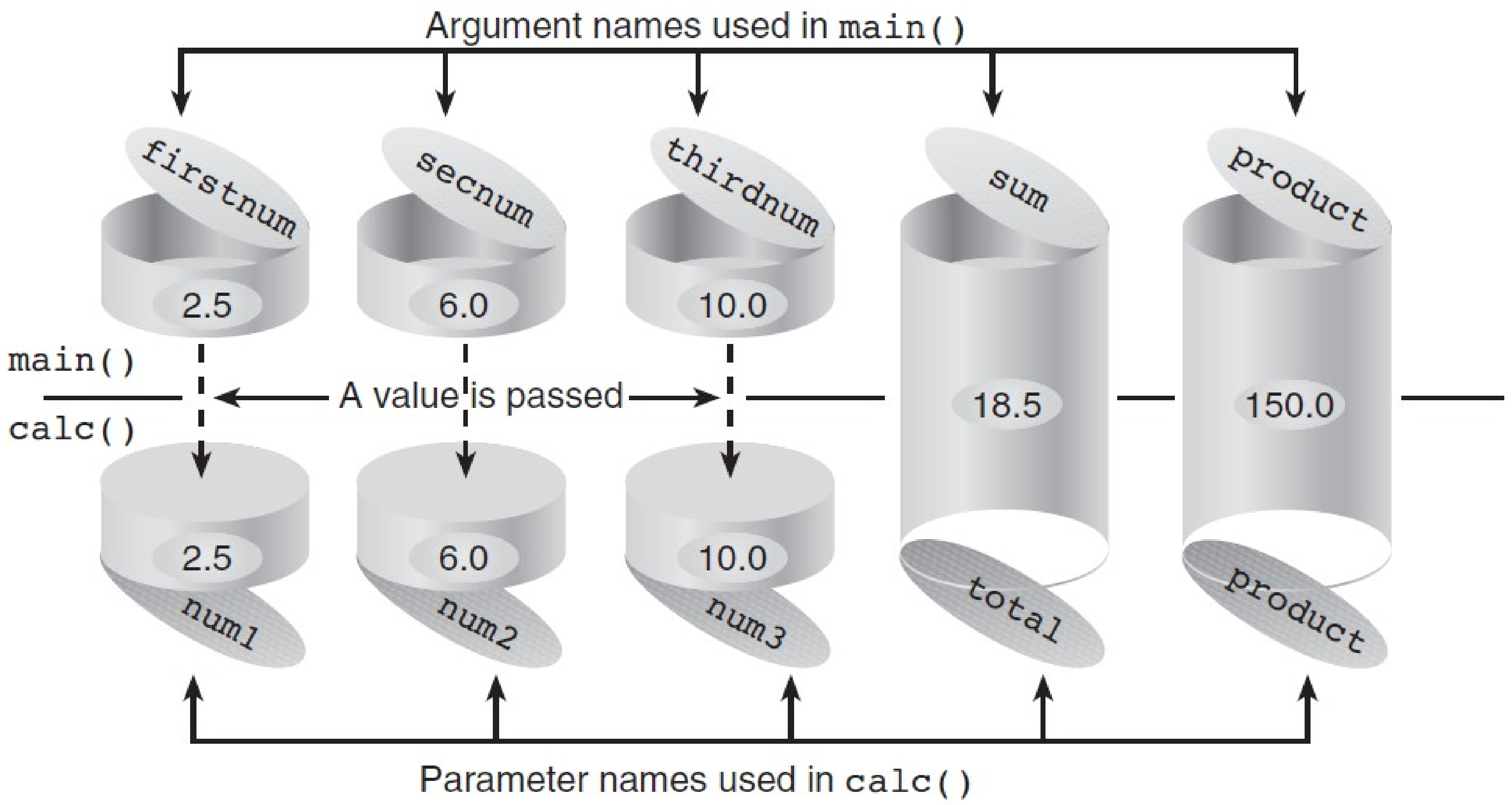

In main() the values are referenced as
firstnum
secnum
One value is stored
One value is stored
xnum
ynum
In newval() the same values are referenced as

Pass By Value and Pass By Reference



```
#include <iostream>
using namespace std;
void calc(double, double, double, double&, double&);  // prototype
int main()
{
  double firstnum, secnum, thirdnum, sum, product;
  cout << "Enter three numbers: ";
  cin  >> firstnum >> secnum >> thirdnum;
  calc(firstnum, secnum, thirdnum, sum, product);  // function call
  cout << "\nThe sum of the numbers is: " << sum << endl;
  cout << "The product of the numbers is: " << product << endl;
  return 0;
}
void calc(double num1, double num2, double num3, double& total, double& product)
{
  total = num1 + num2 + num3;
  product = num1 * num2 * num3;
  return;
}
```

Pass By Value and Pass By Reference



Case Study: Rect. to Polar Coord. Conversion

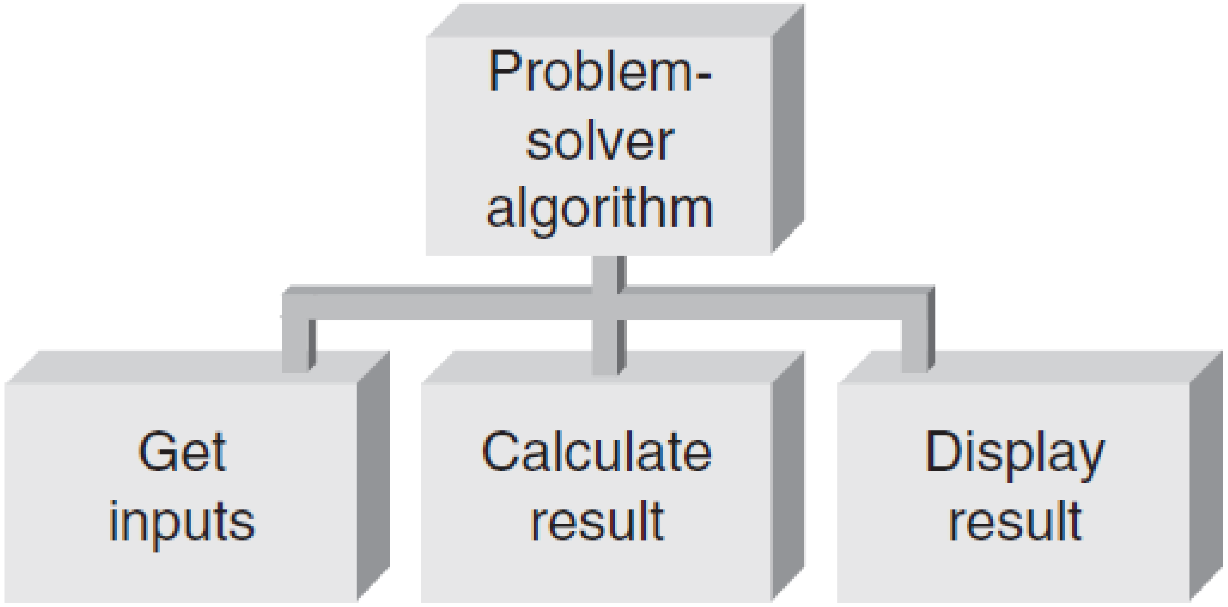
ASTU

- We follow Top Down Design and Ste p-wise refinement
  - This initial outline begins the process of defi ning a more complicated problem as a set o f smaller, more manageable tasks.
  - Each of these tasks can be further subdivid ed or refined into even smaller tasks, if requ ired.
  - After the tasks are well defined, the actual work of coding can begin, starting with any task in any order.

Case Study: Rect. to Polar Coord. Conversion

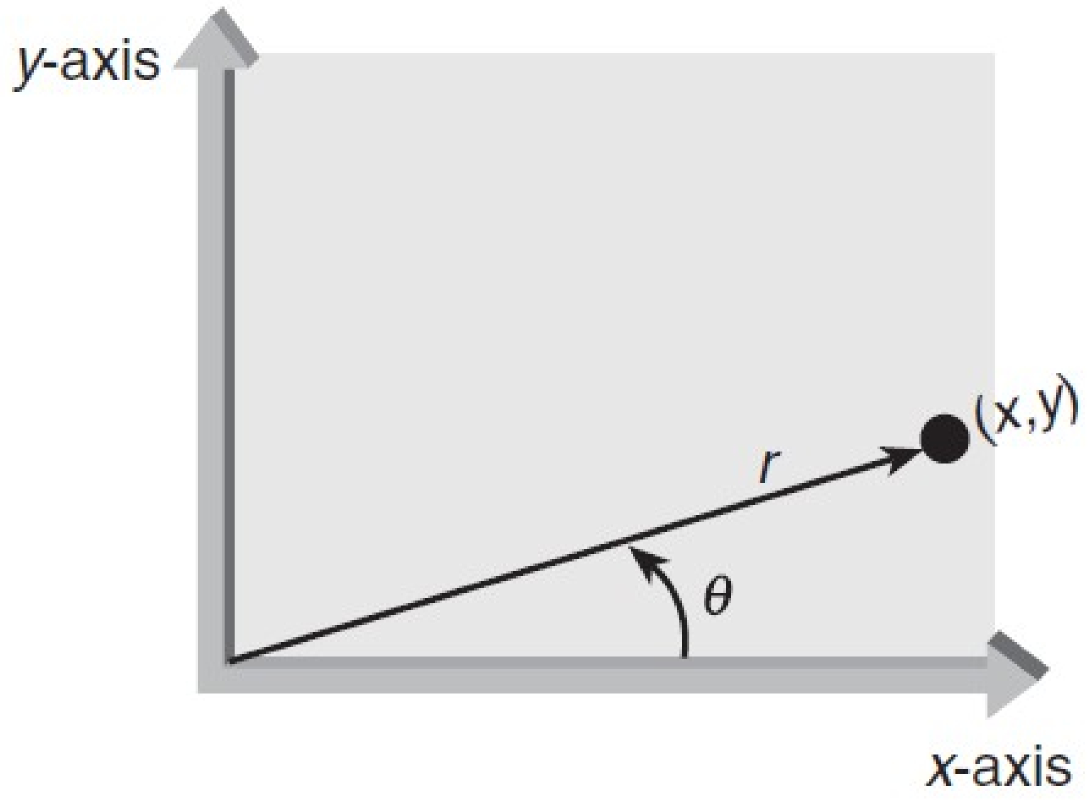
ASTU

- Top Level Algorithm



Step 1: Define a Problem

- Correspondence between polar(distance and angle) and Cartesian(x and y) coordi nate



Step 1: Define a Problem

ASTU

- When the x- and y-coordinates of a point are known, the equivalent r and θ coordinates can be calculated by using these formulas

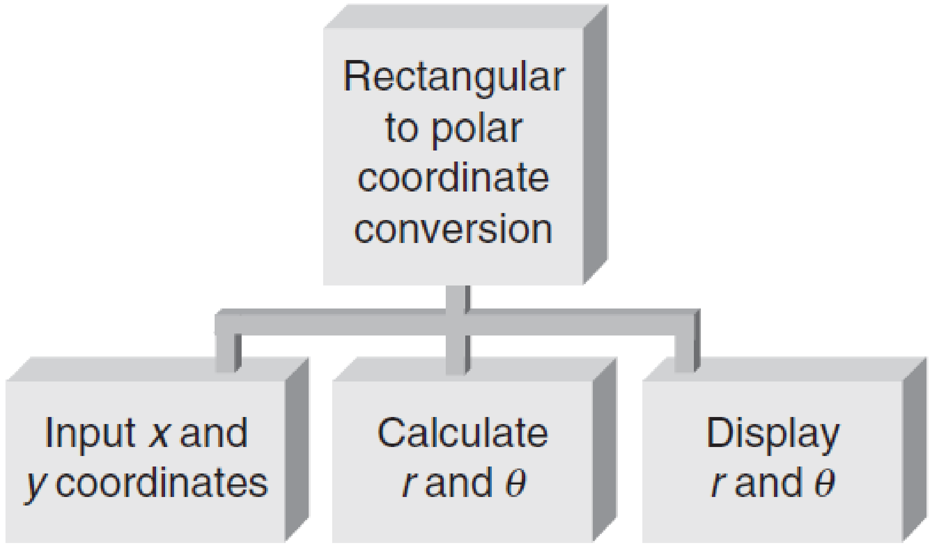
$$r = \sqrt{x^2 + y^2}$$

$$\theta = \tan^{-1}(y/x); x \neq 0$$

Step 1: Define a Problem

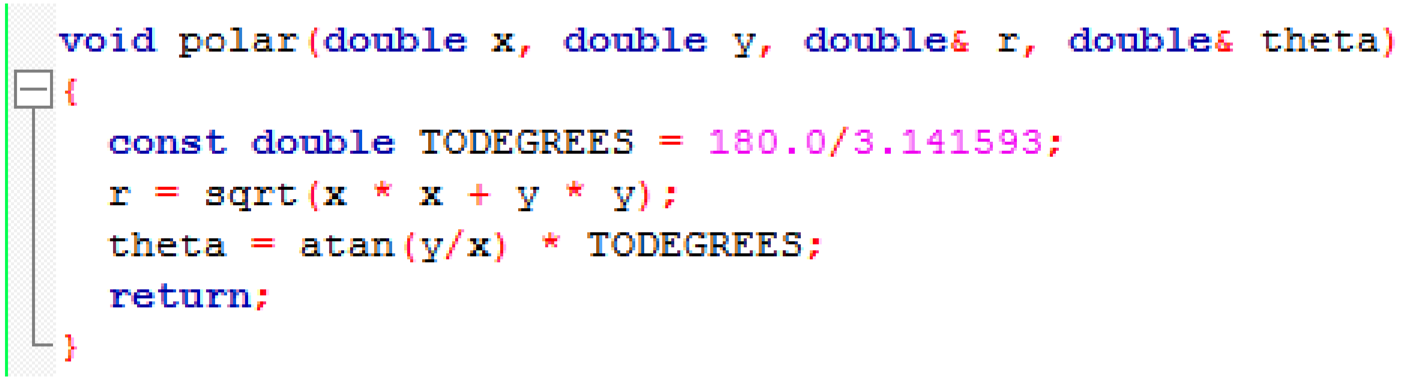
ASTU

- Top Level Structure(Initial Pseudo Code)
  - Get the x- and y-coordinate values
  - Calculate the polar (r and θ) coordinate values
  - Display the polar coordinate values

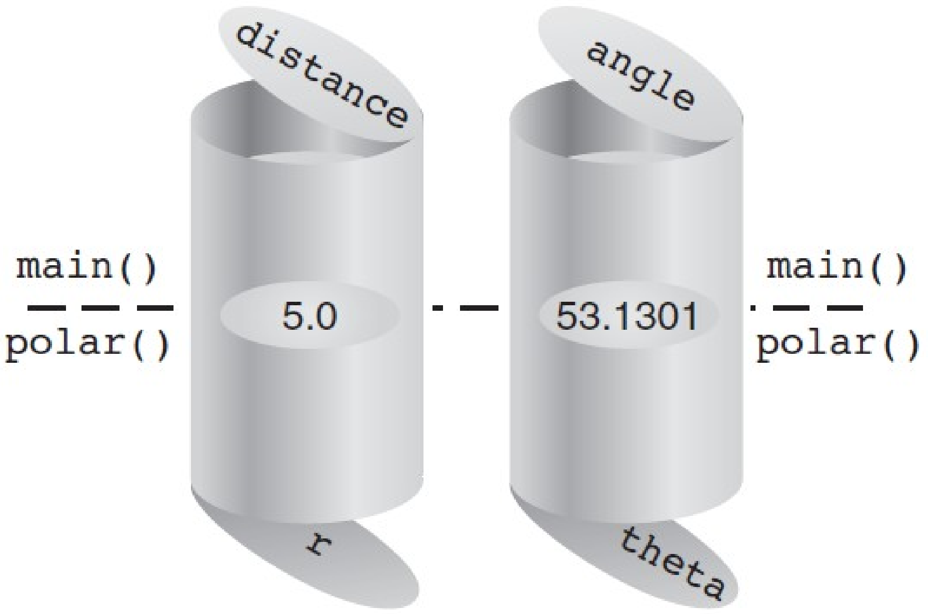

Rectangular to polar coordinate conversion
Input x and y coordinates
Calculate r and θ
Display r and θ

Step 1: Define a Problem

- Initial Coding



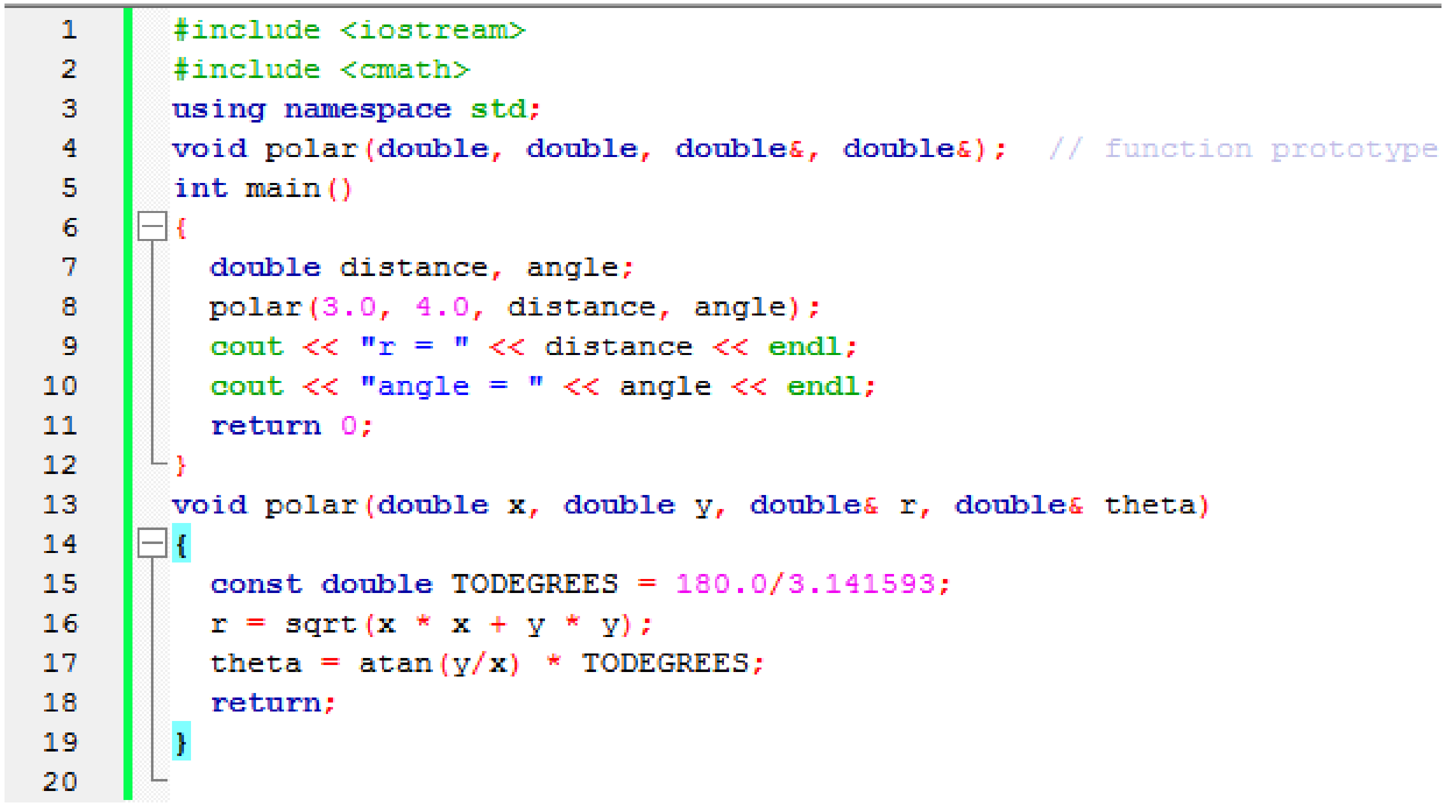
```
void polar(double x, double y, double& r, double& theta)
{
  const double TODEGREES = 180.0/3.141593;
  r = sqrt(x * x + y * y);
  theta = atan(y/x) * TODEGREES;
  return;
}
```



Step 2: Test the Function

ASTU

- After polar() is written, it can be tested i ndependently of any other function.



```cpp
#include <iostream>
#include <cmath>
using namespace std;
void polar(double, double, double&, double&);  // function prototype
int main()
{
  double distance, angle;
  polar(3.0, 4.0, distance, angle);
  cout << "r = " << distance << endl;
  cout << "angle = " << angle << endl;
  return 0;
}
void polar(double x, double y, double& r, double& theta)
{
  const double TODEGREES = 180.0/3.141593;
  r = sqrt(x * x + y * y);
  theta = atan(y/x) * TODEGREES;
  return;
}
```

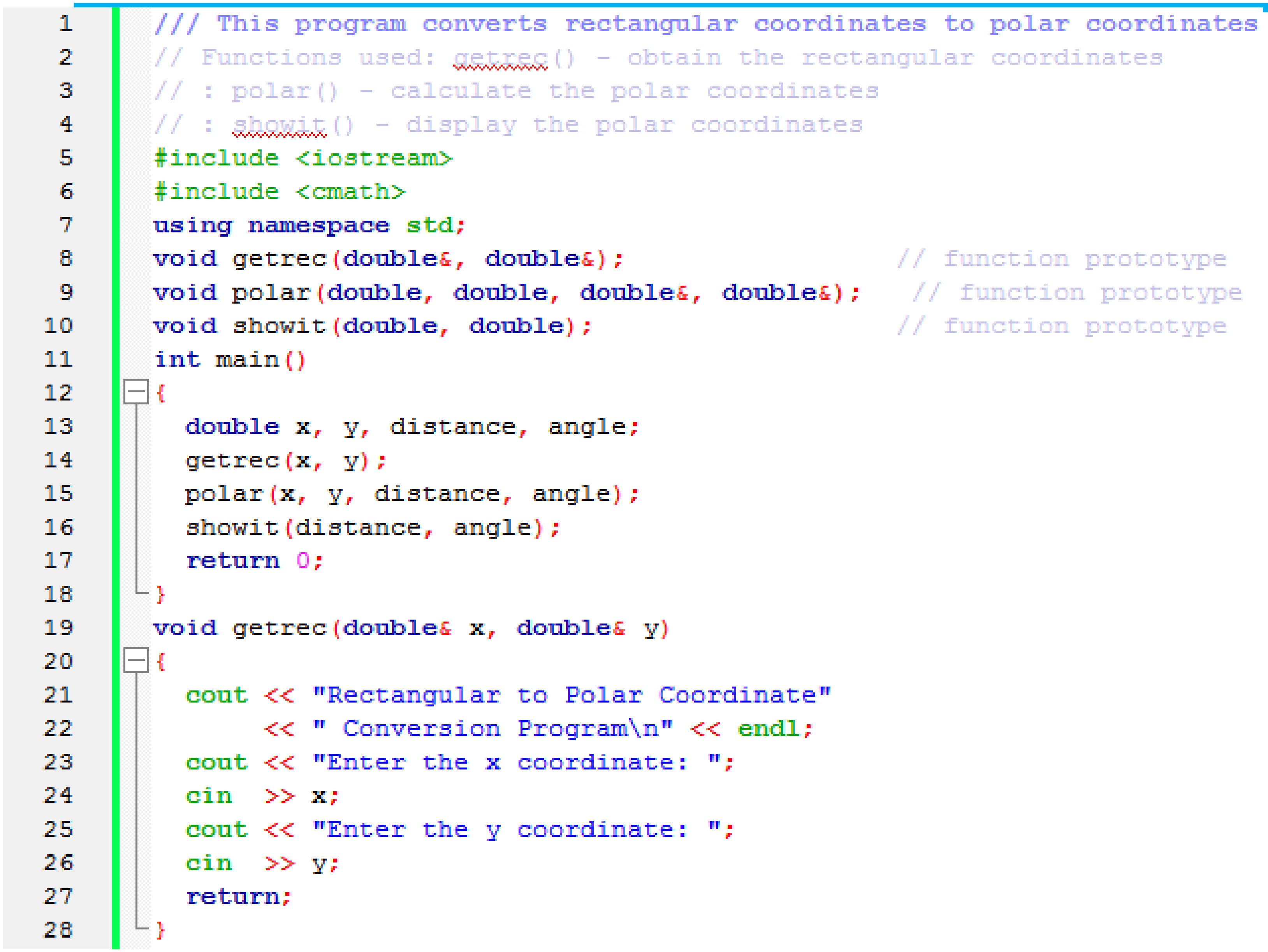
```
r=5
angle=53.1301
```

Step 3: Complete the Program

- The program also requires
  - writing functions for accepting two rectang ular coordinates
    - The following function, **getrec()** , can be used t o accept the input data
    - Test it
  - displaying the calculated polar coordinates
    - The function **showit()** for displaying polar coord inates is constructed in a similar manner.
    - Test it

The Complete Program

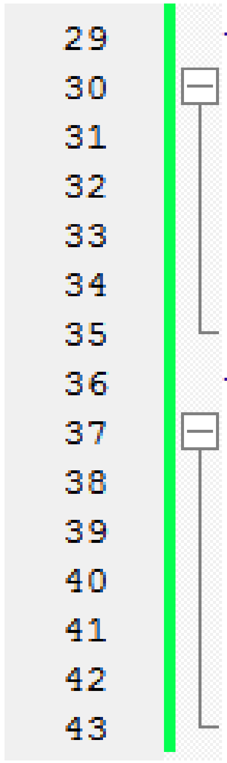
ASTU



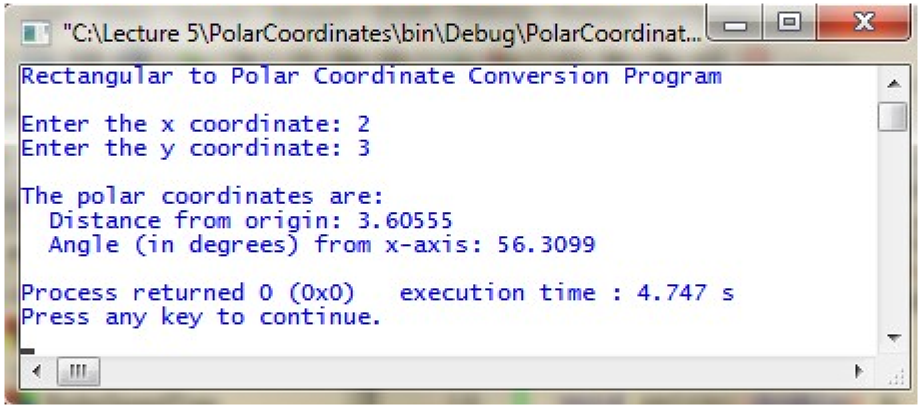
```cpp
/// This program converts rectangular coordinates to polar coordinates
// Functions used: getrec() - obtain the rectangular coordinates
// : polar() - calculate the polar coordinates
// : showit() - display the polar coordinates
#include <iostream>
#include <cmath>
using namespace std;
void getrec(double&, double&);                          // function prototype
void polar(double, double, double&, double&);           // function prototype
void showit(double, double);                            // function prototype
int main()
{
    double x, y, distance, angle;
    getrec(x, y);
    polar(x, y, distance, angle);
    showit(distance, angle);
    return 0;
}
void getrec(double& x, double& y)
{
    cout << "Rectangular to Polar Coordinate"
         << " Conversion Program\n" << endl;
    cout << "Enter the x coordinate: ";
    cin  >> x;
    cout << "Enter the y coordinate: ";
    cin  >> y;
    return;
}
```

The Complete Program

ASTU



```
void polar(double x, double y, double& r, double& theta)
{
  const double TODEGREES = 180.0/3.141593;
  r = sqrt(x * x + y * y);
  theta = atan(y/x) * TODEGREES;
  return;
}
void showit(double radius, double angle)
{
  cout << "\nThe polar coordinates are: " << endl;
  cout << "  Distance from origin: " << radius << endl;
  cout << "  Angle (in degrees) from x-axis: " << angle << endl;
  return;
}
```



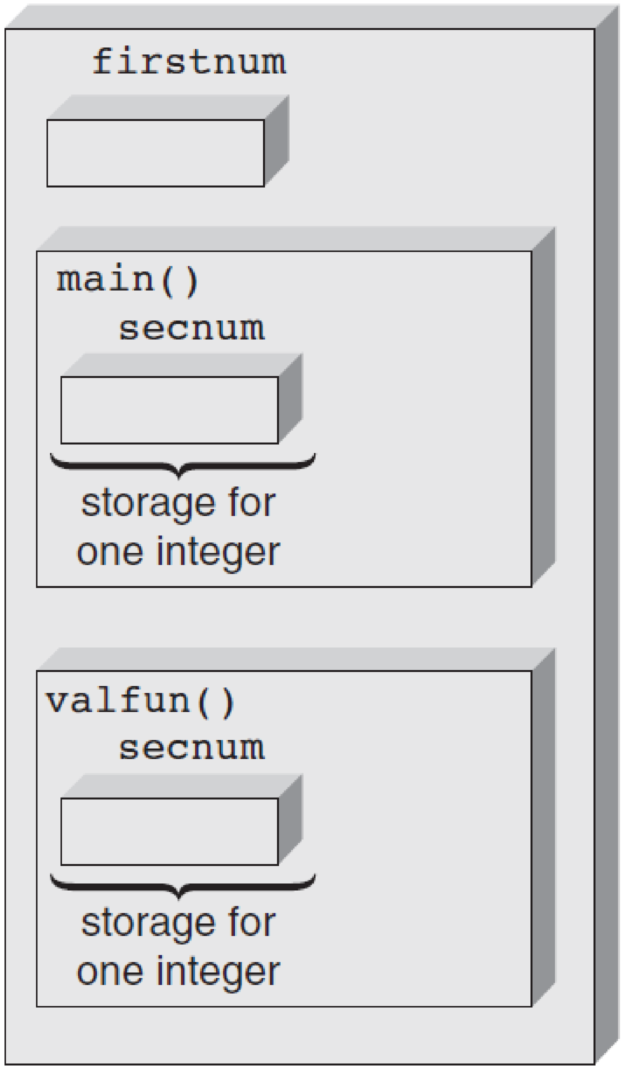
"C:\Lecture 5\PolarCoordinates\bin\Debug\PolarCoordinat...

```
Rectangular to Polar Coordinate Conversion Program

Enter the x coordinate: 2
Enter the y coordinate: 3

The polar coordinates are:
  Distance from origin: 3.60555
  Angle (in degrees) from x-axis: 56.3099

Process returned 0 (0x0)   execution time : 4.747 s
Press any key to continue.
```

Variable Scope

- **Local variables:** Variables created in a function that are conventionally available only to the function
- **Scope:** Section of the program where the identifier is valid or “known”
- A variable with **local scope** is simply one with storage locations set aside for it by a declaration statement inside the function that declared them
- A variable with **global scope** has storage created for it by a declaration statement located outside any function

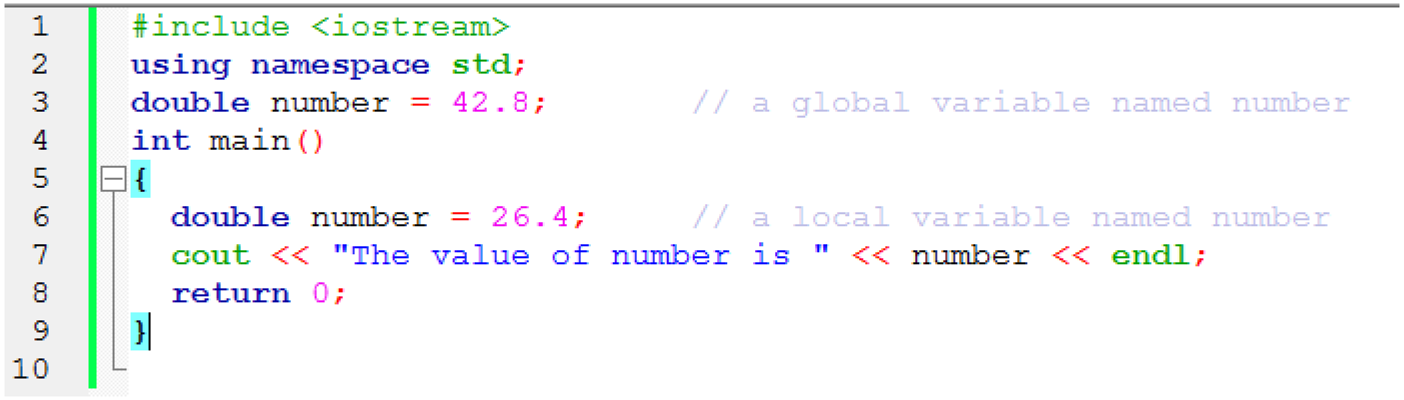
Variable Scope



Variable Scope

ASTU

- When a local variable has the same name as a global variable, all references to the variable name made within the local variable's scope refer to the local variable

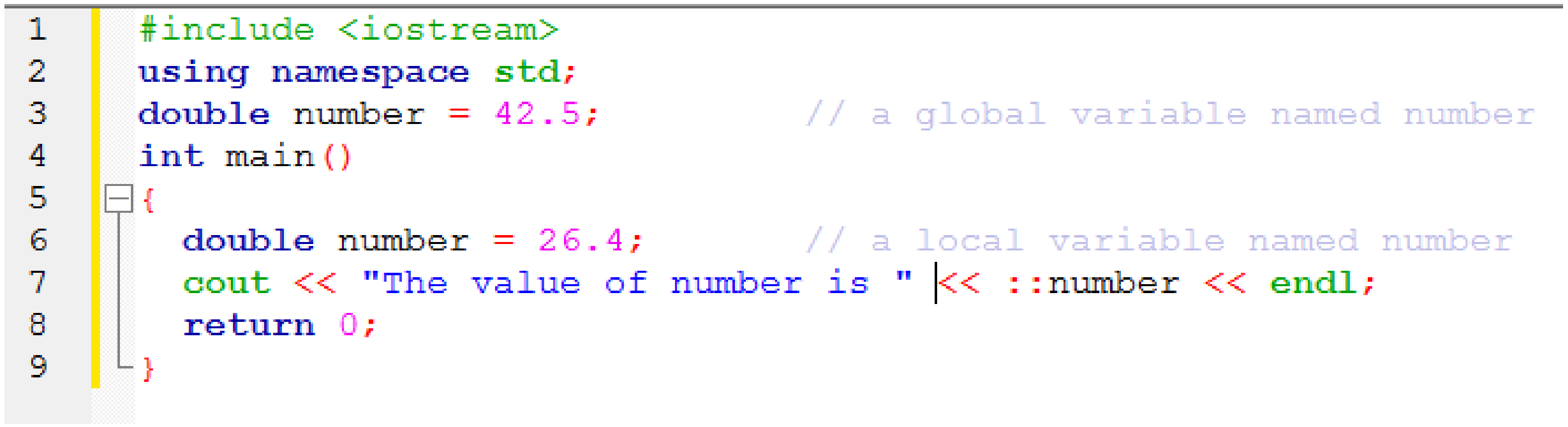


```cpp
#include <iostream>
using namespace std;
double number = 42.8;        // a global variable named number
int main()
{
   double number = 26.4;     // a local variable named number
   cout << "The value of number is " << number << endl;
   return 0;
}
```

Scope Resolution Operator

ASTU

- To reference a global variable when a loc al variable of the same name is in scope, use C++'s scope resolution operator, whi ch is ::



```
#include <iostream>
using namespace std;
double number = 42.5;           // a global variable named number
int main()
{
   double number = 26.4;        // a local variable named number
   cout << "The value of number is " << ::number << endl;
   return 0;
}
```

Misuse of Globals

- Global variables allow programmers to “jump around” the normal safeguards provided by functions
- Instead of passing variables to a function, it is possible to make all variables global: **do not do this**
  - Indiscriminate use of global variables destroys the safeguards C++ provides to make functions independent and insulated from each other
  - Using only global variables can be especially disastrous in large programs with many user-created functions

Lifetime of Variables

ASTU

- A variable’s scope can be thought of as the space in t he program where the variable is valid
- In addition to space dimension represented by scope, variables have a time dimension that refers to the len gth of time storage locations are reserved for a variab le
- This time, dimension is referred to as the variable’s lif etime
- When and how long a variable’s storage locations are kept before they are released can be determined by t he variable’s storage category

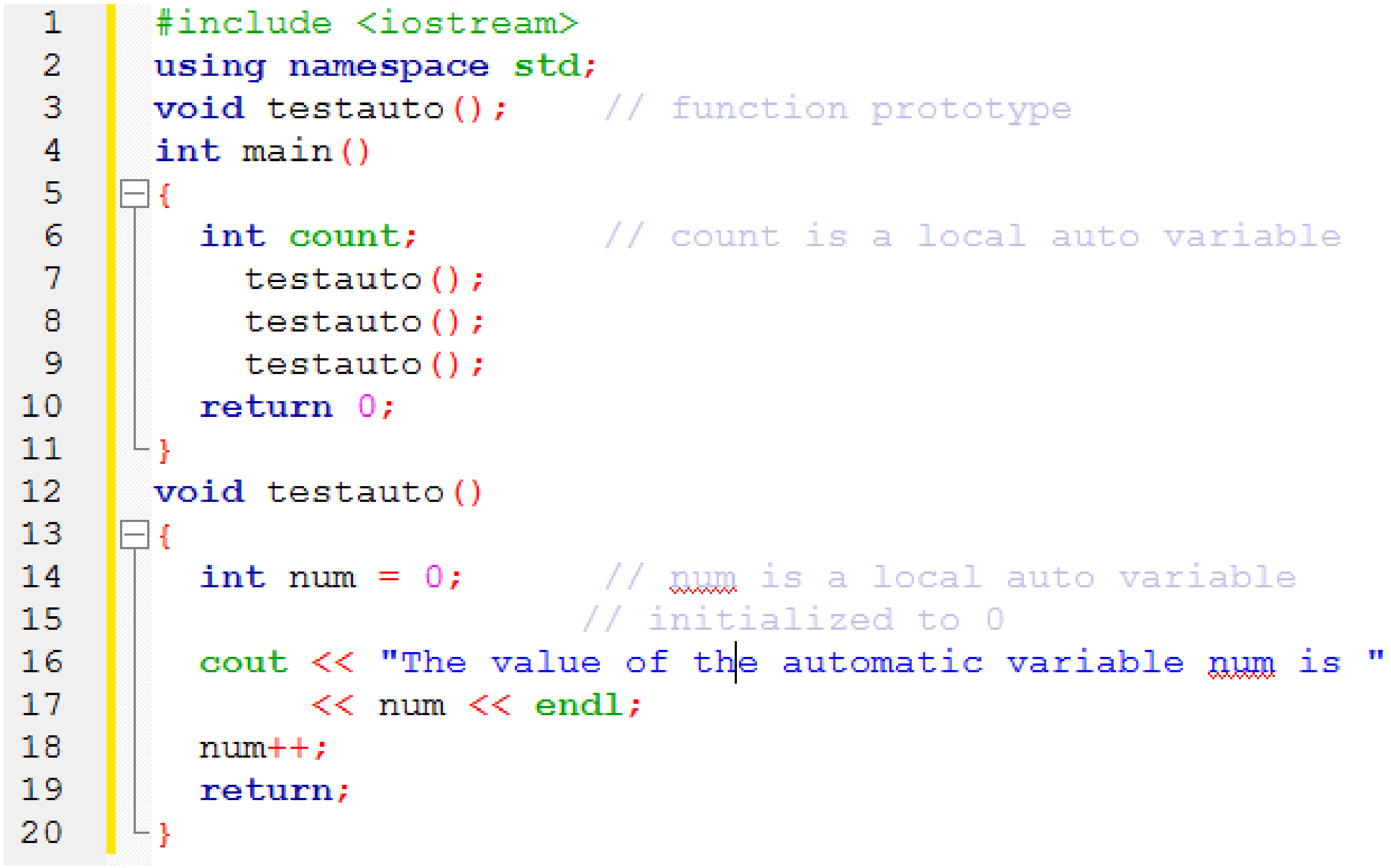
Variable Storage Categories

- The four available storage categories are :
  - auto
  - static
  - extern
  - register

Local Variable Storage Categories

- Local variables can be members only of the auto, st atic, or register storage categories
- Storage for automatic local variables is reserved or created automatically each time a function is called
  - As long as the function hasn't returned control to its call ing function, all automatic variables local to the functio n are "alive"
- A local static variable isn't created and destroyed ea ch time the function declaring it is called
  - Local static variables remain in existence for the progra m's lifetime

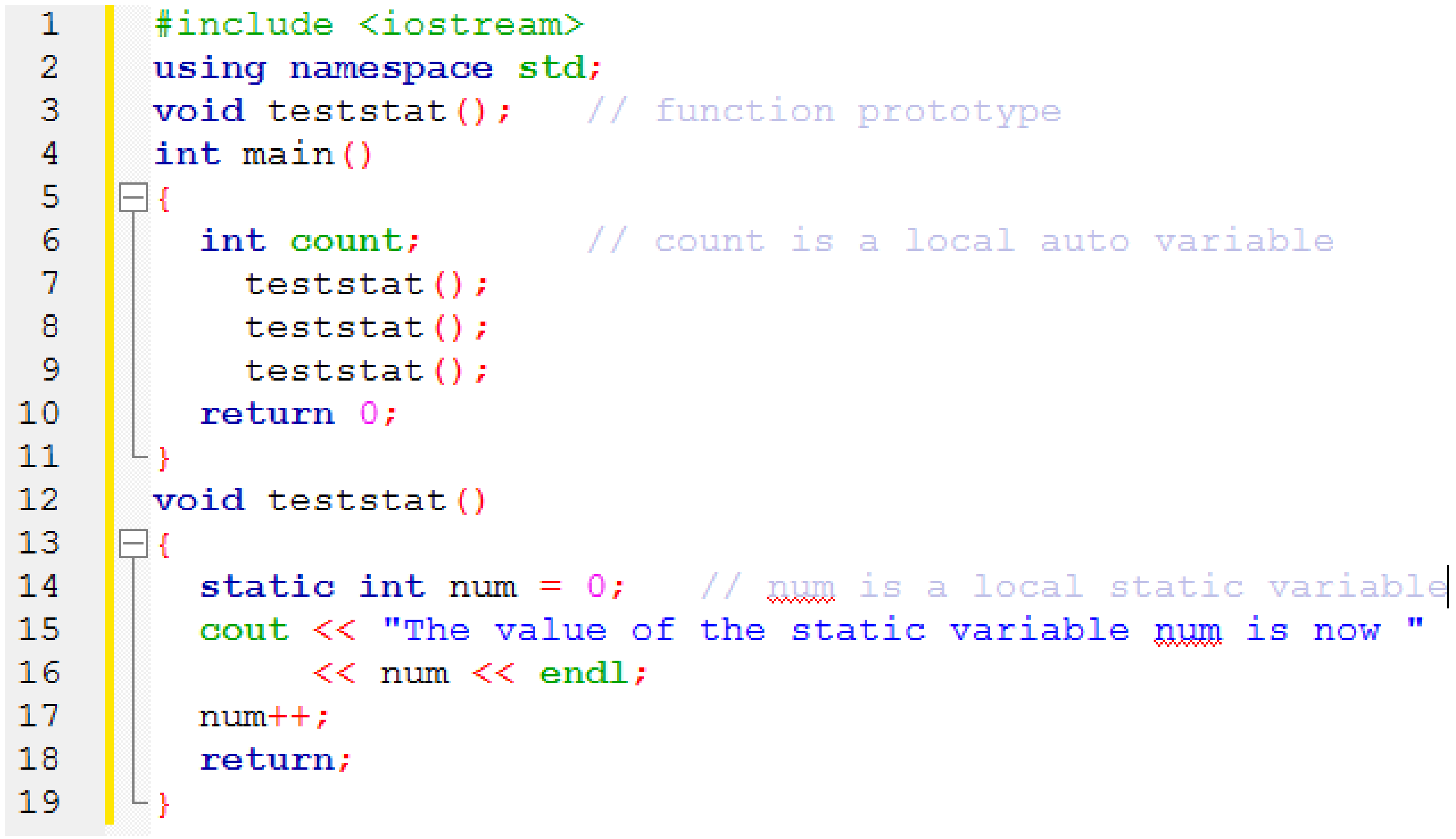
Local Automatic Variable



```
#include <iostream>
using namespace std;
void testauto();    // function prototype
int main()
{
  int count;        // count is a local auto variable
    testauto();
    testauto();
    testauto();
  return 0;
}
void testauto()
{
  int num = 0;      // num is a local auto variable
                   // initialized to 0
  cout << "The value of the automatic variable num is "
       << num << endl;
  num++;
  return;
}
```

Local Static Variable

ASTU



```
#include <iostream>
using namespace std;
void teststat();    // function prototype
int main()
{
  int count;        // count is a local auto variable
    teststat();
    teststat();
    teststat();
  return 0;
}
void teststat()
{
  static int num = 0;    // num is a local static variable
  cout << "The value of the static variable num is now "
       << num << endl;
  num++;
  return;
}
```

Global Variable Storage Categories

- Global variables are created by definition stat ements external to a function
- By their nature, global variables do not come and go with the calling of a function
- After a global variable is created, it exists unt il the program in which it’s declared has finis hed executing
- Global variables can be declared with the sta tic or extern storage category, but not both