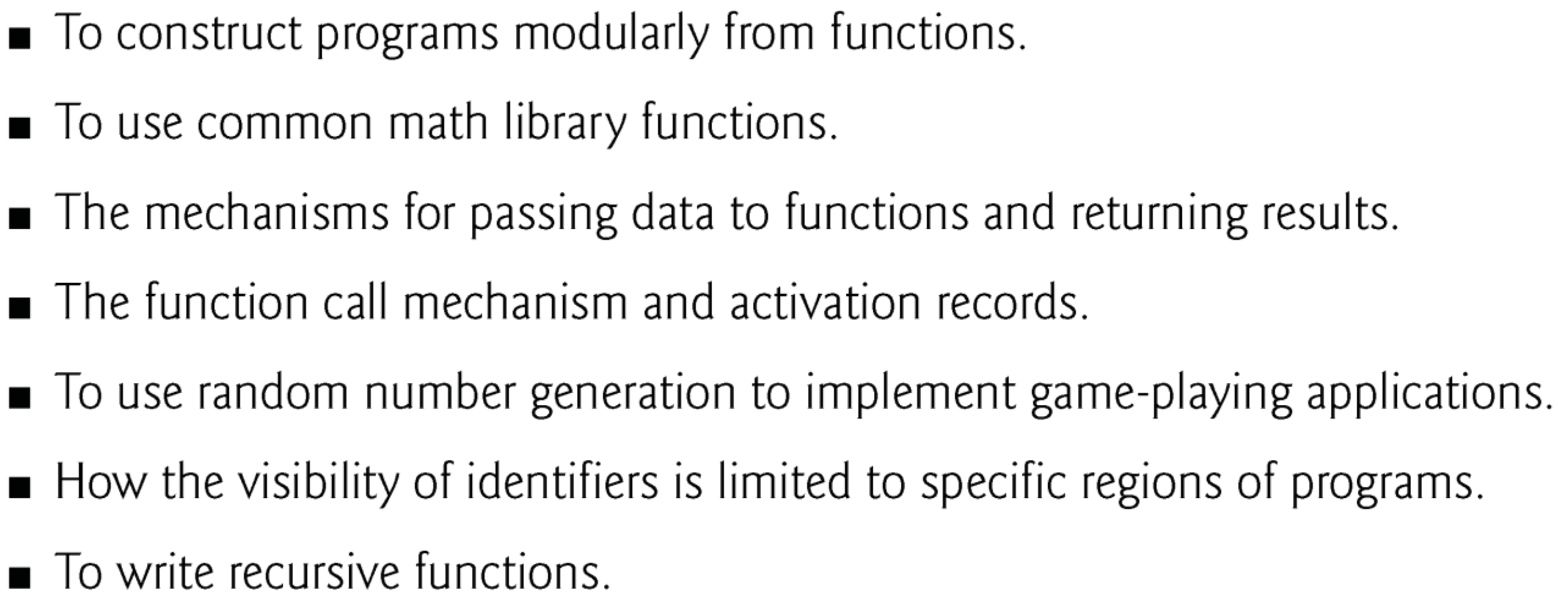
Lecture 6: Functions and Recursion – Part I

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Learning Objective



- To construct programs modularly from functions.
- To use common math library functions.
- The mechanisms for passing data to functions and returning results.
- The function call mechanism and activation records.
- To use random number generation to implement game-playing applications.
- How the visibility of identifiers is limited to specific regions of programs.
- To write recursive functions.

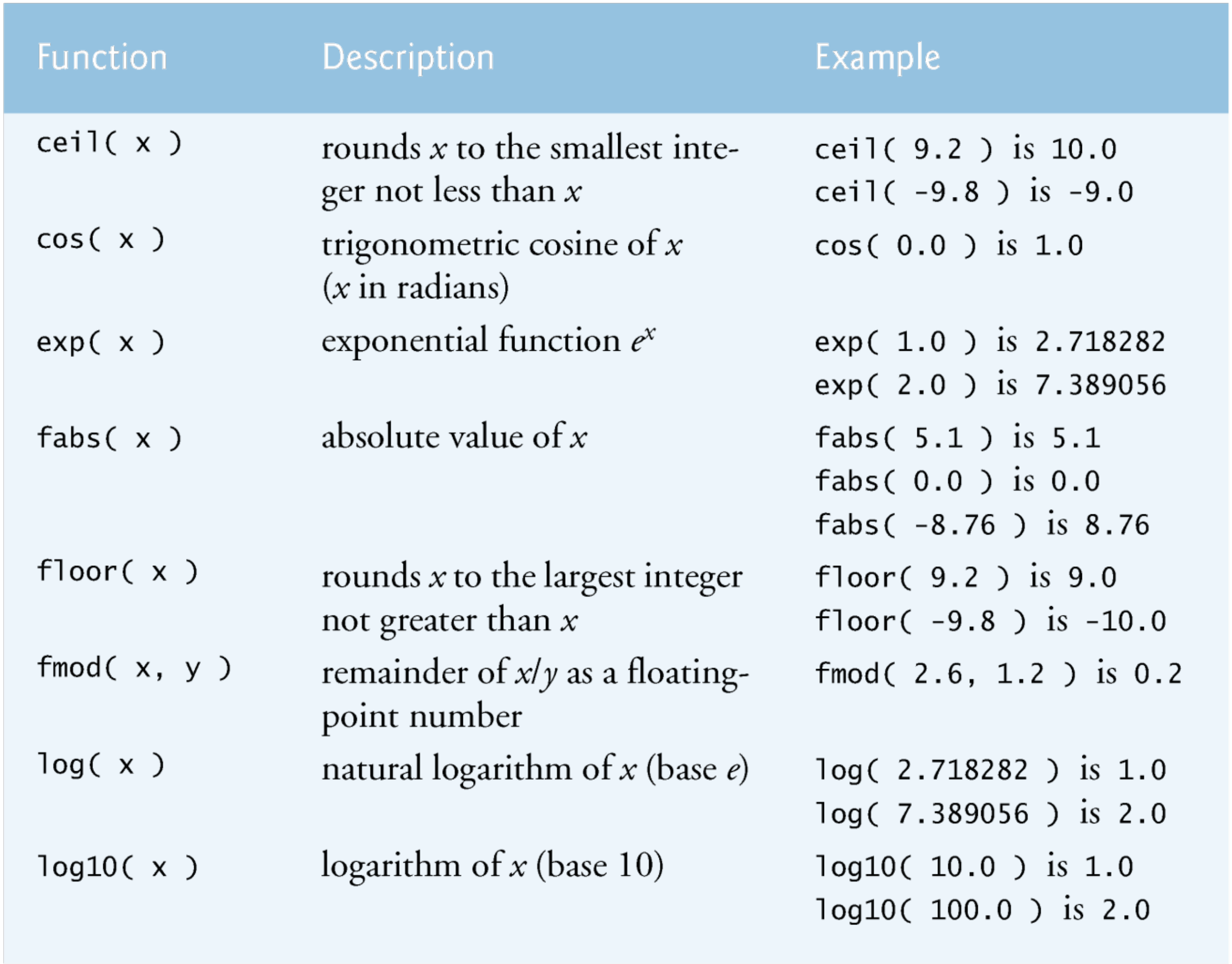
Introduction

- ❑ **Construct programs from small, simple pieces, or components.**
  - ❑ divide and conquer
- ❑ **We emphasize how to declare and use functions to facilitate the design, implementation, operation and maintenance of large programs.**
  - ❑ You’ll learn how to declare your own functions.
- ❑ **We’ll discuss function prototypes and how the compiler uses them to ensure that functions are called properly.**
- ❑ **We’ll take a brief diversion into simulation techniques with random number generation**

Introduction

- **The C++ Standard Library provides a rich collection of functions for**
  - common mathematical calculations,
  - string manipulations,
  - character manipulations,
  - input/output,
  - error checking and
  - many other useful operations.

Math Functions in <cmath>



Function	Description	Example

`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 )` is `10.0`
		`ceil( -9.8 )` is `-9.0`

`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 )` is `1.0`

`exp( x )`	exponential function e^x	`exp( 1.0 )` is `2.718282`
		`exp( 2.0 )` is `7.389056`

`fabs( x )`	absolute value of x	`fabs( 5.1 )` is `5.1`
		`fabs( 0.0 )` is `0.0`
		`fabs( -8.76 )` is `8.76`

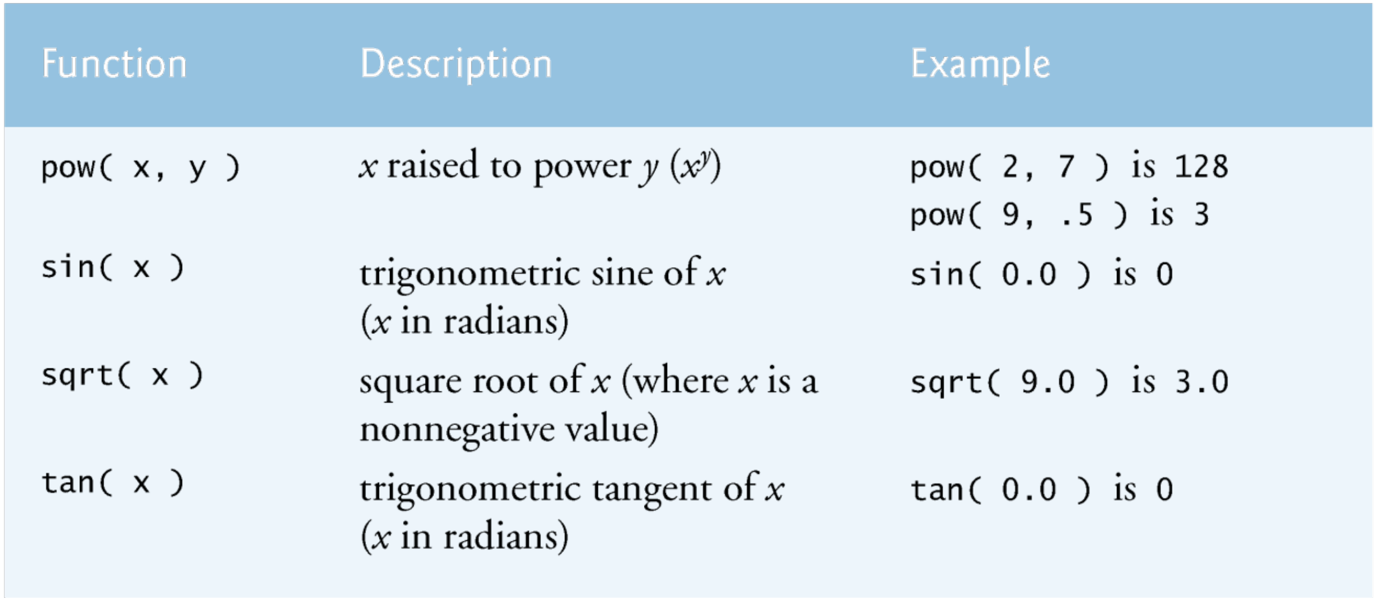
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 )` is `9.0`
		`floor( -9.8 )` is `-10.0`

`fmod( x, y )`	remainder of x/y as a floating-point number	`fmod( 2.6, 1.2 )` is `0.2`

`log( x )`	natural logarithm of x (base e)	`log( 2.718282 )` is `1.0`
		`log( 7.389056 )` is `2.0`

`log10( x )`	logarithm of x (base 10)	`log10( 10.0 )` is `1.0`
		`log10( 100.0 )` is `2.0`

Math Functions in <cmath>



Function	Description	Example

`pow( x, y )`	*x* raised to power *y* (x^y)	`pow( 2, 7 )` is `128`
		`pow( 9, .5 )` is `3`

`sin( x )`	trigonometric sine of *x* (*x* in radians)	`sin( 0.0 )` is `0`
`sqrt( x )`	square root of *x* (where *x* is a nonnegative value)	`sqrt( 9.0 )` is `3.0`
`tan( x )`	trigonometric tangent of *x* (*x* in radians)	`tan( 0.0 )` is `0`

Introduction

- ❑ **Functions you write are referred to as user-defined functions or programmer-defined functions.**
- ❑ **Motivations for "functionalizing" a program.**
  - ❑ Divide-and-conquer makes program development more manageable.
  - ❑ Software reusability—using existing functions as building blocks to create new programs.
    - Programs can be created from standardized functions that accomplish specific tasks.
  - ❑ Avoid repeating code in a program.
    - Packaging code as a function allows the code to be executed from different locations in a program simply by calling the function.

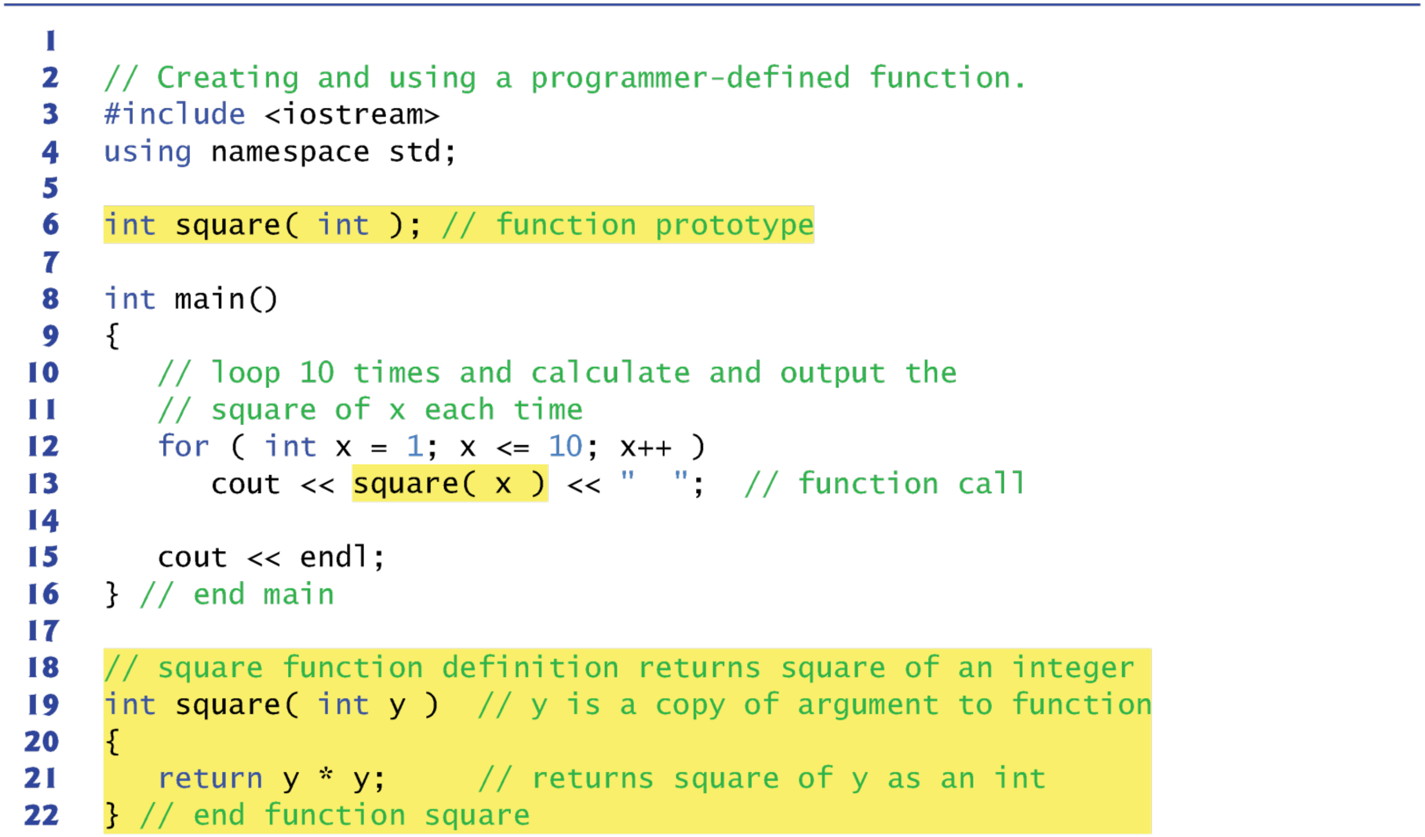
Function Definition

- ❑ **The format of a function definition is as follows:**

```
return-value-type  function-name(  parameter-list  )
{
     declarations and statements
}
```

- ❑ **The *function-name* is any valid identifier.**
- ❑ **The *return-value-type* is the data type of the returned result to the caller.**
  - ❑ The type `void` indicates that a function does not return a value.
- ❑ **All variables defined in a function are local variables—they're known only in the function in which they're defined.**
- ❑ **Most functions have a list of parameters that provide the means for communicating information between functions.**
  - ❑ A function's parameters are also local variables of that function.

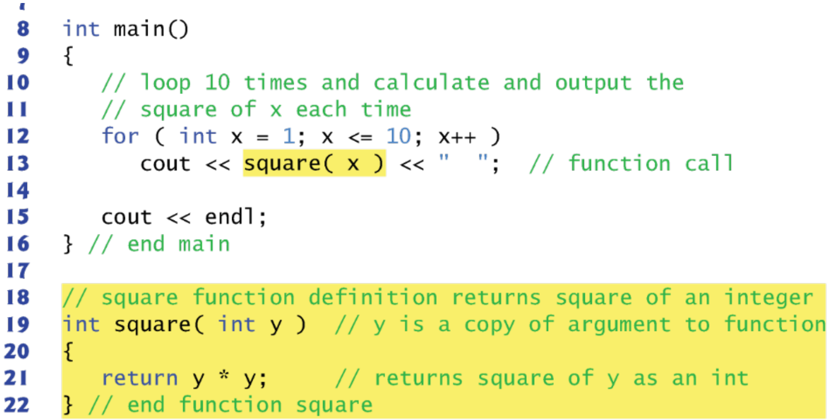
Example



```cpp

// Creating and using a programmer-defined function.
#include <iostream>
using namespace std;

int square( int ); // function prototype

int main()
{
   // loop 10 times and calculate and output the
   // square of x each time
   for ( int x = 1; x <= 10; x++ )
      cout << square( x ) << "  ";  // function call

   cout << endl;
} // end main

// square function definition returns square of an integer
int square( int y )  // y is a copy of argument to function
{
   return y * y;     // returns square of y as an int
} // end function square
```

Debrief

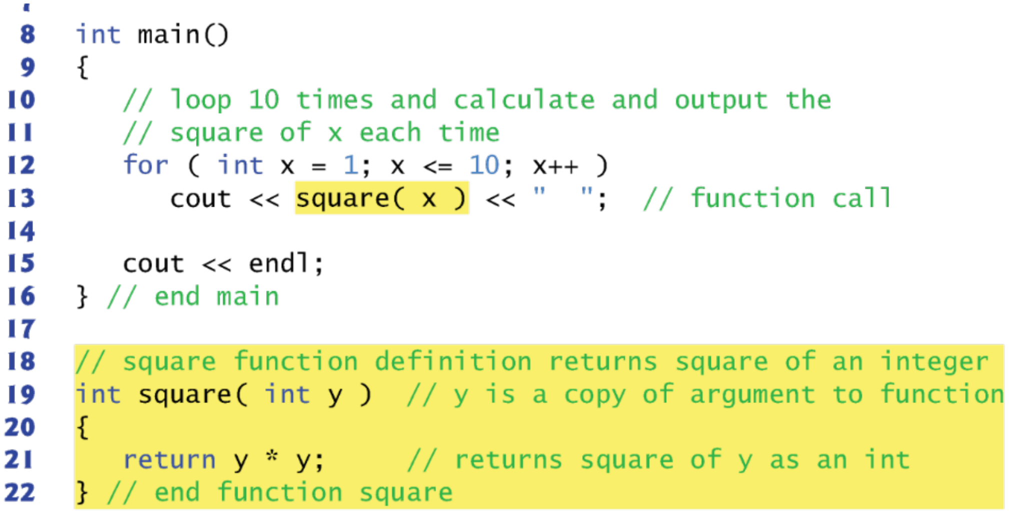
- Function `square` is invoked or called in `main` with the expression `square( x )` in line 13.
- The parentheses `()` in the function call are an operator in C++ that causes the function to be called.
- Function `square` (lines 19–22) receives a copy of the value of argument `x` from line 13 and stores it in the parameter `y`.
- Then `square` calculates `y * y` (line 21) and passes the result back to the point in `main` where `square` was invoked (line 13).
- The result is displayed.
- The function call does not change the value of `x`.
- The `for` repetition structure repeats this process for each of the values 1 through 10.



```
 8  int main()
 9  {
10     // loop 10 times and calculate and output the
11     // square of x each time
12     for ( int x = 1; x <= 10; x++ )
13        cout << square( x ) << "  ";  // function call
14
15     cout << endl;
16  } // end main
17
18  // square function definition returns square of an integer
19  int square( int y )  // y is a copy of argument to function
20  {
21     return y * y;      // returns square of y as an int
22  } // end function square
```

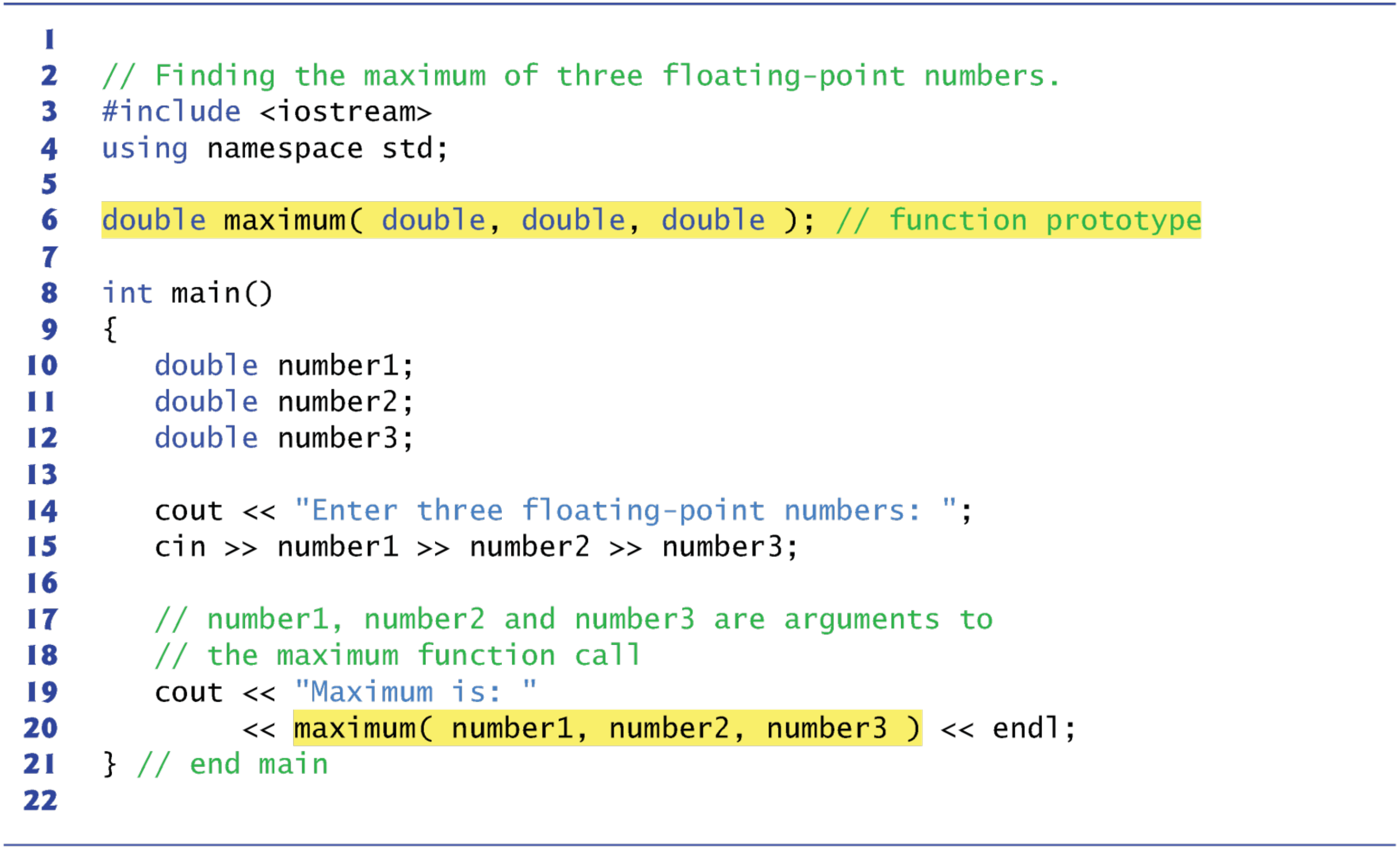
Debrief

- The definition of `square` (lines 19–22) shows that it uses integer parameter `y`.
- Keyword `int` preceding the function name indicates that `square` returns an integer result.
- The `return` statement in `square` (line 21) passes the result of the calculation back to the calling function.



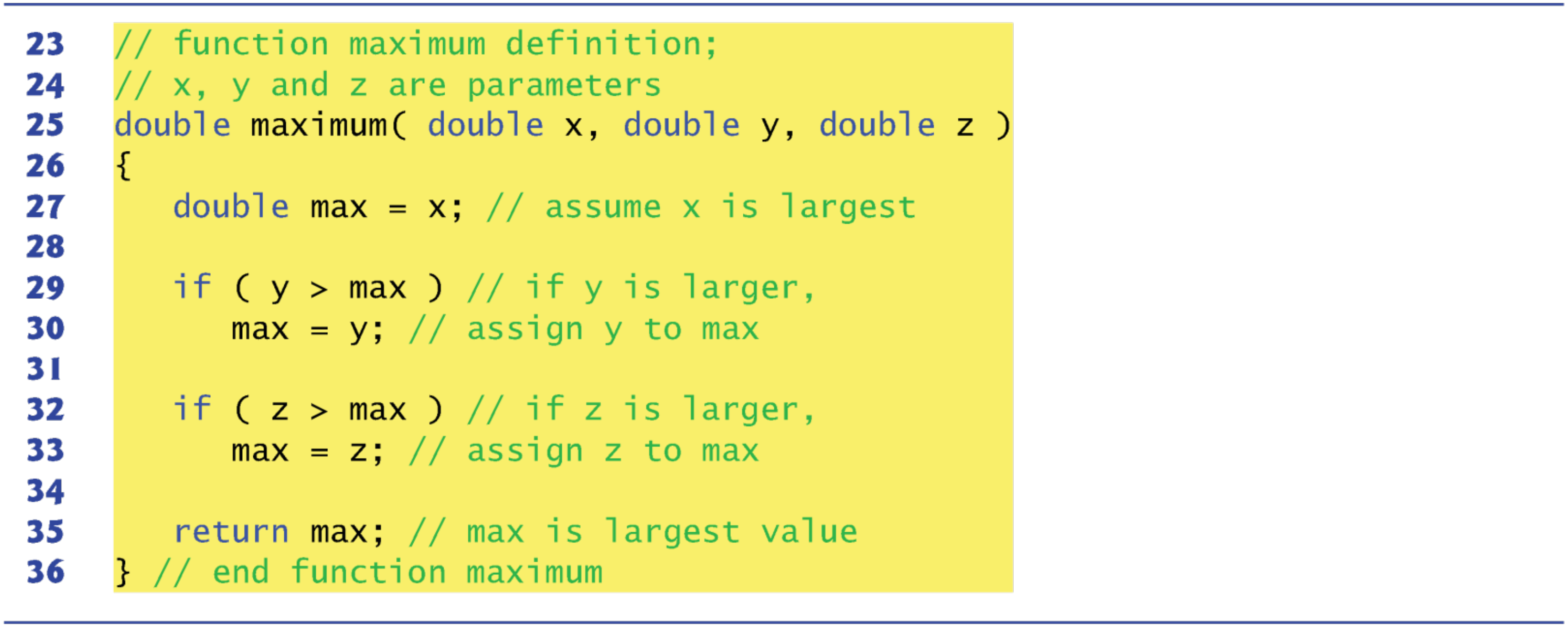
```
 8  int main()
 9  {
10     // loop 10 times and calculate and output the
11     // square of x each time
12     for ( int x = 1; x <= 10; x++ )
13        cout << square( x ) << "  ";  // function call
14
15     cout << endl;
16  } // end main
17
18  // square function definition returns square of an integer
19  int square( int y )  // y is a copy of argument to function
20  {
21     return y * y;     // returns square of y as an int
22  } // end function square
```

Function With Multiple Arguments



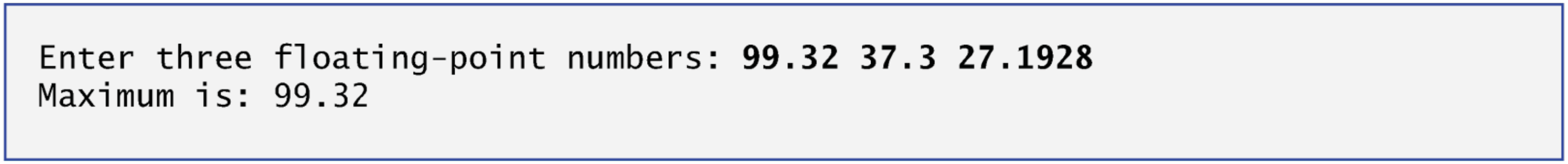
```cpp

// Finding the maximum of three floating-point numbers.
#include <iostream>
using namespace std;

double maximum( double, double, double ); // function prototype

int main()
{
   double number1;
   double number2;
   double number3;

   cout << "Enter three floating-point numbers: ";
   cin >> number1 >> number2 >> number3;

   // number1, number2 and number3 are arguments to
   // the maximum function call
   cout << "Maximum is: "
        << maximum( number1, number2, number3 ) << endl;
} // end main

```

Function with Multiple Arguments

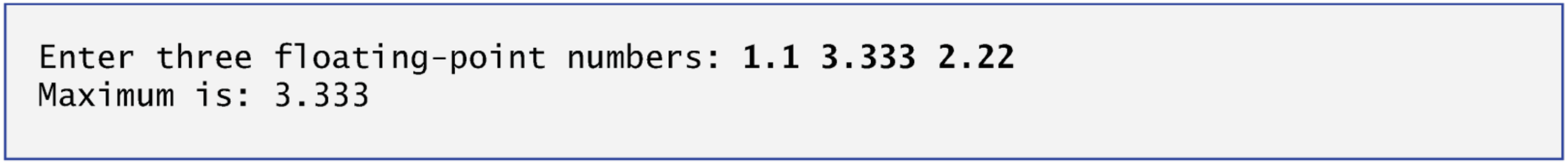


```
// function maximum definition;
// x, y and z are parameters
double maximum( double x, double y, double z )
{
   double max = x; // assume x is largest

   if ( y > max ) // if y is larger,
      max = y; // assign y to max

   if ( z > max ) // if z is larger,
      max = z; // assign z to max

   return max; // max is largest value
} // end function maximum
```

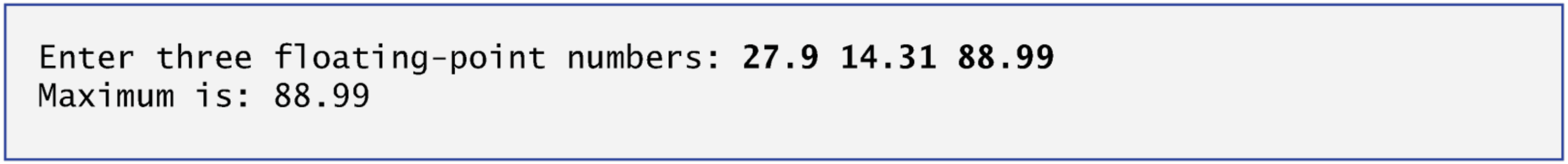
Function with Multiple Arguments



```
Enter three floating-point numbers: 99.32 37.3 27.1928
Maximum is: 99.32
```



```
Enter three floating-point numbers: 1.1 3.333 2.22
Maximum is: 3.333
```



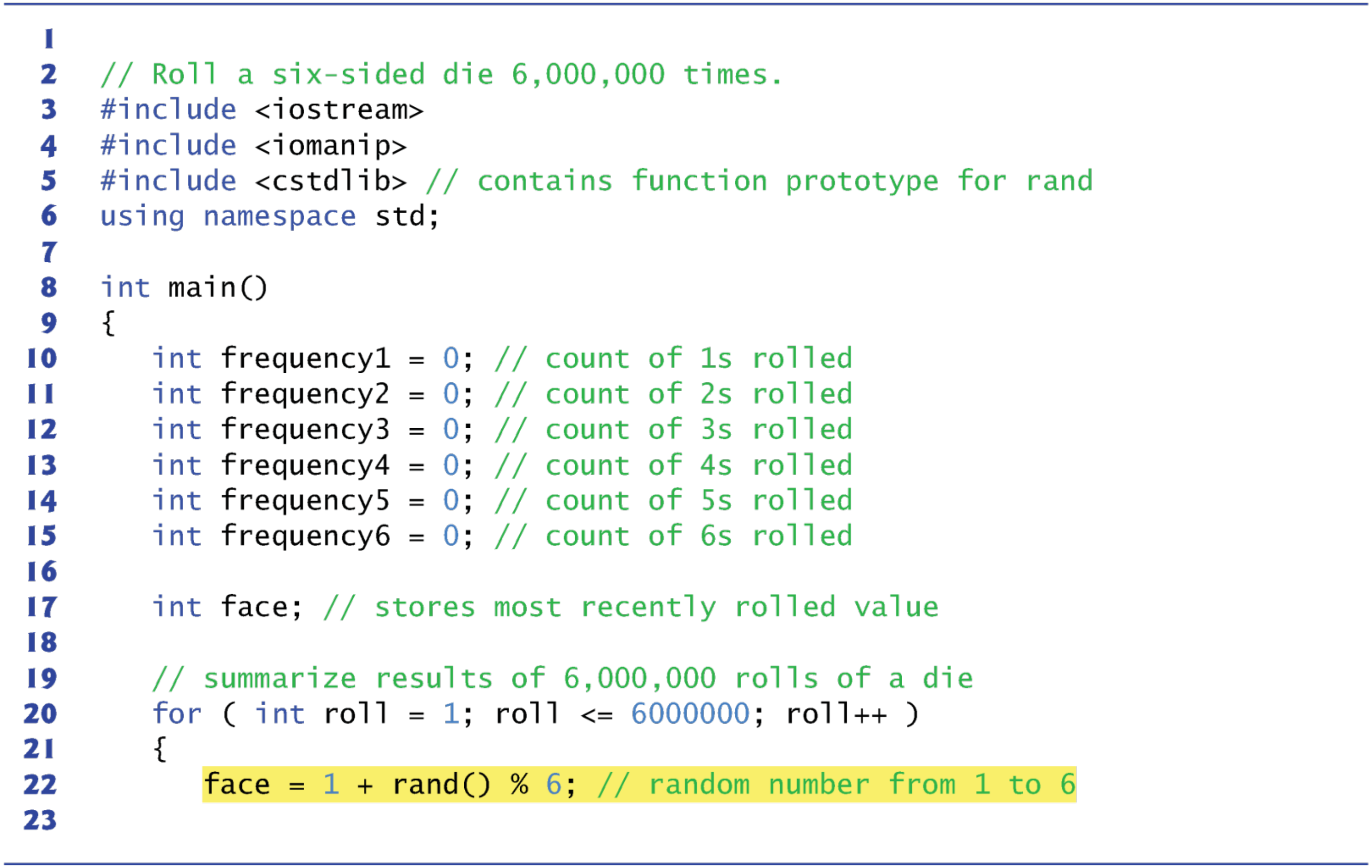
```
Enter three floating-point numbers: 27.9 14.31 88.99
Maximum is: 88.99
```

Case Study: Random Number Generator

- ❑ **The element of chance can be introduced into computer applications by using the C++ Standard Library function rand.**
  - ❑ The function prototype for the `rand` function is in `<cstdlib>`.
  - ❑ For example: `i = rand();`
- ❑ **Function `rand` generates an unsigned integer between 0 and RAND_MAX (a constant defined in the `<cstdlib>` header file).**
  - ❑ For GNU C++, the value of RAND_MAX is 2147483647; for Visual Studio, the value of RAND_MAX is 32767.
- ❑ **To produce integers in the range 0 to 5, we use the modulus operator (%) with `rand` as follows → `rand() % 6`**
  - ❑ The number 6 is called the scaling factor.
- ❑ **Shifting the range of numbers produces the integers from 1 to 6.**

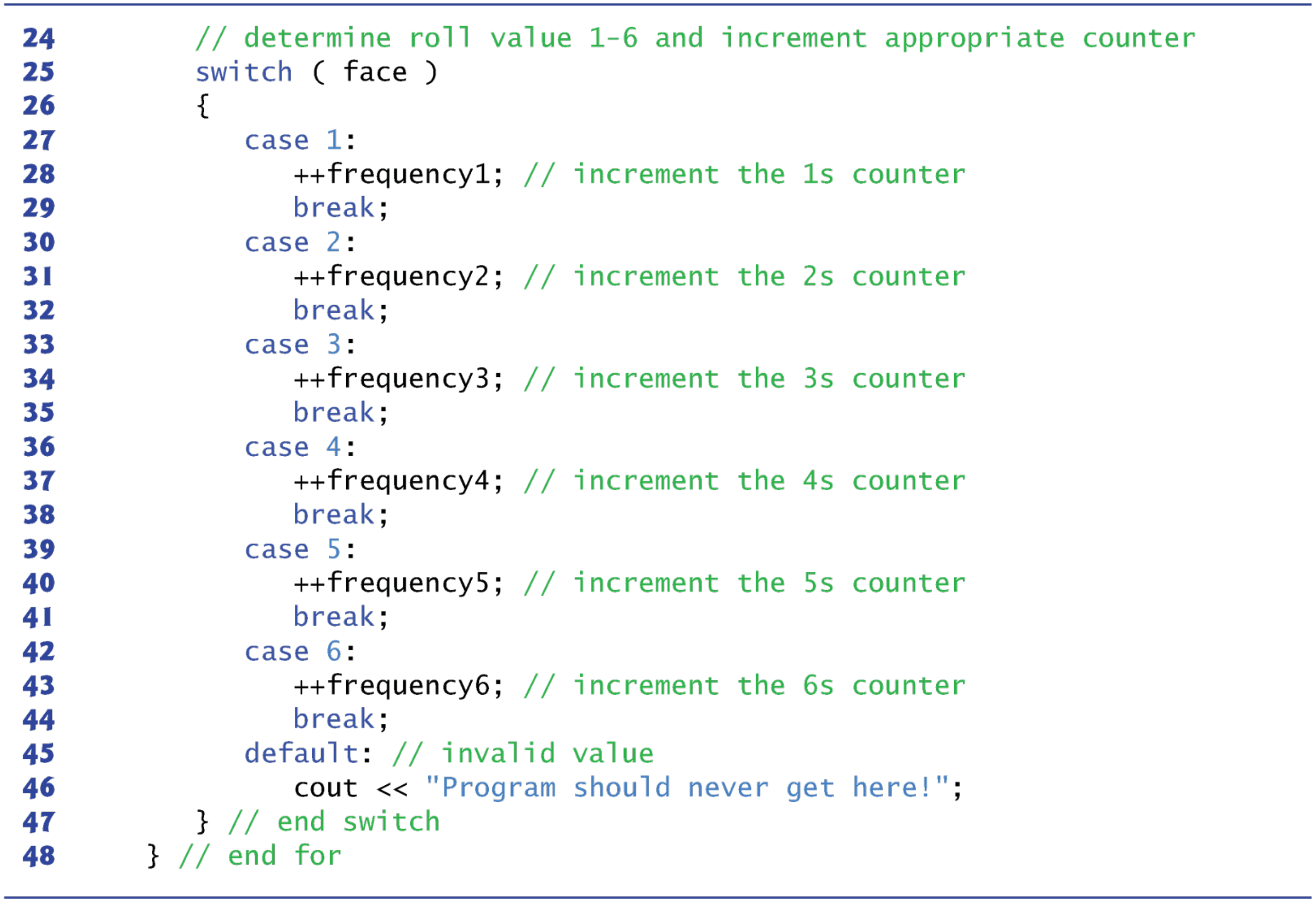
```
1 + rand() % 6
```

Case Study: Random Number Generator



```cpp

// Roll a six-sided die 6,000,000 times.
#include <iostream>
#include <iomanip>
#include <cstdlib> // contains function prototype for rand
using namespace std;

int main()
{
   int frequency1 = 0; // count of 1s rolled
   int frequency2 = 0; // count of 2s rolled
   int frequency3 = 0; // count of 3s rolled
   int frequency4 = 0; // count of 4s rolled
   int frequency5 = 0; // count of 5s rolled
   int frequency6 = 0; // count of 6s rolled

   int face; // stores most recently rolled value

   // summarize results of 6,000,000 rolls of a die
   for ( int roll = 1; roll <= 6000000; roll++ )
   {
      face = 1 + rand() % 6; // random number from 1 to 6

```

Case Study: Random Number Generator



```
      // determine roll value 1-6 and increment appropriate counter
      switch ( face )
      {
         case 1:
            ++frequency1; // increment the 1s counter
            break;
         case 2:
            ++frequency2; // increment the 2s counter
            break;
         case 3:
            ++frequency3; // increment the 3s counter
            break;
         case 4:
            ++frequency4; // increment the 4s counter
            break;
         case 5:
            ++frequency5; // increment the 5s counter
            break;
         case 6:
            ++frequency6; // increment the 6s counter
            break;
         default: // invalid value
            cout << "Program should never get here!";
      } // end switch
   } // end for
```

Case Study: Random Number Generator

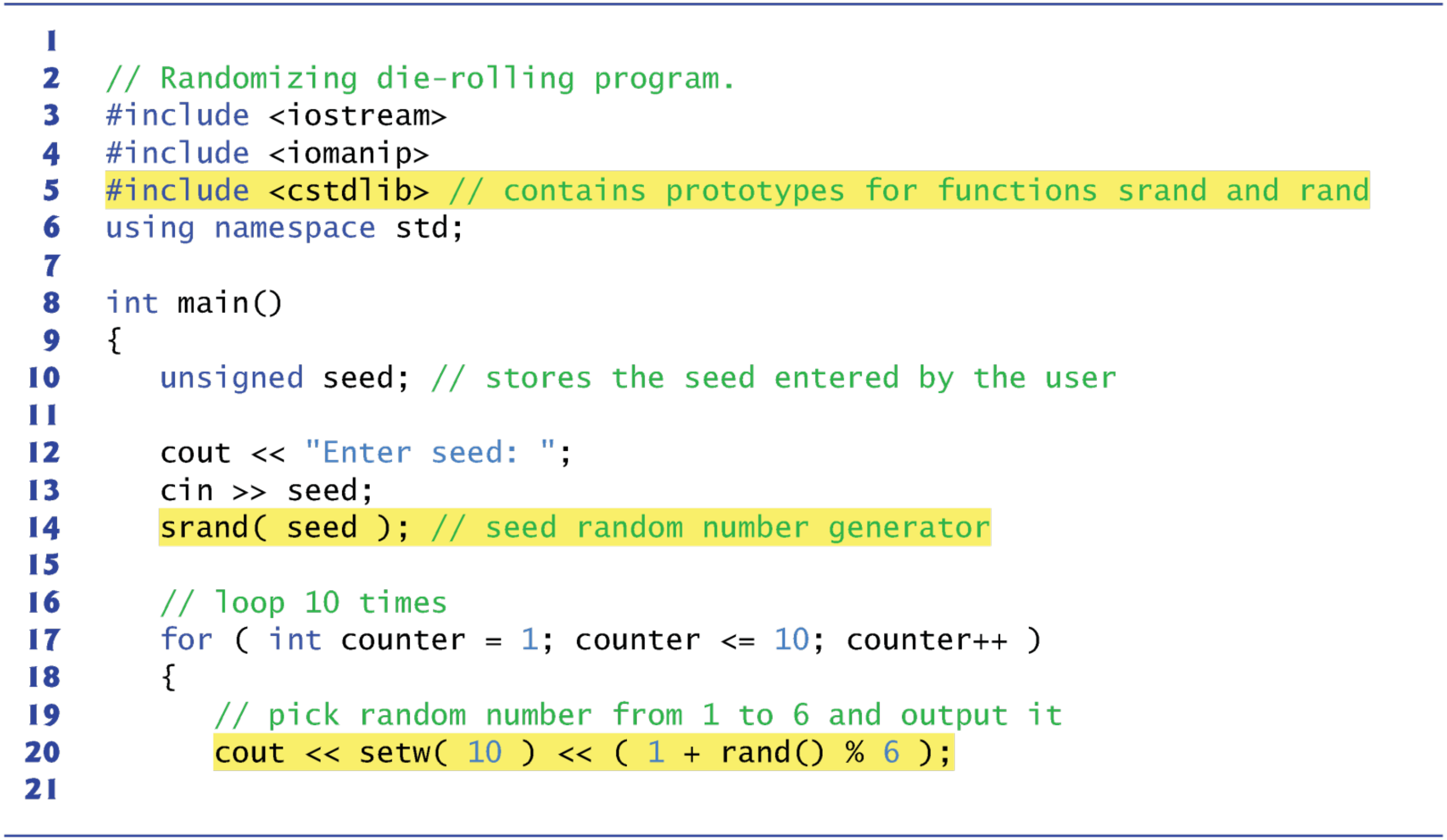
```cpp

   cout << "Face" << setw( 13 ) << "Frequency" << endl; // output headers
   cout << "   1" << setw( 13 ) << frequency1
      << "\n   2" << setw( 13 ) << frequency2
      << "\n   3" << setw( 13 ) << frequency3
      << "\n   4" << setw( 13 ) << frequency4
      << "\n   5" << setw( 13 ) << frequency5
      << "\n   6" << setw( 13 ) << frequency6 << endl;
} // end main
```

```
Face    Frequency
   1       999702
   2      1000823
   3       999378
   4       998898
   5      1000777
   6      1000422
```

True Randomness …?

- ❑ **Function `rand` actually generates pseudorandom numbers.**
- ❑ **The numbers in the sequence appear to be random, but the sequence repeats itself each time the program executes.**
- ❑ **It can be conditioned to produce a different sequence of random numbers for each execution.**
- ❑ **This is called randomizing and is accomplished with the C++ Standard Library function srand.**
- ❑ **Function `srand` takes an `unsigned` integer argument and seeds the `rand` function to produce a different sequence of random numbers for each execution.**
  - ❑ The function prototype for `srand` is in header file `<cstdlib>`.

True Randomness …?



```

// Randomizing die-rolling program.
#include <iostream>
#include <iomanip>
#include <cstdlib> // contains prototypes for functions srand and rand
using namespace std;

int main()
{
   unsigned seed; // stores the seed entered by the user

   cout << "Enter seed: ";
   cin >> seed;
   srand( seed ); // seed random number generator

   // loop 10 times
   for ( int counter = 1; counter <= 10; counter++ )
   {
      // pick random number from 1 to 6 and output it
      cout << setw( 10 ) << ( 1 + rand() % 6 );

```

True Randomness …?

```
      // if counter is divisible by 5, start a new line of output
      if ( counter % 5 == 0 )
         cout << endl;
   } // end for
} // end main
```

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

```
Enter seed: 432
         4         6         3         1         6
         3         1         5         4         2
```

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

Practical Use of Seed

- ❑ To randomize without having to enter a seed each time, we may use a statement like

```
        srand( time( 0 ) );
```

- ❑ This causes the computer to read its clock to obtain the value for the seed.
- ❑ Function time (with the argument 0) typically returns the current time as the number of seconds since January 1, 1970, at midnight Greenwich Mean Time (GMT).
- ❑ This value is converted to an `unsigned` integer and used as the seed to the random number generator.
- ❑ The function prototype for `time` is in `<ctime>`.

Summary

- ❑ Function
- ❑ Function with Multiple Arguments
- ❑ Case Study: Random Number Generator

LAB

- **Write a function “is_prime” that**
  - Takes an positive integer N
  - Returns “true” if N is a prime or “false”
- **In the main function, write a for loop that**
  - Loops from 2 to 1000
  - Call the function “is_prime” at each iteration to check if the number is a prime
  - Prints all prime numbers in the range [2, 1000]