Lecture 5

Functions with math and randomness

3.1 Introduction

- Divide and conquer
  - Construct a program from smaller pieces or components
  - Each piece more manageable than the original program

3.2 Program Components in C++

- Modules: functions and classes
- Programs use new and “prepackaged” modules
  - New: programmer-defined functions, classes
  - Prepackaged: from the standard library
- Functions invoked by function call
  - Function name and information (arguments) it needs
- Function definitions
  - Only written once
  - Hidden from other functions

3.2 Program Components in C++

- Boss to worker analogy
  - A boss (the calling function or caller) asks a worker (the called function) to perform a task and return (i.e., report back) the results when the task is done.

3.3 Math Library Functions

- Perform common mathematical calculations
  - Include the header file **`<cmath>`**
- Functions called by writing
  - functionName (argument);

  or
  - functionName(argument1, argument2, …);
- Example

```
cout << sqrt( 900.0 );
```

  - sqrt (square root) function The preceding statement would print 30
  - All functions in math library return a **`double`**

3.3 Math Library Functions

- Function arguments can be
  - Constants
    - `sqrt( 4 );`
  - Variables
    - `sqrt( x );`
  - Expressions
    - `sqrt( sqrt( x ) ) ;`
    - `sqrt( 3 - 6x );`

Method	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 2.71828 exp(2.0) is 7.38906
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(13.657, 2.333) is 1.992
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
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	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
tan(x)	trigonometric tangent of x (x in radians)	tan(0.0) is 0

Fig. 3.2 Math library functions.

3.4 Functions

- Functions
  - Modularize a program
  - Software reusability
    - Call function multiple times
- Local variables
  - Known only in the function in which they are defined
  - All variables declared in function definitions are local variables
- Parameters
  - Local variables passed to function when called
  - Provide outside information

3.5 Function Definitions

- Function prototype
  - Tells compiler argument type and return type of function
  - `int square( int );`
    - Function takes an `int` and returns an `int`
  - Explained in more detail later

- Calling/invoking a function
  - `square(x);`
  - Parentheses an operator used to call function
    - Pass argument x
    - Function gets its own copy of arguments
  - After finished, passes back result

3.5 Function Definitions

- Format for function definition

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

  - Parameter list
    - Comma separated list of arguments
      - Data type needed for each argument
    - If no arguments, use `void` or leave blank
  - Return-value-type
    - Data type of result returned (use `void` if nothing returned)

3.5 Function Definitions

- Example function

```
int square( int y )
{
  return y * y;
}
```

- **`return`** keyword
  - Returns data, and control goes to function's caller
    - If no data to return, use **`return;`**
  - Function ends when reaches right brace
    - Control goes to caller
- Functions cannot be defined inside other functions
- Next: program examples

fig03_03.cpp (1 of 2)

```
// Fig. 3.3: fig03_03.cpp
// Creating and using a programmer-defined function.
#include <iostream>

using std::cout;
using std::endl;

int square( int );   // function prototype

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

  cout << endl;

  return 0;  // indicates successful termination

} // end main
```

fig03_03.cpp
(2 of 2)

```
// 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
```

fig03_03.cpp
output
(1 of 1)

```
1  4  9  16  25  36  49  64  81  100
```


fig03_04.cpp
(1 of 2)

```
// Fig. 3.4: fig03_04.cpp
// Finding the maximum of three floating-point numbers.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

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;

```

fig03_04.cpp
(2 of 2)

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

   return 0;  // indicates successful termination
} // end main

// 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
```


fig03_04.cpp
output (1 of 1)

```
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
```

3.15 Functions with Empty Parameter Lists

- Empty parameter lists
  - **`void`** or leave parameter list empty
  - Indicates function takes no arguments
  - Function **`print`** takes no arguments and returns no value
    - `void print();`
    - `void print( void );`

fig03_18.cpp (1 of 2)

```
// Fig. 3.18: fig03_18.cpp
// Functions that take no arguments.
#include <iostream>

using std::cout;
using std::endl;

void function1();       // function prototype
void function2( void );  // function prototype

int main()
{
   function1();  // call function1 with no arguments
   function2();  // call function2 with no arguments

   return 0;    // indicates successful termination

} // end main

```

(2 of 2)

fig03_18.cpp output (1 of 1)

```
// function1 uses an empty parameter list to specify that
// the function receives no arguments
void function1()
{
   cout << "function1 takes no arguments" << endl;

} // end function1

// function2 uses a void parameter list to specify that
// the function receives no arguments
void function2( void )
{
   cout << "function2 also takes no arguments" << endl;

} // end function2
```

```
function1 takes no arguments
function2 also takes no arguments
```


3.6 Function Prototypes

- Function prototype contains
  - Function name
  - Parameters (number and data type)
  - Return type (**void** if returns nothing)
  - Only needed if function definition after function call
- Prototype must match function definition
  - Function prototype

    ```
    double maximum( double, double, double );
    ```
  - Definition

    ```
    double maximum( double x, double y, double z )
    {
      …
    }
    ```

3.6 Function Prototypes

- Function signature
  - Part of prototype with name and parameters
    - `double maximum( double, double, double );`

Function signature

- Argument Coercion
  - Force arguments to be of proper type
    - Converting **`int`** (4) to **`double`** (4.0)

    `cout << sqrt(4)`
  - Conversion rules
    - Arguments usually converted automatically
    - Changing from **`double`** to **`int`** can truncate data
      - 3.4 to 3
  - Mixed type goes to highest type (promotion)
    - **`int * double`**

3.6 Function Prototypes

Data types	
`long double`	
`double`	
`float`	
`unsigned long int`	(synonymous with `unsigned long`)
`long int`	(synonymous with `long`)
`unsigned int`	(synonymous with `unsigned`)
`int`	
`unsigned short int`	(synonymous with `unsigned short`)
`short int`	(synonymous with `short`)
`unsigned char`	
`char`	
`bool`	(`false` becomes 0, `true` becomes 1)

Fig. 3.5 Promotion hierarchy for built-in data types.

3.7 Header Files

- Header files contain
  - Function prototypes
  - Definitions of data types and constants
- Header files ending with .h
  - Programmer-defined header files

    `#include "myheader.h"`
- Library header files

    `#include <cmath>`

3.8 Random Number Generation

- **`rand`** function (**`<cstdlib>`**)
  - **`i = rand();`**
  - Generates unsigned integer between 0 and RAND_MAX (usually 32767)
- Scaling and shifting
  - Modulus (remainder) operator: **`%`**
    - **`10 % 3`** is **`1`**
    - **`x % y`** is between **0** and **`y – 1`**
  - Example

    `i = rand() % 6 + 1;`
    - "**`Rand() % 6`**" generates a number between **0** and **5** (scaling)
    - "**`+ 1`**" makes the range 1 to 6 (shift)
  - Next: program to roll dice

fig03_07.cpp
(1 of 2)

```
// Fig. 3.7: fig03_07.cpp
// Shifted, scaled integers produced by 1 + rand() % 6.
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstdlib>   // contains function prototype for rand

int main()
{
```


fig03_07.cpp
(2 of 2)

```
  // loop 20 times
  for ( int counter = 1; counter <= 20; counter++ ) {

    // pick random number from 1 to 6 and output it
    cout << setw( 10 ) << ( 1 + rand() % 6 );

    // if counter divisible by 5, begin new line of output
    if ( counter % 5 == 0 )
      cout << endl;

  } // end for structure

  return 0;  // indicates successful termination

} // end main
```

fig03_07.cpp
output (1 of 1)

```
         6         6         5         5         6
         5         1         1         5         3
         6         6         2         4         2
         6         2         3         4         1
```

3.8 Random Number Generation

- Next
  - Program to show distribution of **`rand()`**
  - Simulate 6000 rolls of a die
  - Print number of 1's, 2's, 3's, etc. rolled
  - Should be roughly 1000 of each

fig03_08.cpp (1 of 3)

```
// Fig. 3.8: fig03_08.cpp
// Roll a six-sided die 6000 times.
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstdlib>   // contains function prototype for rand

int main()
{
   int frequency1 = 0;
   int frequency2 = 0;
```

fig03_08.cpp (2 of 3)

```
int frequency3 = 0;
int frequency4 = 0;
int frequency5 = 0;
int frequency6 = 0;
int face;  // represents one roll of the die

// loop 6000 times and summarize results
for ( int roll = 1; roll <= 6000; roll++ ) {
  face = 1 + rand() % 6;  // random number from 1 to 6

  // determine face value and increment appropriate counter
  switch ( face ) {

    case 1:        // rolled 1
      ++frequency1;
      break;

    case 2:        // rolled 2
      ++frequency2;
      break;

```


fig03_08.cpp (3 of 3)

```
    case 3:        // rolled 3
      ++frequency3;
      break;

    case 4:        // rolled 4
      ++frequency4;
      break;

    case 5:        // rolled 5
      ++frequency5;
      break;

    case 6:        // rolled 6
      ++frequency6;
      break;

    default:       // invalid value
      cout << "Program should never get here!";

  } // end switch

} // end for

```

fig03_08.cpp output (1 of 1)

```
   // display results in tabular format
   cout << "Face" << setw( 13 ) << "Frequency"
      << "\n   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;

   return 0;  // indicates successful termination

} // end main
```

```
Face     Frequency
   1          1003
   2          1017
   3           983
   4           994
   5          1004
   6           999
```


3.8 Random Number Generation

- Calling rand() repeatedly
  - Gives the same sequence of numbers
- Pseudorandom numbers
  - Preset sequence of "random" numbers
  - Same sequence generated whenever program run
- To get different random sequences
  - Provide a seed value
    - Like a random starting point in the sequence
    - The same seed will give the same sequence
  - **`srand(seed);`**
    - **`<cstdlib>`**
    - Used before **`rand()`** to set the seed

fig03_09.cpp (1 of 2)

```
// Fig. 3.9: fig03_09.cpp
// Randomizing die-rolling program.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include <iomanip>

using std::setw;

// contains prototypes for functions srand and rand
#include <cstdlib>

// main function begins program execution
int main()
{
```


fig03_09.cpp (2 of 2)

```
  unsigned seed;

  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 );

    // if counter divisible by 5, begin new line of output
    if ( counter % 5 == 0 )
      cout << endl;

  } // end for

  return 0;  // indicates successful termination

} // end main
```

fig03_09.cpp output (1 of 1)

rand() gives the same sequence if it has the same initial seed.

```
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
```


3.8 Random Number Generation

- Can use the current time to set the seed
  - No need to explicitly set seed every time
  - **`srand( time( 0 ) );`**
  - **`time( 0 );`**
    - **`<ctime>`**
    - Returns current time in seconds
- General shifting and scaling
  - *Number* = *shiftingValue* + **`rand()`** % *scalingFactor*
  - shiftingValue = first number in desired range
  - scalingFactor = width of desired range

3.9 Example: Game of Chance and Introducing enum

- Enumeration
  - Set of integers with identifiers

    enum *typeName* **{** *constant1*, *constant2*... **}** ;
  - Constants start at 0 (default), incremented by 1
  - Constants need unique names
  - Cannot assign integer to enumeration variable
    - Must use a previously defined enumeration type
- Example

```
enum Status {CONTINUE, WON, LOST};
Status enumVar;
enumVar = WON; // cannot do enumVar = 1
```


3.9 Example: Game of Chance and Introducing enum

- Enumeration constants can have preset values

```
enum Months { JAN = 1, FEB, MAR, APR, MAY,
 JUN, JUL, AUG, SEP, OCT, NOV, DEC};
```

  - Starts at 1, increments by 1
- Next: craps simulator
  - Roll two dice
  - 7 or 11 on first throw: player wins
  - 2, 3, or 12 on first throw: player loses
  - 4, 5, 6, 8, 9, 10
    - Value becomes player's "point"
    - Player must roll his point before rolling 7 to win

fig03_10.cpp (1 of 5)

```
// Fig. 3.10: fig03_10.cpp
// Craps.
#include <iostream>

using std::cout;
using std::endl;

// contains function prototypes for functions srand and rand
#include <cstdlib>

#include <ctime>   // contains prototype for function time

int rollDice( void );  // function prototype

```


fig03_10.cpp (2 of 5)

```
int main()
{
   // enumeration constants represent game status
   enum Status { CONTINUE, WON, LOST };

   int sum;
   int myPoint;

   Status gameStatus;  // can contain CONTINUE, WON or LOST

   // randomize random number generator using current time
   srand( time( 0 ) );

   sum = rollDice();   // first roll of the dice

```

fig03_10.cpp (3 of 5)

```
 // determine game status and point based on sum of dice
 switch ( sum ) {
   // win on first roll
   case 7:
   case 11:
     gameStatus = WON;
     break;

   // lose on first roll
   case 2:
   case 3:
   case 12:
     gameStatus = LOST;
     break;

   // remember point
   default:
     gameStatus = CONTINUE;
     myPoint = sum;
     cout << "Point is " << myPoint << endl;
     break;          // optional
 } // end switch

```


fig03_10.cpp (4 of 5)

```
 // while game not complete ...
 while ( gameStatus == CONTINUE ) {
   sum = rollDice();        // roll dice again

   // determine game status
   if ( sum == myPoint )     // win by making point
     gameStatus = WON;
   else
     if ( sum == 7 )        // lose by rolling 7
       gameStatus = LOST;

 } // end while

 // display won or lost message
 if ( gameStatus == WON )
   cout << "Player wins" << endl;
 else
   cout << "Player loses" << endl;

 return 0;  // indicates successful termination

} // end main

```

```
// roll dice, calculate sum and display results
int rollDice( void )
{
   int die1;
   int die2;
   int workSum;

   die1 = 1 + rand() % 6;  // pick random die1 value
   die2 = 1 + rand() % 6;  // pick random die2 value
   workSum = die1 + die2;  // sum die1 and die2

   // display results of this roll
   cout << "Player rolled " << die1 << " + " << die2
      << " = " << workSum << endl;

   return workSum;       // return sum of dice

} // end function rollDice
```


fig03_10.cpp output (1 of 1)

```
Player rolled 2 + 5 = 7
Player wins

Player rolled 6 + 6 = 12
Player loses

Player rolled 3 + 3 = 6
Point is 6
Player rolled 5 + 3 = 8
Player rolled 4 + 5 = 9
Player rolled 2 + 1 = 3
Player rolled 1 + 5 = 6
Player wins
Player rolled 1 + 3 = 4
Point is 4
Player rolled 4 + 6 = 10
Player rolled 2 + 4 = 6
Player rolled 6 + 4 = 10
Player rolled 2 + 3 = 5
Player rolled 2 + 4 = 6
Player rolled 1 + 1 = 2
Player rolled 4 + 4 = 8
Player rolled 4 + 3 = 7
Player loses
```