Chapter 2
C++ Fundamentals

3rd Edition

Computing Fundamentals with C++

Rick Mercer
Franklin, Beedle & Associates

Goals

- Reuse existing code in your programs with `#include`
- Obtain input data from the user and display information to the user with `cin` and `cout`
- Evaluate and create arithmetic expressions
- Use operations on objects
- Get input, show output

The C++ Programming Language

- A C++ program is a sequence of characters created with a text editor and stored as a file.
  - this is the source code
- The file type is usually .cpp

```
CourseGrade.cpp
```

General Forms

- General forms provide information to create syntactically correct programs
  - Anything in yellow boldface must be written exactly as shown (`cout <<` for example)
  - Anything in *italic* represents something that must be supplied by the user
  - The italicized portions are defined elsewhere

General Form for a program

```
// Comment
#include-directive(s)
using namespace std;
int main() {
   object-initializations
   statement(s)
   return 0;
}
```

Example C++ program

```
// This C++ program gets a number from the
// user and displays that value squared

#include <iostream>  // for cout cin endl

using namespace std;
int main() {
  double x;

  cout << "Enter a number: ";
  cin >> x;
  cout << "x squared: " << (x * x) << endl;

  return 0;
}
```

The compiler

- The compiler
  - reads source code in a character by character fashion
  - reports errors whenever possible
  - reports warnings to help avoid errors
  - conceptually replaces #includes with the source code of the #included file

#include directives

- General form: *#include-directive*

  #include <*include-file*>

  -or-

  #include "*include-file*"

  - < > causes a search of the system folder(s)
    - these files should be found automatically.
  - The form with " " first searches the working folder before searching the system folder(s)
    - the " " indicates a new file from your working folder.

Pieces of a C++ Program

- A token is the smallest recognizable unit in a programming language.
- C++ has four types of tokens:
  - special symbols
  - keywords
  - identifiers
  - constants

Tokens

- Each color represents a different type of token

```
special symbol    identifier     reserved-identifier
literal           comment
```

```
// Comment: This is a complete C++ program
#include <iostream>
using namespace std;

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

Special Symbols

- One or two character sequences (no spaces).

```
// <  >  (  )  {  <<  ;  }  !=
```

- Some special symbols mean different things in different contexts.

Identifiers

- There are some standard (always available with the C++ compiler) identifiers:

  `endl sqrt string width std`

- The programmer can make up new identifiers

  `test1   x1   aNumber   MAXIMUM   A_1`

Identifiers

- Identifiers have from 1 to 32 characters:
  `'a'..'z'    'A'..'Z'    '0'..'9'    '_'`
- Identifiers should start with a letter: a1 is legal, 1a is not (can also start with underscore _
- C++ is case sensitive. `A` and `a` are different.
- Which of these are valid identifiers?

```
a) abc        e) ABC            i) a_1
b) m/h        f) 25or6to4       j) student Number
c) main       g) 1_time         k) string
d) double     h) first name     l) ______
```

Reserved Identifiers

- Word like tokens with a pre-defined meaning that can't be changed (reserved-identifiers)

```
double   int
```

- Some of the keywords in the text :

```
bool     class    for      operator    typedef
case     do       if       return      void
char     else     long     switch      while
```

Literals

- floating-point literals

  `1.234   -12.5   0.0   0.   .0   1e10   0.1e-5`

- string literals

  `"character between double quotes"`

- integer literals

  `-1   0   1   -32768   +32767`

- character literals

  `'A'   'b'   '\n'   '1'`

Comments

- Provide internal documentation
- Helps us understand program that we must read--including our own
- Can be used as pseudo code within a program and later changed into C++ or left as is to provide documentation

```
// on one line or

/*
  between slash star and star slash
*/
```

Common Operations on Objects

- Common Operations for many classes of objects include these four
  - Declaration — *Construct an object*
  - Initialization — *Initialize the state of an object*
  - Assignment — *Modify the state of an object*
  - Input — *Modify the state of an object*
  - Output — *Inspect the state of an object*

Declare and Initialize Variables

- No initial state (values):

  type identifier;

```
double aNumber; // garbage value
```

  type identifier , identifier , … , identifier ;

```
int a, b, c;    // all have garbage values
```

- Supply initial state (values):

  type identifier = *initial-state*;

```
double aNumber = 0.0;
string name = "Chris Plumber";
```

  type identifier = *identifier* (*initial-state*);

```
string address("1040 E 4th");
```

Output with `cout`

- Programs must communicate with users
- This can be done with keyboard input statements and screen output statements
- A C++ statement is composed of several components properly grouped together to perform some operation.
- The next slide has the first statement used to display constants and object state to the screen

The `cout` statement

- The general form of a `cout` statement:

  cout << *expression-1* << *expression-2* << *expression-n* ;

- Example

```
cout << "Grade: " << courseGrade << endl;
```

What happens with cout?

- When a `cout` statement is encountered, the expressions are displayed on the screen in a manner appropriate to the expression
- When encountered in a `cout` statement, `endl` generates a new line on the console
- To properly use `cout` and `endl` your program must have this code at the top of the file:

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

What is the output?

```
#include <iostream>  // for cout and endl
using namespace std; // so we don't need std::

int main() {
  double aDouble = 1.1;
  string name = "Carpenter";

  cout << (3 * 2.5) << (2 * 3) << endl;
  cout << 2 * aDouble;
  cout << name;

  return 0;
}
```

Output?

Assignment

- Certain objects have undefined state

```
double dunno, do_you;
cout << dunno << endl;  // Output? ______
```

- The programmer can set the state of objects with assignment operations of this form:

  object-name = *expression* ;

- Examples:

```
dunno = 1.23;
do_you = dunno - 0.23;
```

Memory before and after

Object Name	*Old State*	*Modified State*
`dunno`	`?`	`1.23`
`do_you`	`?`	`1.0`

- The expression must be a value that the object can store (assignment compatible)

```
dunno = "Ohhh no, you can't do that"; // <- Error
string str;
str = 1.23;  // <- Error also
```

Assignment Statement

- Write the values for `bill` and `name`

```
double bill;
string name;

bill = 10.00;
bill = bill + (0.06 * bill);
name = "Bee Bop";
name = "Hip Hop";
// bill is  ___________?

// name is now ________?
```

Input with cin

- General forms :

  cin >> *object-1* ;

  -or-

  cin >> *object-1* >> *object-2* >> *object-n* ;

- Example: `cin >> test1;`
  - When a cin statement is encountered
    - the program pauses for user input
    - the characters typed by the user are processed
    - the object's state is changed to the value of the input

Input is Separated by Whitespace

blanks, tabs, newlines

```
#include <iostream>  // for cout, cin, endl
#include <string>    // for class string
using namespace std; // avoid writing std::

int main() {
  string name;
  cout << "Enter your name: ";
  cin >> name;
  cout << "Hello " << name;
  return 0;
}
```

Dialogue when the user enters `Dakota Butler`

Note: `WindowMaker` *is still waiting for a non-existent future* `cin`

```
Enter your name: Dakota Butler
Hello Dakota
```

Arithmetic Expressions

- Arithmetic expressions consist of operators
  `+ - / * %`
  and operands like `40 payRate hours`
- Example expression used in an assignment:
  ```
  grossPay = payRate * hours;
  ```
- Example expression:
  ```
  (40 * payRate) + 1.5 * payRate * (hours - 40)
  ```
- The previous expression has how many
  operators ?___ operands ?___

Arithmetic Expressions

- A recursive definition

a numeric object	x
or a numeric constant	100 or 99.5
or expression + expression	1.0 + x
or expression - expression	2.5 - x
or expression * expression	2 * x
or expression / expression	x / 2.0
or (expression)	(1 + 2.0)

Precedence of Arithmetic Operators

- Expressions with more than one operator require some sort of precedence rules:

  `*   /` evaluated left to right order

  `-   +` evaluated left to right order

  What is `2.0 + 4.0 - 6.0 * 8.0 / 6.0` ______

- Use (parentheses) for readability or to intentionally alter an expression:

```
double C;
double F = 212.0;
C = 5.0 / 9.0 * (F - 32);  // C = _____
```

What is the complete dialogue with input 2.3 5.0 2.0

```
#include <iostream>  // for cin, cout, and endl
using namespace std;
int main() {
  // Declare Objects
  double x, y, z, average;

  // Input: Prompt for input from the user
  cout << "Enter three numbers: ";
  cin >> x >> y >> z;
  // Process:
  average = (x + y + z) / 3.0;
  // Output:
  cout << "Average: " << average << endl;

  return 0;
}
```

`int` Arithmetic

- `int` variables are similar to double, except they can only store whole numbers (integers)

```
int anInt = 0;
int another = 123;
int noCanDo = 1.99; // <- ERROR
```

- Division with `int` is also different
  - performs quotient remainder whole numbers only

```
anInt = 9 / 2;        // anInt = 4, not 4.5
anInt = anInt / 5;    // What is anInt now? ____
anInt = 5 / 2;        // What is anInt now? ____
```

The integer % operation

- The % operator returns the remainder

```
int anInt = 9 % 2;   // anInt ___1___
anInt = 101 % 2;     // What is anInt now? ___
anInt = 5 % 11;      // What is anInt now? ___
anInt = 361 % 60;    // What is anInt now? ___

int quarter;
quarter = 79 % 50 / 25; // What is quarter? ___
quarter = 57 % 50 / 25; // What is quarter? ___
```

Integer division, watch out

```
int celcius, fahrenheit;

fahrenheit = 212;
celcius = 5 / 9 * (fahrenheit - 32);

// What is celcius? _____
```

const objects

- It is sometimes convenient to have objects that cannot have altered state.

  `const` class-name identifier `=` expression`;` *`// use this form`*

  -or-

  `const` class-name identifier `(` expression `);`

Examples:

```
const double PI = 3.1415926;
const string ERROR_MESSAGE = "Nooooooo";
```

Errors would occur on assignment:

```
PI = 9.8;
cin >> ERROR_MESSAGE;
```

Mixing types

- There are many numeric types
- In general, if the type differ, promote the "smaller" to the "larger"
- int + double will be a double

```
  5 + 1.23
5.0 + 1.23
   6.23
```

Another Algorithm Pattern: Prompt then Input

- The *Input/Process/Output* programming pattern can be used to help design many programs in the first several chapters
- The *Prompt then Input* pattern also occurs frequently.
  - The user is often asked to enter data
  - The programmer must make sure the user is told what to enter

Prompt then Input Pattern

Pattern	Prompt then Input
Problem	The user must enter something
Outline	1) Prompt the user for input 2) Obtain the input
Code Example	`cout << "Enter your first name: ";` `cin >> firstName;`

Examples

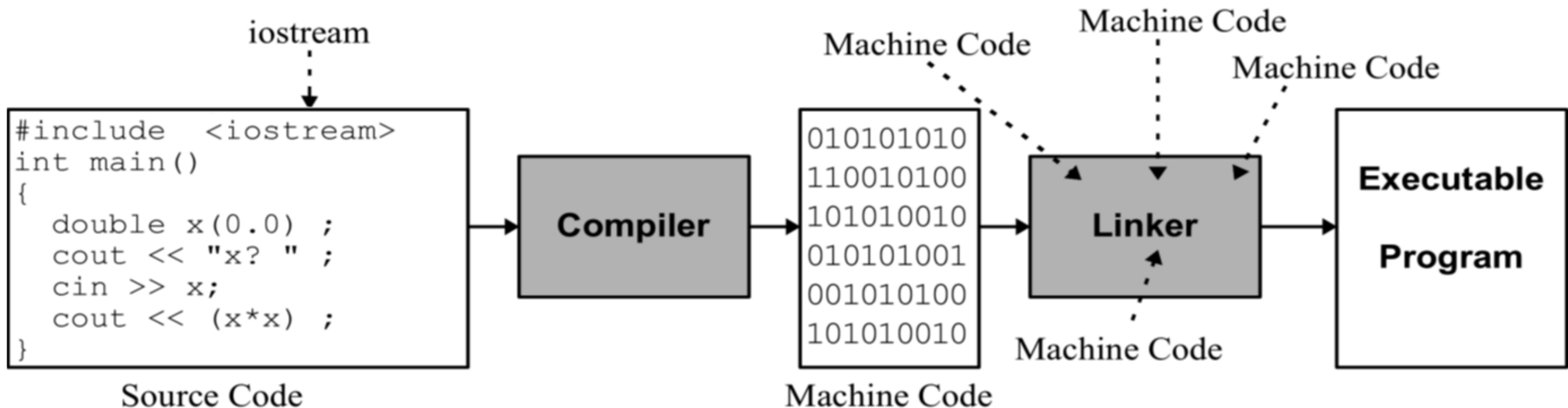
- Instances of the *Prompt then Input* pattern:

```
cout << "Enter your first name: ";
cin >> firstName;

cout << "Enter accumulated credits: ";
cin >> credits;
```

Compile, Link, and Run time

- The compiler translates source code into machine code
- The linker puts together several pieces of machine code to create an executable program
- Errors occur at various times



Errors and Warnings

- *compiletime*—syntax errors that occur during compilation (missing semicolon)
- *warnings*—code that appears risky, suggesting there may be a future error (<type>> needs space)
- *linktime*—errors that occur when the linker cannot find what it needs (missing .o file)
- *runtime*—errors that occur while the program is executing (a file is not found)
- *intent*—the program does what was typed, not what was intended (logic error produces wrong answer)

Errors Detected at Compiletime

- Generated while the compiler is processing the source code
- Compiler reports violations of syntax rules.
- For example, the compiler will try to inform you of the two syntax errors in this line

```
int student Number
```

Warnings generated by the compiler

- The compiler generates warnings when it discovers something that is legal, but potentially problematic
- Example

```
double x, y, z;
y = 2 * x;
```

warning: unused variable 'z' [-Wunused-variable]
warning: variable 'x' is uninitialized when used here [-Wuninitialized]

Linktime Errors

- Errors that occur while trying to put together (link) an executable program
- For example, we must always have a function named `main`
  - `Main` or `MAIN` won't do

Runtime Errors

- Errors that occur at runtime, that is, while the program is running
- Examples
  - Invalid numeric input by the user
  - Dividing an integer by 0
  - File not found when opening it (wrong name)
  - Indexing a list element with the index is out of range index

Intent Errors

- When the program does what you typed, not what you intended
- Imagine this code

```
cout << "Enter sum: ";
cin >> n;
cout << "  Enter n: ";
cin >> sum;
average = sum / n;
```

- Whose responsibility is it to find this error?

When the program doesn't work

- If none of the preceding errors occur, the program may still not be right
- The working program may not match the specification because either
  - The programmers did not match, or understand the problem statement
  - The problem statement may have been incorrect
  - Someone may have changed the problem