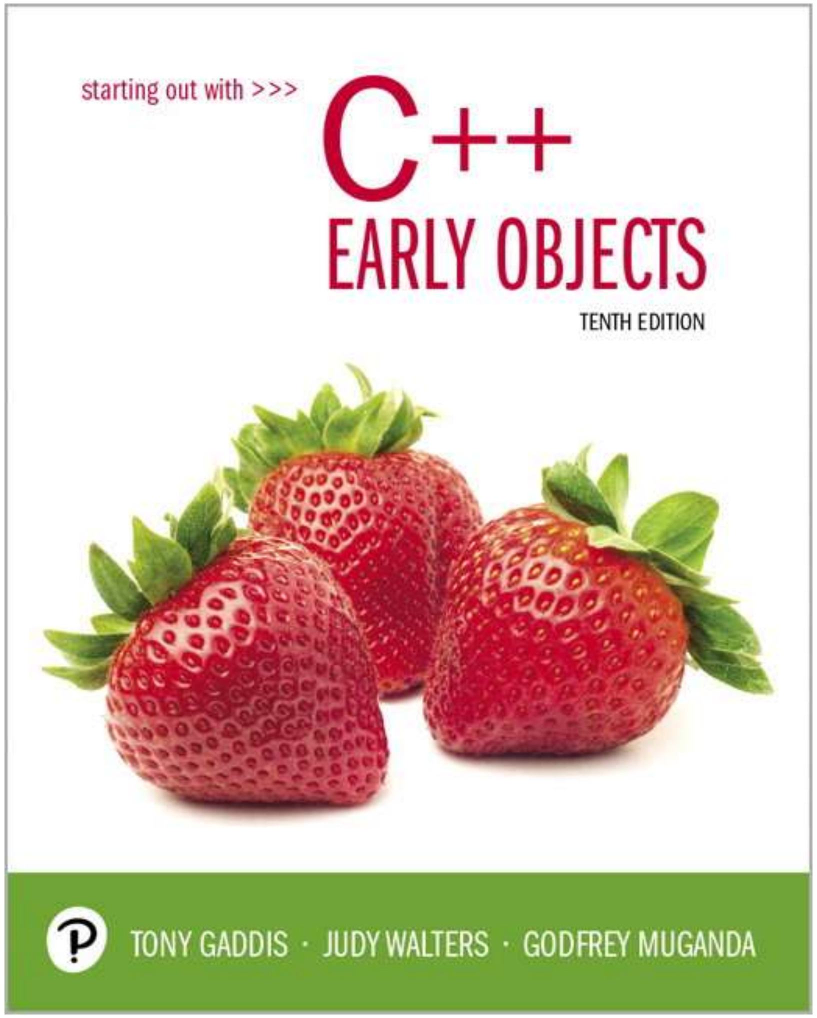
Starting Out with C++ Early Objects

Tenth Edition



Chapter 2

Introduction to C++

Topics 1of 2

Topics 2 of 2

2.10 The C++ **`string`** Class

2.11 The **`bool`** Data Type

2.12 Determining the Size of a Data Type

2.13 More on Variable Assignments and Initialization

2.14 Scope

2.15 Arithmetic Operators

2.16 Comments

2.17 Programming Style

2.1 The Parts of a C++ Program 1 of 2

```
// sample C++ program          ←  comment
#include <iostream>            ←  preprocessor directive
using namespace std;           ←  which namespace to use
int main()                     ←  beginning of function named main
{                              ←  beginning of block for main
    cout << "Hello, there!";   ←  output statement
    return 0;                  ←  send 0 back to operating system
}                              ←  end of block for main
```

2.1 The Parts of a C++ Program 2 of 2

Statement	Purpose
`// sample C++ program`	comment
`#include <iostream>`	preprocessor directive
`using namespace std;`	which namespace (set of names) to use
`int main()`	beginning of function named `main`
`{`	beginning of block for `main`
`    cout << "Hello, there!";`	output statement
`    return 0;`	send 0 back to the operating system
`}`	end of block for `main`

Checkpoint Pg. 33

Special Characters

Character	Name	Description
`//`	Double Slash	Begins a comment
`#`	Pound Sign	Begins preprocessor directive
`< >`	Open, Close Brackets	Encloses filename used in `#include` directive
`( )`	Open, Close Parentheses	Used when naming a function
`{ }`	Open, Close Braces	Encloses a group of statements
`" "`	Open, Close Double Quote Marks	Encloses a string of characters
`;`	Semicolon	Ends a programming statement

Important Details

- C++ is case-sensitive. Uppercase and lowercase characters are different characters. ‘`Main`’ is not the same as ‘`main`’.

- Every { must have a corresponding }, and vice-versa.

2.2 The `cout` Object

- Displays information on computer screen
- Use **<<** to send information to cout

```
cout << "Hello, there!";
```

- You can use **<<** to send multiple items to cout

```
cout << "Hello, " << "there!";
```

Or

```
cout << "Hello, ";
cout << "there!";
```

Starting a New Line

- To get multiple lines of output on screen
  - Use **`endl`**

    ```
    cout << "Hello, there!" << endl;
    ```

  - Use **`\n`** in an output string

    ```
    cout << "Hello, there!\n";
    ```

Hands-On Pg. 34

- Listing 2-2
- Listing 2-3

Hands-On Pg. 35

- Listing 2-4
- Listing 2-5

Hands-On Pg. 36

- Listing 2-6

Escape Sequences – More Control Over Output

Escape Sequence	Name	Description
`\n`	Newline	Causes the cursor to go to the next line for subsequent printing
`\t`	Horizontal tab	Causes the cursor to skip over to the next tab stop
`\a`	Alarm	Causes the computer to beep
`\b`	Backspace	Causes the cursor to back up (i.e., move left) one position
`\r`	Return	Causes the computer to go to the beginning of the current line, not the next line
`\\`	Backslash	Causes a backslash to be printed
`\'`	Single quote	Causes a single quotation mark to be printed
`\"`	Double quote	Causes a double quotation mark to be printed

Common Escape Sequence Mistakes

1) Don’t confuse `"\"` (a back slash) and `"/"` (a forward slash)
2) Remember to put **`\n`** in double quotation marks

2.3 The `#include` Directive

- Inserts the contents of another file into the program
- It is a preprocessor directive
  - Not part of the C++ language
  - Not seen by compiler
- Example:

```
#include <iostream>
```

No ; goes here

2.4 Variables and the Assignment Statement 1 of 2

A Variable

- Is used to refer to a location in memory where a value can be stored.
- An assignment statement is used to store a value.
- The value that is stored can be changed, *i.e.*, it can "vary".
- You must define the variable (indicate the name and the type of value that it can hold) before you can use it to store a value.

Variables

- If a new value is stored in the variable, it replaces the previous value
- The previous value is overwritten and can no longer be retrieved

```
int age;
age = 17;       // Assigns 17 to age
cout << age;    // Displays 17
age = 18;       // Now age is 18
cout << age;    // Displays 18
```

Assignment Statement

- Uses the = operator
- Has a single variable on the left side and a value on the right side
- Copies the value on the right into the location in memory that is associated with the variable on the left

```
item = 12;
```

Hands-On Pg. 39

- Listing 2-7

2.5 Literals

A Literal is a piece of data that is written directly in the source code of the program.

```
'A'       // character literal
"Hello"   // string literal
12        // integer literal
"12"      //  string literal (yes!)
3.14      // floating-point literal
```

Hands-On Pg. 41

- Listing 2-8

2.6 Identifiers

- Programmer-chosen names to represent parts of the program, such as variables
- Name should indicate the use of the identifier
- Cannot use C++ key words as identifiers
- Must begin with alphabetic character or _, followed by any number of alphabetic, numeric, or _ characters.
- Alphabetic characters may be upper- or lowercase

C++ Reserved Keywords

- Table 2-4, pg. 43

Multi-word Variable Names

- A variable name should reflect its purpose
- Descriptive variable names may include multiple words
- Two conventions to use in naming variables:
  - Capitalize all words but the first letter of first word. Run words together:

```
quantityOnOrder        totalSales
```

  - Use the underscore _ character as a word separator:

```
quantity_on_order      total_sales
```

- Use one convention consistently throughout a program

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
`totalSales`	Yes	
`total_sales`	Yes	
`total.Sales`	No	Cannot contain period
`4thQtrSales`	No	Cannot begin with digit
`total$Sales`	No	Cannot contain $

2.7 Integer Data Types

- Designed to hold whole (non-decimal) numbers
- Can be **`signed`** or **`unsigned`**

  `12        -6        +3`

- Available in different sizes (*i.e.*, number of bytes): **`short int`**, **`int`**, **`long int`**, and **`long long int`**
- **`long long int`** was introduced in C++ 11.

Signed vs. Unsigned Integers

- C++ allocates one bit for the sign of the number. The rest of the bits are for data.
- If a variable in a program will not hold negative values, you can declare the variable to be **`unsigned`**. All of the bits in unsigned numbers are used for data.
- A variable is signed unless the **`unsigned`** keyword is used at variable definition.

Integer Data Types

- Table 2-6 Pg. 46
- Table 2-7 Pg. 47

Hands-On Pg. 47

- Listing 2-9

Defining Variables

- Variables of the same type can be defined
  - In separate statements

```
int length;
int width;
```

  - In the same statement

```
int length,
    width;
```

- Variables of different types must be defined in separate statements

Abbreviated Variable Definitions

- **`int`** can be omitted from a variable definition for any datatype except an **`int`** itself.
- Examples:

```
short temperatures;
unsigned short booksOnOrder;
unsigned long long magnitude;
int grades;
```

Integral Literals

- To store an integer literal in a long memory location, put '**`L`**' at the end of the number:
  ```
  long rooms = 234L;
  ```
- Use '**`LL`**' at the end to put an integer literal in a long long memory location.
- Literals that begin with '**`0`**' (zero) are octal, or base 8: `075`
- Literals that begin with '**`0x`**' are hexadecimal, or base 16: `0x75A`

Hands-On Pg. 48

- Listing 2-10

2.8 Floating-Point Data Types

- Designed to hold real numbers

  `12.45`   **`-3.8`**

- Stored in a form similar to scientific notation
- Numbers are all signed
- Available in different sizes (number of bytes): **`float`**, **`double`**, and **`long double`**
- Size of **`float`** ≤ size of **`double`**
  ≤ size of **`long double`**

Floating-point Literals

- Can be represented in
  - Fixed point (decimal) notation:

    `31.4159`　　　`0.0000625`
  - E notation:

    `3.14159E1`　　　`6.25e-5`
- Are **`double`** by default
- Can be forced to be float **`3.14159F`** or long double **`0.0000625L`**

Hands-On Pg. 51

- Listing 2-11

Assigning Floating-point Values to Integer Variables

If a floating-point value (a literal or a variable) is assigned to an integer variable

- The fractional part will be truncated (*i.e.*, "chopped off" and discarded)
- The value is not rounded

```
int rainfall = 3.88;
cout << rainfall;  // Displays 3
```

2.9 The `char` Data Type

- Used to hold single characters or very small integer values
- Usually occupies 1 byte of memory
- A numeric code representing the character is stored in memory

SOURCE CODE	MEMORY
`char letter = 'C';`	variable letter holds the integer value 67

Hands-On Pg. 54

- Listing 2-12

Character Literal

- A character literal is a single character
- When referenced in a program, it is enclosed in single quotation marks:

```
    cout << 'Y' << endl;
```

- The quotation marks are not part of the literal, and are not displayed

Hands-On Pg. 55

- Listing 2-13
- Google search for ASCII table

String Literals

- Can be stored as a series of characters in consecutive memory locations

  `"Hello"`

- Stored in consecutive memory locations with the null terminator, `\0`, automatically placed at the end:

`H`	`e`	`l`	`l`	`o`	`\0`

- Is comprised of the characters between the `"   "`

A character or a string literal?

- A character literal is a single character, enclosed in single quotes:

```
    'C'
```

- A string literal is a sequence of characters enclosed in double quotes:

```
    "Hello, there!"
```

- A single character in double quotes is a string literal, not a character literal:

```
    "C"
```

2.10 The C++ **`string`** Class

- Must **`#include <string>`** to create and use string objects
- Can define **`string`** variables in programs

```
      string name;
```

- Can assign values to string variables with the assignment operator

```
    name = "George";
```

- Can display them with **`cout`**

```
    cout << "My name is " << name;
```

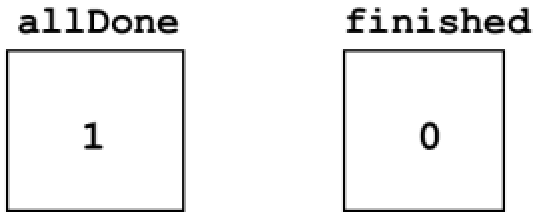
Hands-On Pg. 58

- Listing 2-15

2.11 The `bool` Data Type

- Represents values that are **`true`** or **`false`**
- **`bool`** values are stored as integers
- **`false`** is represented by 0, **`true`** by 1

```
bool allDone = true;
bool finished = false;
```



Hands-On Pg. 60

- Listing 2-16

2.12 Determining the Size of a Data Type

The **`sizeof`** operator gives the size in number of bytes of any data type or variable

```
double amount;

cout << "A float is stored in "
     << sizeof(float) << " bytes\n";

cout << "Variable amount is stored in "
     << sizeof(amount) << " bytes\n";
```

Hands-On Pg. 61

- Listing 2-17

2.13 More on Variable Assignments and Initialization

Assigning a value to a variable

- Assigns a value to a previously created variable
- A single variable name must appear on left side of the = symbol

```
int size;
size = 5+2;   // legal
5 = size;     // not legal
```

Variable Assignment vs. Initialization

Initializing a variable

- Gives an initial value to a variable at the time it is defined
- Some or all of the variables being defined can be initialized

```
int length = 12;
int width = 7, height = 5, area;
```

Hands-On Pg. 62

- Listing 2-18

Using `auto` in Variable Declarations

If you are initializing a variable when it is defined, the **`auto`** keyword will determine the data type to use based on the type of the initialization value. Introduced in C++ 11.

```
auto length = 12;     // length is an int
auto width = length; // also an int
auto area = 100.0;   // area is a double
```

2.14 Scope

- The scope of a variable is that part of the program where the variable may be used
- A variable cannot be used before it is defined

```
int num1 = 5;
cout << num1;    // legal
cout << num2;    // illegal
int num2 = 12;
```

Hands-On Pg. 64

- Listing 2-19

2.15 Arithmetic Operators

- Used for performing numeric calculations
- C++ has unary, binary, and ternary operators
  - unary (1 operand) `-5`
  - binary (2 operands) `13 - 7`
  - ternary (3 operands) **`exp1 ? exp2 : exp3`**

Binary Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	`ans`
`+`	addition	`ans = 7 + 3;`	`10`
`-`	subtraction	`ans = 7 - 3;`	`4`
`*`	multiplication	`ans = 7 * 3;`	`21`
`/`	division	`ans = 7 / 3;`	`2`
`%`	modulus	`ans = 7 % 3;`	`1`

/ Operator

- C++ division operator (`/`) performs integer division if both operands are integers

```
cout << 13 / 5;     // displays 2
cout <<  2 / 4;     // displays 0
```

- If either operand is floating-point, the result is floating-point

```
cout << 13 / 5.0;   // displays 2.6
cout << 2.0 / 4;    // displays 0.5
```

`%` Operator

- C++ modulus operator (`%`) computes the remainder resulting from integer division

```
cout << 9 % 2;   // displays 1
```

- `%` requires integers for both operands

```
cout << 9 % 2.0; // error
```

Hands-On Pg. 66

- Listing 2-20

Hands-On Pg. 67

- Listing 2-21

2.16 Comments

- Are used to document parts of a program
- Are written for persons reading the source code of the program
  - Indicate the purpose of the program
  - Describe the use of variables
  - Explain complex sections of code
- Are ignored by the compiler

Single-Line Comments

- Begin with `//` and continue to the end of the line

```
int length = 12; // length in inches
int width = 15;  // width in inches
int area;        // calculated area

// Calculate rectangle area
area = length * width;
```

Multi-Line Comments

- Begin with `/*` and end with `*/`
- Can span multiple lines

```
/*----------------------------
   Here's a multi-line comment
   that has two lines of text
 ----------------------------*/
```

- Can also be used as single-line comments

```
int area;   /* Calculated area */
```

Hands-On Pg. 69

- Listing 2-22

2.17 Programming Style

- Refers to the visual arrangement of the source code
- Provides information to the reader about the organization of the program
- Includes alignment of matching { and } and the use of indentation and whitespace.
- Should be used consistently throughout a program

Hands-On Pg. 70

- Listing 2-23 & 2-24 READ ONLY

Hands-On Pg. 71

- Listing 2-25

Copyright