THIRD EDITION

A First Book of C++

From Here to There

CHAPTER 2

Data Types, Declarations, and Displays

Objectives

You should be able to describe:

- Data Types
- Arithmetic Operators
- Numerical Output Using `cout`
- Variables and Declarations
- Common Programming Errors

Data Types

- The objective of all programs is to process data
- It is necessary to classify data into specific types
  - Numerical
  - Alphabetical
  - Audio
  - Video
- C++ allows only certain operations to be performed on certain types of data
  - Prevents inappropriate programming operations

Data Types (continued)

- **Data Type:** A set of values and operations that can be applied to these values
- **Example of Data Type**: Integers
  - The Values: Set of all Integer (whole) numbers
  - The Operations: Familiar mathematical and comparison operators

Data Types (continued)

- **Class**:
  - Programmer-created data type
  - Set of acceptable values and operations defined by a programmer using C++ code
- **Built-In**: Provided as an integral part of C++
  - Also known as a **primitive** type
  - Requires no external code
  - Consists of basic numerical types
  - Majority of operations are symbols (e.g. +,-,*,…)

Data Types (continued)

TABLE 2.1 *Built-In Data Type Operations*

Built-in Data Types	Operations
Integer	+, −, *, /, %, =, ==, !=, <=, >=, sizeof(), and bit operations (see Sec. 17.4)
Floating Point	+, −, *, /, =, ==, !=, <=, >=, sizeof()

Data Types (continued)

- **Literal**: Acceptable value for a data type
  - Value explicitly identifies itself
- The numbers 2, 3.6 and –8.2 are literals
  - Values are literally displayed
- The text “Hello World!” is a literal
  - Text itself is displayed
- Literals also known as **literal values** and **constants**

Integer Data Types

- C++ provides nine built-in integer data types
- Three most important
  - `int`
  - `char`
  - `bool`
- Reason for remaining types is historical
  - Originally provided for special situations
  - Difference among types based on storage requirements

The `int` Data Type

- Set of values supported are whole numbers
  - Whole numbers mathematically known as integers
- Explicit signs allowed
- Commas, decimal points, and special signs not allowed
- **Examples of `int`**:
  - Valid: 0 5 -10 +25 1000 253 -26351 +36
  - Invalid: $255.62 2,523 3. 6,243,982 1,492.89

The `int` Data Type (continued)

- Different compilers have different internal limits on the largest and smallest values that can be stored in each data type
  - Most common allocation for `int` is four bytes
- **Atomic Data Value**: a complete entity that cannot be decomposed into a smaller data type
  - All built-in data types are atomic

The `char` Data Type

- Used to store individual characters
  - Letters of the alphabet (upper and lower case)
  - Digits 0 through 9
  - Special symbols such as + $. , - !
- **Single Character Value**: letter, digit or special character enclosed in single quotes
  - Examples ‘A’ ‘$’ ‘b’ ‘7’ ‘y’ ‘!’ ‘M’ ‘q’

The `char` Data Type (continued)

- Character values stored in ASCII or Unicode codes
- **ASCII**: American Standard Code for Information Exchange
  - Provides English-language based character set plus codes for printer and display control
  - Each character code contained in one byte
  - 256 distinct codes

The `char` Data Type (continued)

- **Unicode**: Provides other language character sets
  - Each character contained in two bytes
  - Can represent 65,536 characters
- First 256 Unicode codes have same numerical value as the 256 ASCII codes

The Escape Character

- **Backslash (\ ):** the escape character
  - Special meaning in C++
  - Placed before a select group of characters, it tells the compiler to escape from normal interpretation of these characters
- **Escape Sequence**: combination of a backslash and specific characters
  - **Example**: newline escape sequence, `\n`

The Escape Character (continued)

TABLE 2.3 *Escape Sequences*

Escape Sequence	Character Represented	Meaning	ASCII Code
`\n`	Newline	Move to a new line	00001010
`\t`	Horizontal tab	Move to next horizontal tab setting	00001001
`\v`	Vertical tab	Move to next vertical tab setting	00001011
`\b`	Backspace	Move back one space	00001000
`\r`	Carriage return	(Moves the cursor to the start of the current line—used for overprinting)	00001101
`\f`	Form Feed	Issue a form feed	00001100

The Escape Character (continued)

TABLE 2.3 *Escape Sequences*

Escape Sequence	Character Represented	Meaning	ASCII Code
`\a`	Alert	Issue an alert (usually a bell sound)	00000111
`\\`	Backslash	Insert a backslash character (this is used to place an actual backslash character within a string)	01011100
`\?`	Question mark	Insert a question mark character	00111111
`\'`	Single quotation	Insert a single quote character (this is used to place an inner single quote within a set of outer single quotes)	00100111
`\"`	Double quotation mark	Insert a double quote character (this is used to place an inner double quote within a set of outer double quotes)	00100010
`\`*nnn*	Octal number	The number *nnn* (*n* is a digit) is to be considered an octal number	–
`\x`*hhhh*	Hexadecimal number	The number *hhhh* (*h* is a digit) is to be considered a hexadecimal number	–
`\0`	Null character	Insert the Null character, which is defined as having the value 0	00000000

The Escape Character (continued)

- Both ‘\n’ and “\n” contain the newline character
  - ‘\n’ is a character literal
  - “\n” is a string literal
- Both cause the same thing to happen
  - A new line is forced on the output display
- Good programming practice is to end the final output display with a newline escape sequence

The `bool` Data Type

- Represents boolean (logical) data
- Restricted to true or false values
- Often used when a program must examine a specific condition
  - If condition is true, the program takes one action, if false, it takes another action
- Boolean data type uses an integer storage code

Determining Storage Size

- C++ makes it possible to see how values are stored
- **`sizeof()`:** provides the number of bytes required to store a value for any data type
  - Built-in operator that does not use an arithmetic symbol

Signed and Unsigned Data Types

- **Signed Data Type**: stores negative, positive and zero values
- **Unsigned Data Type**: stores positive and zero values
  - Provides a range of positive values double that of unsigned counterparts
- `char` and `bool` are unsigned data types
  - No codes for storing negative values
- Some applications only use unsigned data types
  - **Example:** date applications in form *yearmonthday*

Signed and Unsigned Data Types (continued)

TABLE 2.4 *Integer Data Type Storage*

Name of Data Type	Storage Size (in bytes)	Range of Values
`char`	1	256 characters
`bool`	1	true (which is considered as any positive value) and false (which is a zero)
`short int`	2	–32,768 to +32,767
`unsigned short int`	2	0 to 65,535
`int`	4	–2,147,483,648 to +2,147,483,647
`unsigned int`	4	0 to 4,294,967,295
`long int`	4	–2,147,483,648 to +2,147,483,647
`unsigned long int`	4	0 to 4,294,967,295

Floating-Point Types

- The number zero or any positive or negative number that contains a decimal point
  - Also called real number
  - Examples: +10.625   5.   -6.2   3521.92   0.0
  - 5. and 0.0 are floating-point, but same values without a decimal (5, 0) would be integers
- C++ supports three floating-point types:
  - `float`, `double`, `long double`
  - Different storage requirements for each

Floating-Point Types (continued)

- **Precision**: either the accuracy or number of significant digits
  - A computer programming definition
  - Significant digits = number of correct digits + 1
- Significant digits in a number may not correspond to the number of digits displayed
  - **Example**: if 687.45678921 has five significant digits, it is only accurate to the value 687.46

Exponential Notation

- Floating-point numbers can be written in exponential notation
  - Similar to scientific notation
  - Used to express very large and very small values in compact form

Decimal Notation	Exponential Notation	Scientific Notation
1625.	1.625e3	1.625×10^{3}
63421.	6.3421e4	6.3421×10^{4}
.00731	7.31e–3	7.31×10^{-3}
.000625	6.25e–4	6.25×10^{-4}

Arithmetic Operators

Operation	Operator
Addition	+
Subtraction	–
Multiplication	*
Division	/
Modulus	%

- **Binary operators**: require two operands
  - Can be a literal or an identifier
- **Binary Arithmetic Expression**:
  - **Format**: *`literalValue operator literalValue`*

Arithmetic Operators (continued)

- **Mixed-mode Expression**: arithmetic expression containing integer and non-integer operands
- **Rule for evaluating arithmetic expressions**:
  - **Both operands are integers**: result is integer
  - **One operand is floating-point**: result is floating-point
- Overloaded Operator: a symbol that represents more than one operation
  - Execution depends on types of operands

Integer Division

- Division of two integers yields an integer
  - Integers cannot contain a fractional part - results may seem strange
  - **Example**: integer 15 divided by integer 2 yields the integer result 7
- **Modulus Operator (%):** captures the remainder
  - Also called the **remainder operator**
  - **Example**: 9 % 4 is 1 (remainder of 9/4 is 1)

Negation

- A unary operation that negates (reverses the sign of) the operand
- Uses same sign as binary subtraction (-)

Operator Precedence and Associativity

- Rules for expressions with multiple operators
  - Two binary operators cannot be placed side by side
  - Parentheses may be used to form groupings
  - Expressions within parentheses are evaluated first
  - Sets of parentheses may be enclosed by other parentheses
  - Parentheses cannot be used to indicate multiplication (multiplication operator (*) must be used)

Operator Precedence and Associativity (continued)

TABLE 2.7 *Operator Precedence and Associativity*

Operator	Associativity
unary –	right to left
* / %	left to right
+ –	left to right

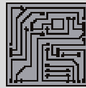
Numerical Output Using `cout`

- `cout` allows for display of result of a numerical expression
  - Display is on standard output device
- **Example:**
  - `cout << “The total of 6 and 15 is ” << (6 + 15)`
  - Statement sends string and value to `cout`
  - **String**: “The total of 6 and 15 is ”
  - **Value**: value of the expression 6 + 15
  - **Display produced**: The total of 6 and 15 is 21

Formatted Output

- A program must present results attractively
- **Field width manipulators**: control format of numbers displayed by `cout`
  - Manipulators included in the output stream

Formatted Output (continued)



Program 2.3

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

int main()
{
  cout << 6 << endl
       << 18 << endl
       << 124 << endl
       << "---\n"
       << (6+18+124) << endl;

  return 0;
}
```

The output of Program 2.3 is

```
6
18
124
---
148
```

Formatted Output (continued)

TABLE 2.9 *Effect of Format Manipulators*

Manipulators	Number	Display	Comments
`setw(2)`	3	`| 3|`	Number fits in field
`setw(2)`	43	`|43|`	Number fits in field
`setw(2)`	143	`|143|`	Field width ignored
`setw(2)`	2.3	`|2.3|`	Field width ignored

`setw(5)`
`fixed`
`setprecision(2)`
	2.366	`| 2.37|`	Field width of 5 with 2 decimal digits

`setw(5)`
`fixed`
`setprecision(2)`
	42.3	`|42.30|`	Number fits in field with specified precision

`setw(5)` `setprecision(2)`	142.364	`|1.4e+002|`	Field width ignored and scientific notation used with the `setprecision` manipulator specifying the total number of significant digits (integer plus fractional)

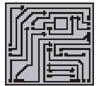
Variables and Declarations

- Symbolic names used in place of memory addresses
  - Symbolic names are called variables
  - These variables refer to memory locations
  - The value stored in the variable can be changed
  - Simplifies programming effort
- **Assignment statement**: assigns a value to a variable
  - **Format**: *variable name = value assigned;*
  - **Example**: `num1 = 45;`

Declaration Statements

- Names a variable, specifies its data type
  - **General form**: *dataType variableName;*
  - **Example**: `int sum;` Declares `sum` as variable which stores an integer value
- Declaration statements can be placed anywhere in function
  - Typically grouped together and placed immediately after the function's opening brace
- Variable must be declared before it can be used

Declaration Statements (continued)



Program 2.8

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

int main()
{
  char ch;    // this declares a character variable

  ch = 'a';   // store the letter a into ch
  cout << "The character stored in ch is " << ch << endl;
  ch = 'm';   // now store the letter m into ch
  cout << "The character now stored in ch is "<< ch << endl;

  return 0;
}
```

When Program 2.8 is run, the output produced is

```
The character stored in ch is a
The character now stored in ch is m
```

Multiple Declarations

- Variables with the same data type can be grouped together and declared in one statement
  - **Format**: *dataType variableList*;
  - **Example**: `double grade1, grade2, total, average;`
- **Initialization**: using a declaration statement to store a value in a variable
  - Good programming practice is to declare each initialized variable on a line by itself
  - **Example:** `double grade2 = 93.5;`

Reference Variables

- After a variable is declared, it may be given additional names
  - Use a **reference declaration**
  - **Format**: *dataType &newName* = *existingName*;
  - **Example**: `double &sum = total;`
  - **Multiple references**: each reference name must be preceded by an ampersand symbol
  - **Example**: `double &sum = total, &mean = average;`

Specifying Storage Allocation

- Each data type has its own storage requirements
  - Computer must know variable's data type to allocate storage
  - **Definition statements**: declaration statements used for the purpose of allocating storage

Specifying Storage Allocation (continued)

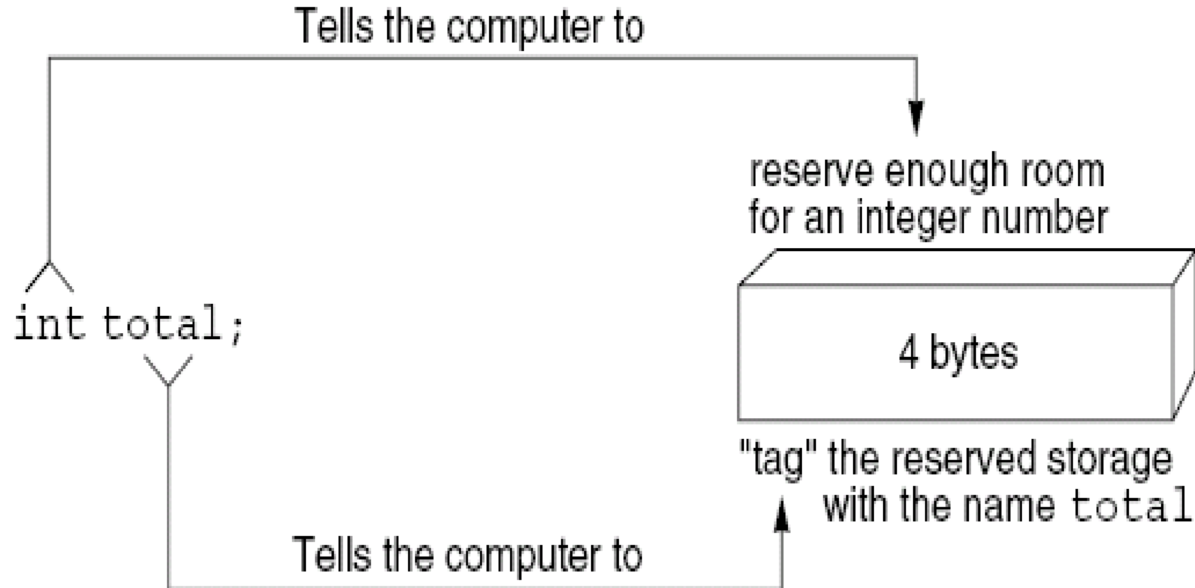


FIGURE 2.8a *Defining the Integer Variable Named* `total`

Common Programming Errors

- Forgetting to declare all variables used in a program
- Attempting to store one data type in a variable declared for a different type
- Using a variable in an expression before the variable is assigned a value
- Dividing integer values incorrectly

Common Programming Errors (continued)

- Mixing data types in the same expression without clearly understanding the effect produced
  - It is best not to mix data types in an expression unless a specific result is desired
- Forgetting to separate individual data streams passed to `cout` with an insertion (“put to”) symbol

Summary

- Four basic types of data recognized by C++:
  - Integer, floating-point, character, boolean
- `cout` object can be used to display all data types
- Every variable in a C++ program must be declared as the type of variable it can store
- Reference variables can be declared that associate a second name to an existing variable

Summary (continued)

- A simple C++ program containing declaration statements has the format:

```
#include <iostream>
using namespace std;
int main()
{
  declaration statements;

  other statements;

  return 0;
}
```

Summary (continued)

- **Declaration Statements**: inform the compiler of function's valid variable names
- **Definition Statements**: declaration statements that also cause computer to set aside memory locations for a variable
- **`sizeof()` operator**: determines the amount of storage reserved for a variable