Chapter 2: Introduction to C++

Starting Out with C++
Early Objects
Seventh Edition

by Tony Gaddis, Judy Walters,
and Godfrey Muganda

Topics

Topics (continued)

2.1 The Parts of a C++ Program

Statement	Purpose
`// sample C++ program`	
`#include <iostream>`	
`using namespace std;`	
`int main()`	
`{`	
`  cout << "Hello, there!";`	
`  return 0;`	
`}`	

←

Statement	Purpose
`// sample C++ program`	comment

`// sample C++ program`

Comments are ignored by the compiler.

Statement	Purpose
`// sample C++ program`	comment
`#include <iostream>`	preprocessor directive

`#include <iostream>`

Because this line starts with a #, it is called a *preprocessor directive.*

The preprocessor reads your program before it is compiled and only executes those lines beginning with a # symbol.

The **`#include`** directive causes the preprocessor to include the contents of another file in the program. The word inside the brackets, iostream, is the name of the file that is to be included.

←

Statement	Purpose
`// sample C++ program`	comment
`#include <iostream>`	preprocessor directive

The iostream file contains code that allows a C++ program to display output on the screen and read input from the keyboard.

Because this program uses cout to display screen output, the iostream file must be included.

The iostream file is called a *header file*, so it should be included at the head, or top, of the program.

Statement	Purpose
`// sample C++ program`	comment
`#include <iostream>`	preprocessor directive
`using namespace std;`	which namespace to use

using namespace std;

In order for a program to use the entities in iostream, it must have access to the std namespace. C++ uses *namespaces* to organize the names of program entities.

using namespace std; declares that the program will be accessing entities whose names are part of the namespace called std.

Statement	Purpose
`// sample C++ program`	comment
`#include <iostream>`	preprocessor directive
`using namespace std;`	which namespace to use
`int main()`	beginning of function named `main`

`int main()`

This marks the beginning of a function. A *function* can be thought of as a group of one or more programming statements that collectively has a name.

The name of this function is *main*, and the set of parentheses that follows the name indicate that it is a function. The word int stands for "integer." It indicates that the function sends an integer value back to the operating system when it is finished executing.

Although most C++ programs have more than one function, every C++ program must have a function called main. It is the starting point of the program.

Statement	Purpose
`// 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`

`{`

`}` This is called a left-brace, or an opening brace, and it is associated with the beginning of the function main. All the statements that make up a function are enclosed in a set of braces. If you look at the third line down from the opening brace you'll see the closing brace. Everything between the two braces is the contents of the function main.

Statement	Purpose
`// 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

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

To put it simply, this line displays a message on the screen. You will read more about cout and the << operator later in this chapter.

The message "Hello, there!" is printed without the quotation marks.

In programming terms, the group of characters inside the quotation marks is called a *string literal* or *string constant.*

Statement	Purpose
`// 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

At the end of the line is a semicolon. Just as a period marks the end of a sentence in English, a semicolon marks the end of a complete statement in C++.

Preprocessor directives, like #include statements, simply end at the end of the line and never require semicolons.

The beginning of a function, like int main(), is not a complete statement, so you don't place a semicolon there either.

Statement	Purpose
`// 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 the operating system

```
return 0;
```

This statement sends the integer value 0 back to the operating system upon the program's completion.
The value 0 usually indicates that a program executed successfully.

Statement	Purpose
`// 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 the operating system
`}`	end of block for `main`

```
}
```

This brace marks the end of the main function. Since main is the only function in this program, it also marks the end of the program.

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 function
`{ }`	Open, Close Braces	Encloses a group of statements
`" "`	Open, Close Quote Marks	Encloses 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!";
```

- Can use **<<** to send multiple items to cout

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

Or

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

↓

Hello, 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";
```

Hello, there!
_

```
int main()
{
   cout << "The following items were top sellers";
   cout << "during the month of June:";
   cout << "Computer games";
   cout << "Coffee";
   cout << "Aspirin";
   return 0;
}
```

Program Output

```
The following items were top sellersduring the month of JuneComputer gamesCoff
eeAspirin
```

```
int main()
{
   cout << "The following items were top sellers" << endl;
   cout << "during the month of June:" << endl;
   cout << "Computer games" << endl;
   cout << "Coffee" << endl;
   cout << "Aspirin" << endl;
   return 0;
}
```

```
The following items were top sellers
during the month of June:
Computer games
Coffee
Aspirin
```

```
int main()
{
   cout << "The following items were top sellers\n";
   cout << "during the month of June:\n";
   cout << "Computer games\nCoffee";
   cout << "\nAspirin\n";
   return 0;
}
```

Program Output

```
The following items were top sellers
during the month of June:
Computer games
Coffee
Aspirin
```

Table 2-2 Common Escape Sequences

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, or move left one position.
\r	Return	Causes the cursor 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.

2.4 Standard and Prestandard C++

Older-style C++ programs

- Use `.h` at end of header files
  `#include <iostream.h>`
- Do not use `using namespace` convention
- May not compile with a standard C++ compiler

2.5 Variables, Constants, and the Assignment Statement

- Variable
  - Has a name and a type of data it can hold

```
data type        variable name
char letter;
```

  - Is used to reference a location in memory where a value can be stored
  - Must be defined before it can be used
  - The value that is stored can be changed, *i.e.*, it can "vary"

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;      // age is 17
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 variable on the left

```
item = 12;
```


Constants

- Constant
  - Data item whose value does not change during program execution
  - Is also called a literal

```
 'A'      // character constant
"Hello"   // string literal
12        // integer constant
3.14      // floating-point constant
```


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

Variable names are examples of identifiers. You may choose your own variable names as long as you do not use any of the C++ *key words*.

Table 2-4 The C++ Key Words

and	continue	goto	public	try
and_eq	default	if	register	typedef
asm	delete	inline	reinterpret_cast	typeid
auto	do	int	return	typename
bitand	double	long	short	union
bitor	dynamic_cast	mutable	signed	unsigned
bool	else	namespace	sizeof	using
break	enum	new	static	virtual
case	explicit	not	static_cast	void
catch	export	not_eq	struct	volatile
char	extern	operator	switch	wchar_t
class	false	or	template	while
compl	float	or_eq	this	xor
const	for	private	throw	xor_eq
const_cast	friend	protected	true	

Legal Identifiers

Here are some specific rules that must be followed with all identifiers.

- The first character must be one of the letters a through z, A through Z, or an underscore character (_).
- After the first character you may use the letters a through z or A through Z, the digits 0 through 9, or underscores.
- Uppercase and lowercase characters are distinct. This means ItemsOrdered is not the same as itemsordered.

Checkpoint

2.7 Which of the following are illegal variable names, and why?

```
x
99bottles
july97
thesalesFigureForFiscalYear98
rkd
grade_report
```

s with a digit.

e letters, digits, or unders

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
`totalSales`		
`total_Sales`		
`total.Sales`		
`4thQtrSales`		
`totalSale$`		

2.7 Integer Data Types

- Designed to hold whole numbers
- Can be **`signed`** or **`unsigned`**

  `12` **`-6`** **`+3`**
- Available in different sizes (*i.e.*, number of bytes): **`short`**, **`int`**, and **`long`**

↓

- Size of **`short`** ≤ size of **`int`** ≤ size of **`long`**

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

Program 2-7

```
// This program has a variable.
#include <iostream>
using namespace std;

int main()
{
   int number;

   number = 5;
   cout << "The value in number is " << number << endl;
   return 0;
}
```

Program Output

```
The value in number is 5
```

Program 2-8

```
// This program has a variable.
#include <iostream>
using namespace std;

int main()
{
   int number;

   number = 5;
   cout << "The value in number is " << "number" << endl;
   return 0;
}
```

Program Output

Literals

A variable is called a "variable" because its value may be changed.

A literal, on the other hand, is a value that does not change during the program's execution. Program 2-9 contains both literals and a variable.

Table 2-3

Literal	Type of Literal
20	Integer literal
"Today we sold "	String literal
"bushels of apples.\n"	String literal
0	Integer literal

Program 2-9

```
// This p
#include
using nam

int main()
{
   int apples;

   apples = 20;
   cout << "Today we sold " << apples << " bushels of apples.\n";
   return 0;
}
```

Program Output

```
Today we sold 20 bushels of apples.
```

Integral Constants

- To store an integer constant in a long memory location, put '**`L`**' at the end of the number: `1234L`
- Constants that begin with '0' (zero) are octal, or base 8: `075`

↓

- Constants that begin with '**`0x`**' are hexadecimal, or base 16: `0x75A`

Table 2-6 Integer Data Types, Sizes, and Ranges

Data Type	Size	Range
short	2 bytes	–32,768 to +32,767
unsigned short	2 bytes	0 to +65,535
int	4 bytes	–2,147,483,648 to +2,147,483,647
unsigned int	4 bytes	0 to 4,294,967,295
long	4 bytes	–2,147,483,648 to +2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Here are some examples of variable definitions:

```
int days;
unsigned speed;
short month;
unsigned short amount;
long deficit;
unsigned long insects;
```

2.8 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

```
char letter = 'C';
```

MEMORY

letter

67

Printable ASCII Characters

Dec	Hex	Oct	Character
32	20	40	(Space)
33	21	41	!
34	22	42	"
35	23	43	#
36	24	44	$
37	25	45	%
38	26	46	&
39	27	47	'
40	28	50	(
41	29	51	)
42	2a	52	*
43	2b	53	+
44	2c	54	,
45	2d	55	-
46	2e	56	.
47	2f	57	/
48	30	60	0
49	31	61	1
50	32	62	2
51	33	63	3
52	34	64	4
53	35	65	5
54	36	66	6
55	37	67	7
56	38	70	8
57	39	71	9
58	3a	72	:
59	3b	73	;
60	3c	74	<
61	3d	75	=
62	3e	76	>
63	3f	77	?
64	40	100	@
65	41	101	A
66	42	102	B
67	43	103	C

String Constant

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

  `"Hello"`

- Stored with the null terminator, `\0`, at end

H	e	l	L	O	\0

- Is comprised of characters between the `" "`

A character or a string constant?

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

  `'C'`

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

  `"Hello, there!"`

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

  `"C"`

2.9 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 << name;
  ```

2.10 Floating-Point Data Types

- Designed to hold real numbers
  `12.45` `-3.8`
- Available in different sizes (number of bytes): **`float`**, **`double`**, and **`long double`**
- Size of **`float`** ≤ size of **`double`**
  ≤ size of **`long double`**

Floating-point Constants

- 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`**

Assigning Floating-point Values to Integer Variables

If a floating-point value 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
```


2.11 The **`bool`** Data Type

- Represents values that are **`true`** or **`false`**
- **`bool`** values are stored as short integers

↓

- **`false`** is represented by 0, **`true`** by 1

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

allDone	finished
1	0

2.12 Determining the Size of a Data Type

The **`sizeof`** operator gives the size 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";
```


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;    // legal
5 = size;    // not legal
```

Variable Assignment vs. Initialization

- Initializing a variable
  - Gives an initial value to a variable at the time it is created
  - Can initialize some or all variables at definition

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


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 a;
cin >> a;   // legal
cin >> b;   // illegal
int b;
```


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
```

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 `//` through to the end of 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
  ----------------------------*/
```

- Can also be used as single-line comments

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


Program 2-25

```
#include <iostream>
using namespace std;int main();{double shares=220.0;
double avgPrice=14.67;cout<<"There were "<<shares
<<" shares sold at $"<<avgPrice<<" per share.\n";
return 0;}
```

Program 2-26

```
// This example is much more readable than Program 2-25.
#include <iostream>
using namespace std;

int main()
{
   double shares = 220.0;
   double avgPrice = 14.67;

   cout << "There were " << shares << " shares sold at $";
   cout << avgPrice << " per share.\n";
   return 0;
}
```

Program Output

```
There were 220 shares sold at $14.67 per share.
```

6. What will the following programs print on the screen?

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

int main()
{
   int freeze = 32, boil = 212;
   freeze = 0;
   boil = 100;
   cout << freeze << endl << boil << endl;
   return 0;
}
```

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

int main()
{
   int x = 3, y = 2;
   x = y * 4;
   cout << x << endl << y << endl;
   return 0;
}
```

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

int main()
{
   cout << "I am the incredible";
   cout << "computing\nmachine";
   cout << "\nand I will\namaze\n";
   cout << "you.";
   return 0;
}
```

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

int main()
{
   cout << "Be careful\n";
   cout << "This might/n be a trick ";
   cout << "question\n";
   return 0;
}
```

HW

Lab 1 LM_Chapter_01_Intro_8e.pdf

Lab 2 LM_Chapter_02_CppIntro_8e.pdf

1. Students should read the Pre-lab Reading Assignment before coming to lab.
2. Students should complete the Pre-lab Writing Assignment before coming to lab. (photocopy or copy/paste, answer then print and bring to class.)

Algorithm Workbench

25. Convert the following pseudocode to C++ code. Be sure to define the appropriate variables.

    Store 20 in the `speed` variable.
    Store 10 in the `time` variable.
    Multiply `speed` by time and store the result in the `distance` variable.
    Display the contents of the `distance` variable.

26. Convert the following pseudocode to C++ code. Be sure to define the appropriate variables.

    Store 172.5 in the `force` variable.
    Store 27.3 in the `area` variable.
    Divide area by `force` and store the result in the `pressure` variable.
    Display the contents of the `pressure` variable.

Programming Challenges

1. **Sum of Two Numbers**

   Write a program that stores the integers 62 and 99 in variables, and stores the sum of these two in a variable named `total`.

2. **Sales Prediction**

   The East Coast sales division of a company generates 62 percent of total sales. Based on that percentage, write a program that will predict how much the East Coast division will generate if the company has $4.6 million in sales this year.

3. **Sales Tax**

   Write a program that will compute the total sales tax on a $52 purchase. Assume the state sales tax is 4 percent and the county sales tax is 2 percent.

4. **Restaurant Bill**

   Write a program that computes the tax and tip on a restaurant bill for a patron with a $44.50 meal charge. The tax should be 6.75 percent of the meal cost. The tip should be 15 percent of the total after adding the tax. Display the meal cost, tax amount, tip amount, and total bill on the screen.

17. **Stock Commission**

    Kathryn bought 600 shares of stock at a price of $21.77 per share. She must pay her stock broker a 2 percent commission for the transaction. Write a program that calculates and displays the following:

    - The amount paid for the stock alone (without the commission)
    - The amount of the commission
    - The total amount paid (for the stock plus the commission)

18. **Energy Drink Consumption**

    A soft drink company recently surveyed 12,467 of its customers and found that approximately 14 percent of those surveyed purchase one or more energy drinks per week. Of those customers who purchase energy drinks, approximately 64 percent of them prefer citrus flavored energy drinks. Write a program that displays the following:

    - The approximate number of customers in the survey who purchase one or more energy drinks per week
    - The approximate number of customers in the survey who prefer citrus flavored energy drinks