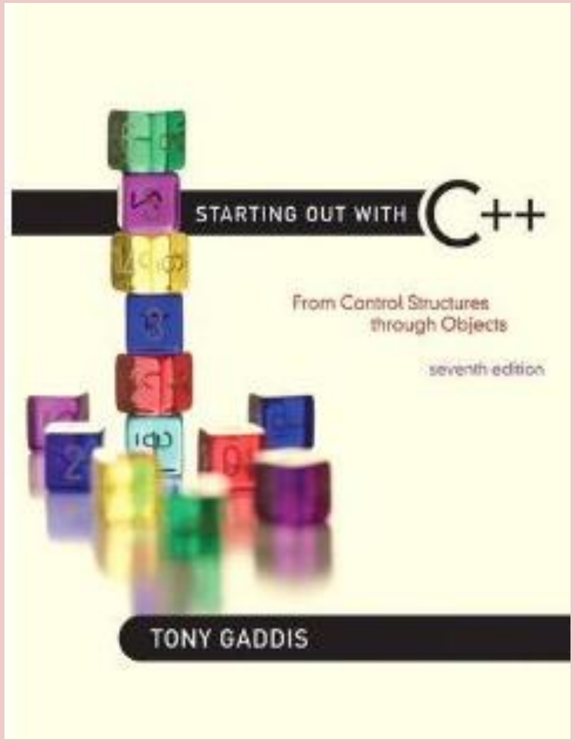
Software Design & Programming I



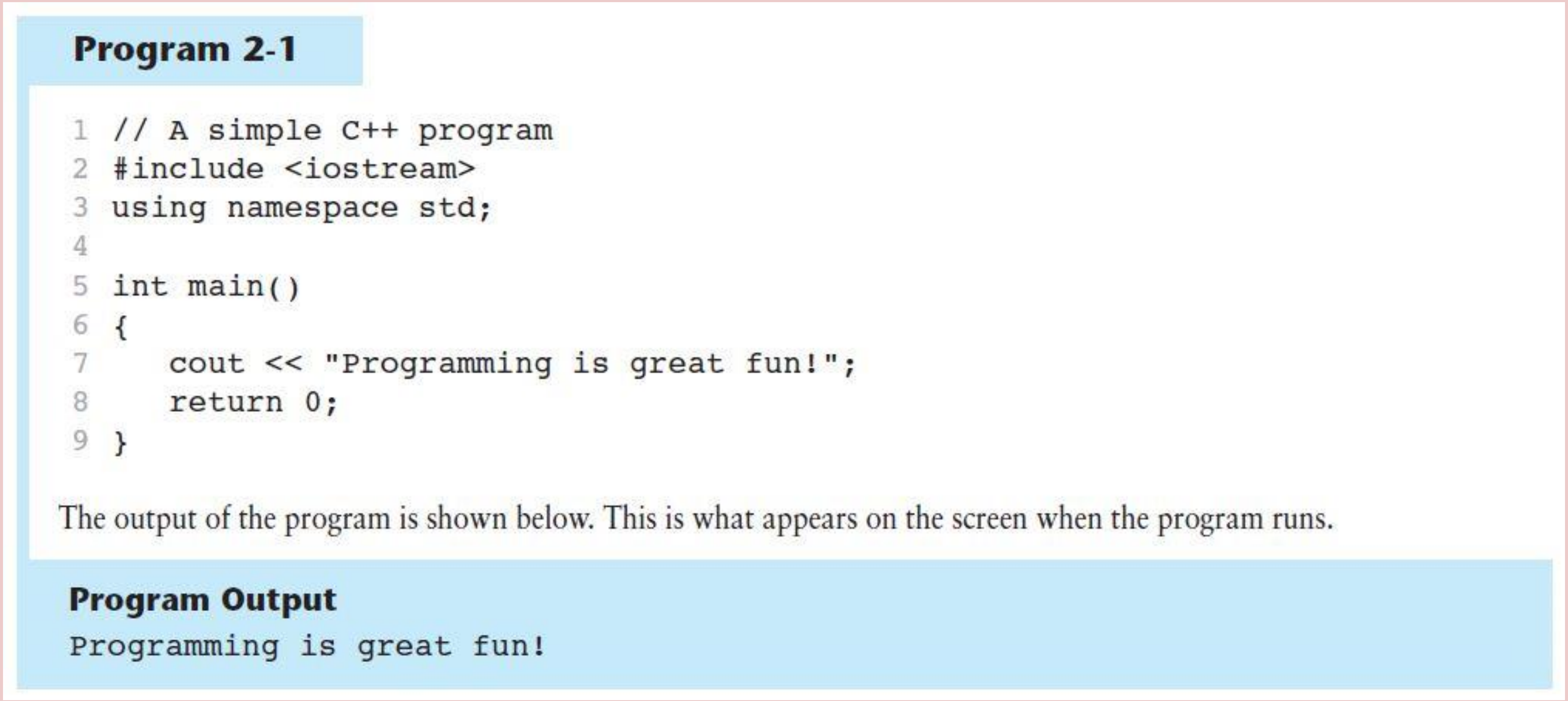
Starting Out with C++ (From Control Structures through Objects) 7th Edition
Written by: Tony Gaddis
Pearson - Addison Wesley
ISBN: 13-978-0-132-57625-3

Chapter 2

Introduction to C++

The Parts of a C++ Program

Every C++ program has an anatomy. Unlike human anatomy, the parts of C++ programs are not always in the same place. Nevertheless, the parts are there and your first step in learning C++ is to learn what they are.



Program 2-1

```cpp
// A simple C++ program
#include <iostream>
using namespace std;

int main()
{
   cout << "Programming is great fun!";
   return 0;
}
```

The output of the program is shown below. This is what appears on the screen when the program runs.

Program Output

```
Programming is great fun!
```

// A simple C++ program

The // marks the beginning of a comment. The compiler ignores everything from the double-slash to the end of the line.

#include <iostream>

This line must be included in a C++ program in order to get input from the keyboard or print output to the screen. Since the cout statement (on line 7) will print output to the computer screen, we need to include this line. When a line begins with a # it indicates it is a preprocessor directive. The preprocessor reads your program before it is compiled and only executes those lines beginning with a # symbol.

using namespace std;

The statement **using namespace std;** declares that the program will be accessing entities whose names are part of the namespace called std. The program needs access to the std namespace because every name created by the iostream file is part of that namespace.

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 has a name. The name of this function is main, and the set of parentheses that follows the name indicates 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.

Important Note:

1. 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.
2. C++ is a case-sensitive language. That means it regards uppercase letters as being entirely different characters than their lowercase counterparts. In C++, the name of the function main must be written in all lowercase letters. C++ doesn’t see “main” the same as “Main” or “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.

Important Note:

1. Make sure you have a closing brace for every opening brace in your program.

cout << “Programming is fun!”;

This line displays a message on the screen. You will read more about cout and the << operator later in this chapter. The message “Programming is great fun!” is printed without the quotation marks. In programming terms, the group of characters inside the quotation marks is called a string literal, a string constant, or simply a string.

Note: The line with cout ends with a semicolon. Just as a period marks the end of a sentence, a semicolon is required to mark the end of a complete statement in C++. But many C++ lines do not end with semicolons. Some of these include comments, preprocessor directives, and the beginning of functions.

Here are some examples of when to use, and not use, semicolons.

```
// Semicolon examples              // This is a comment
# include <iostream>               // This is a preprocessor directive
int main()                         // This begins a function
cout << "Hello";                   // This is a complete statement
```

```
return 0;
```

This 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.

}

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

Special Characters used in C++

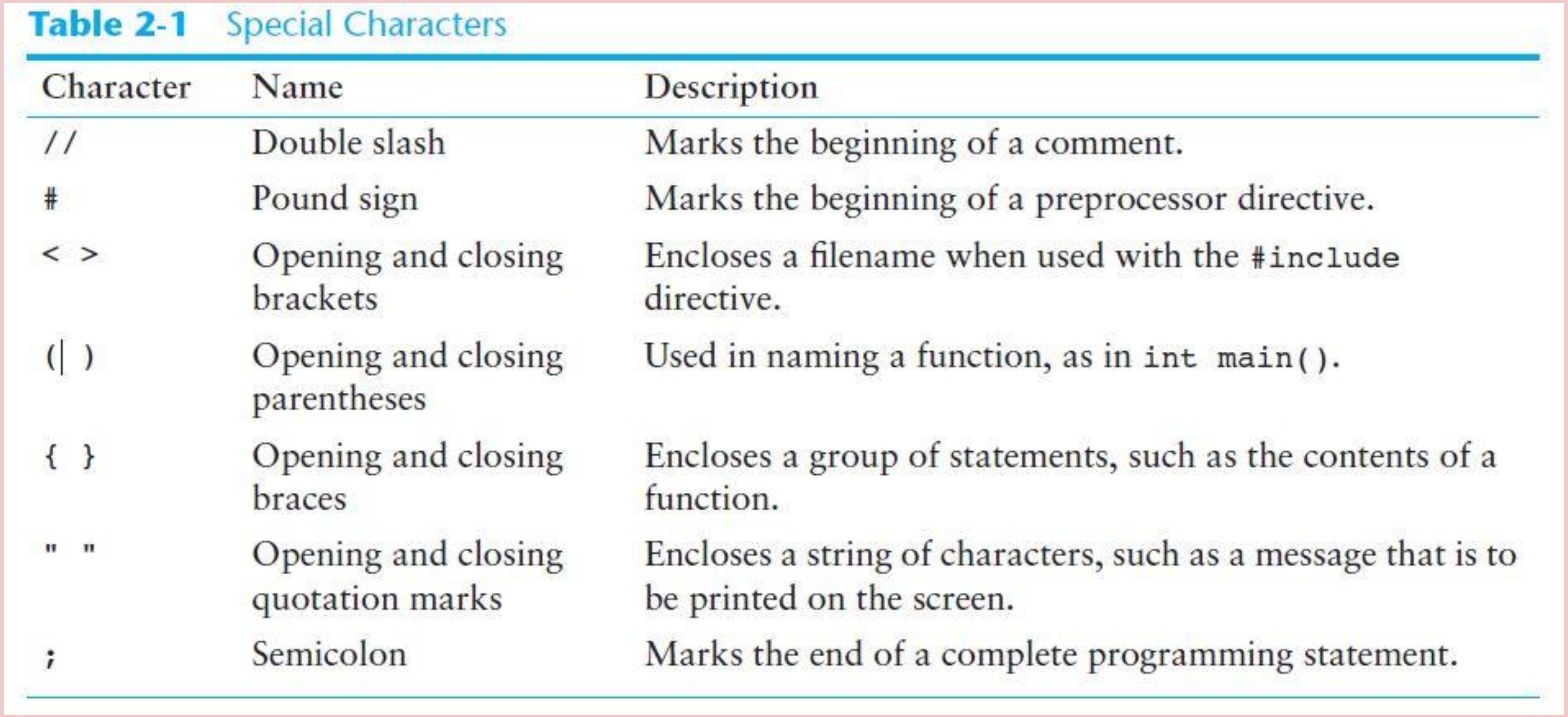


Table 2-1 Special Characters

Character	Name	Description
`//`	Double slash	Marks the beginning of a comment.
`#`	Pound sign	Marks the beginning of a preprocessor directive.
`< >`	Opening and closing brackets	Encloses a filename when used with the `#include` directive.
`( )`	Opening and closing parentheses	Used in naming a function, as in `int main()`.
`{ }`	Opening and closing braces	Encloses a group of statements, such as the contents of a function.
`" "`	Opening and closing quotation marks	Encloses a string of characters, such as a message that is to be printed on the screen.
`;`	Semicolon	Marks the end of a complete programming statement.

The cout Object

The simplest type of screen output that a program can display is console output, which is merely plain text. The word console is an old computer term. It comes from the days when a computer operator interacted with the system by typing on a terminal. The terminal, which consisted of a simple screen and keyboard, was known as the console.

On modern computers, running graphical operating systems such as Windows or Mac OS X, console output is usually displayed in a window. C++ provides an object named cout that is used to produce console output. (You can think of the word cout as meaning console output.

cout is classified as a stream object, which means it works with streams of data. To print a message on the screen, you send a stream of characters to cout.

Ex:

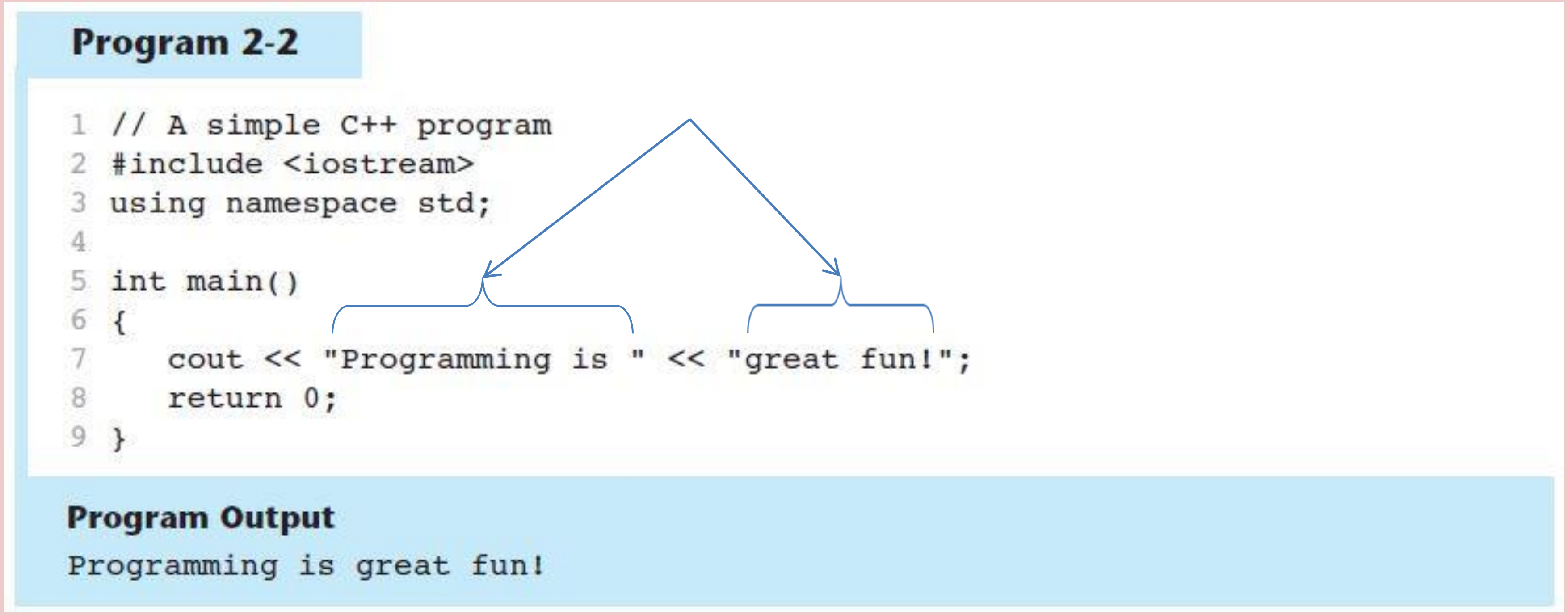
```
cout << "Programming is great fun!";
```

The << operator is used to send the string “Programming is great fun!” to cout. When the <<

symbol is used this way, it is called the stream-insertion operator. The item immediately to

the right of the operator is sent to cout and then displayed on the screen.

cout Examples



Program 2-2

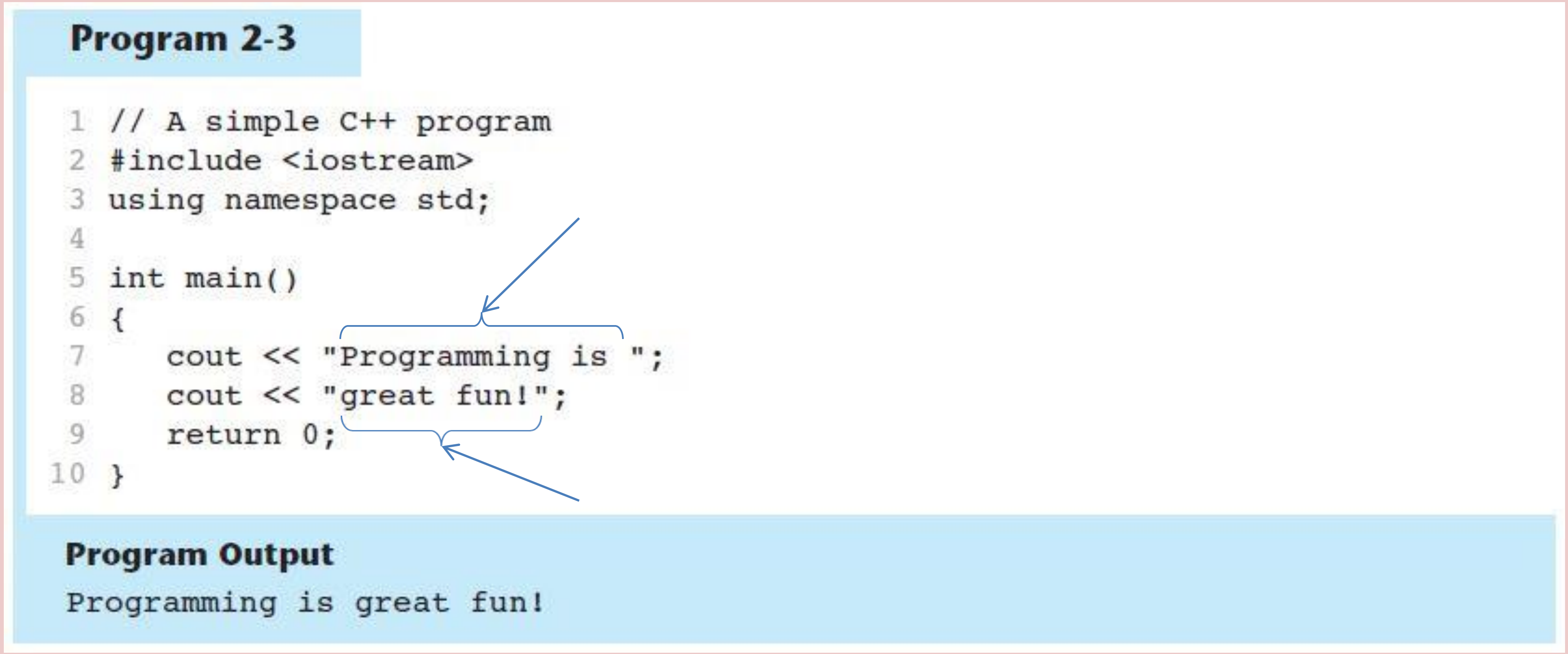
```
// A simple C++ program
#include <iostream>
using namespace std;

int main()
{
   cout << "Programming is " << "great fun!";
   return 0;
}
```

Program Output

```
Programming is great fun!
```

As you can see, the stream-insertion operator can be used to send more than one item to cout.

cout Examples



Program 2-3

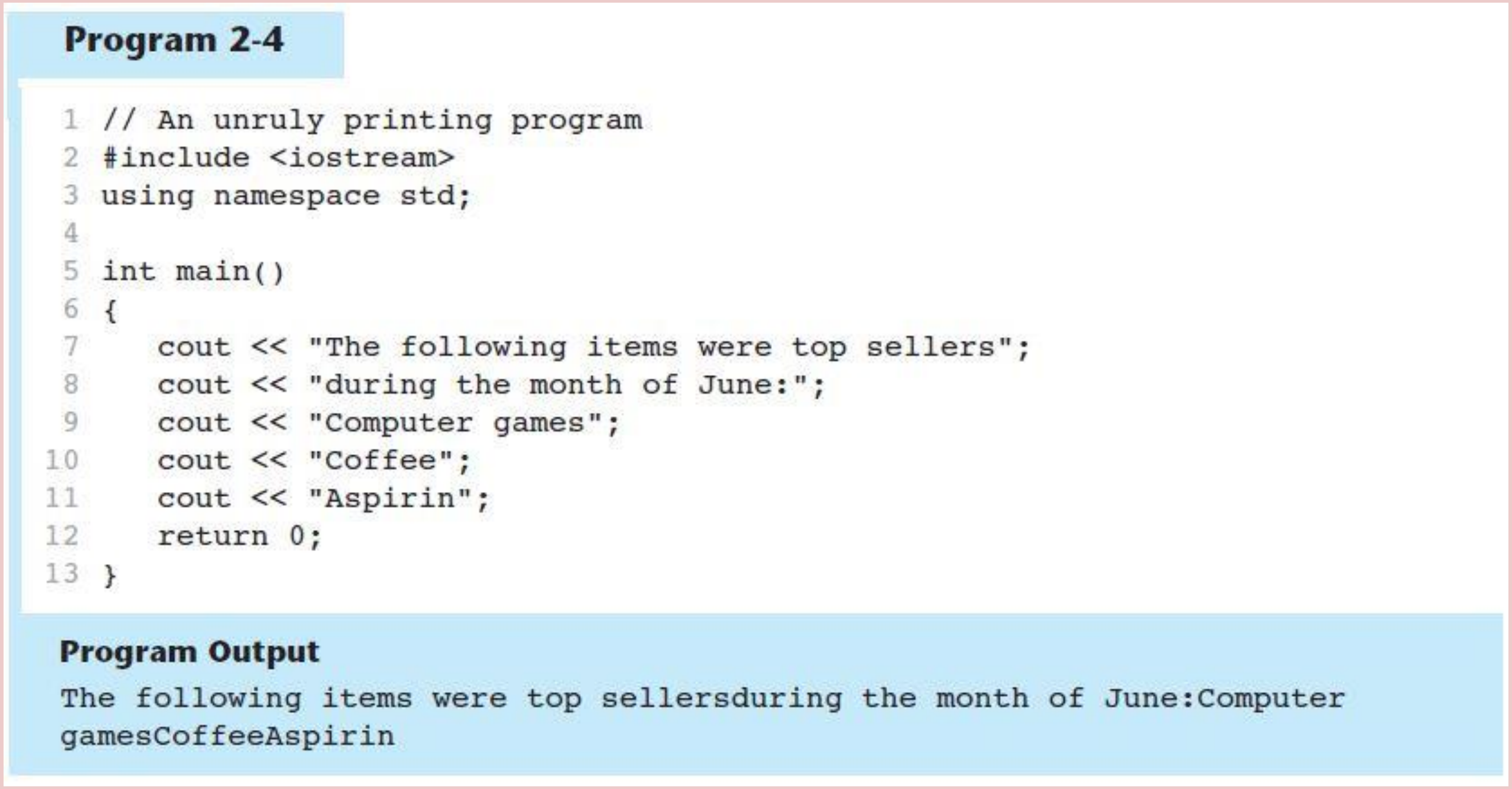
```
// A simple C++ program
#include <iostream>
using namespace std;

int main()
{
    cout << "Programming is ";
    cout << "great fun!";
    return 0;
}
```

Program Output

```
Programming is great fun!
```

As you can see, the cout statement can be used in difference ways to produce the same output.

cout Examples



Program 2-4

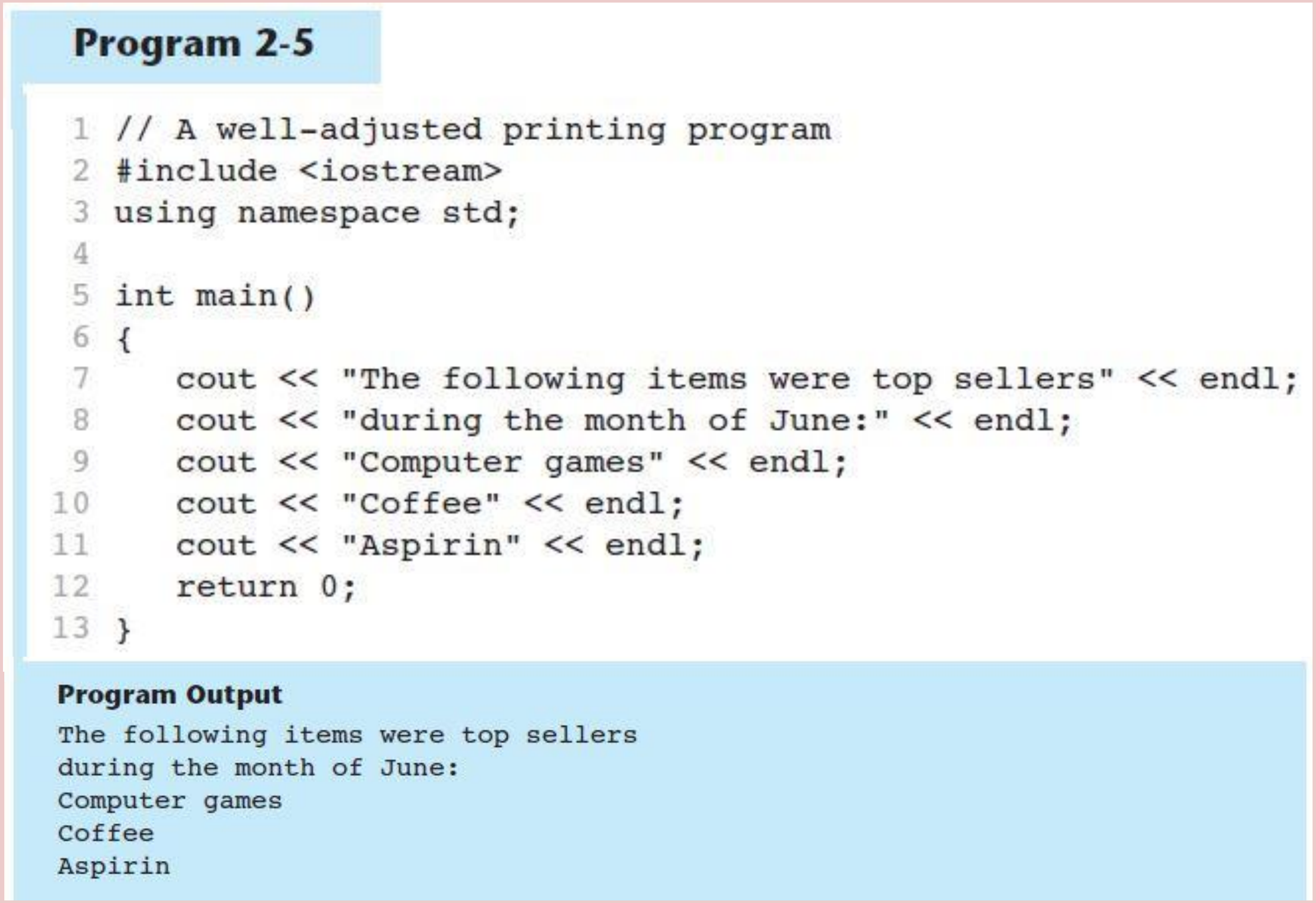
```
// An unruly printing program
#include <iostream>
using namespace std;

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 June:Computer
gamesCoffeeAspirin
```

Notice that the layout of the actual output looks nothing like the arrangement of the strings in the source code.

cout Examples



Program 2-5

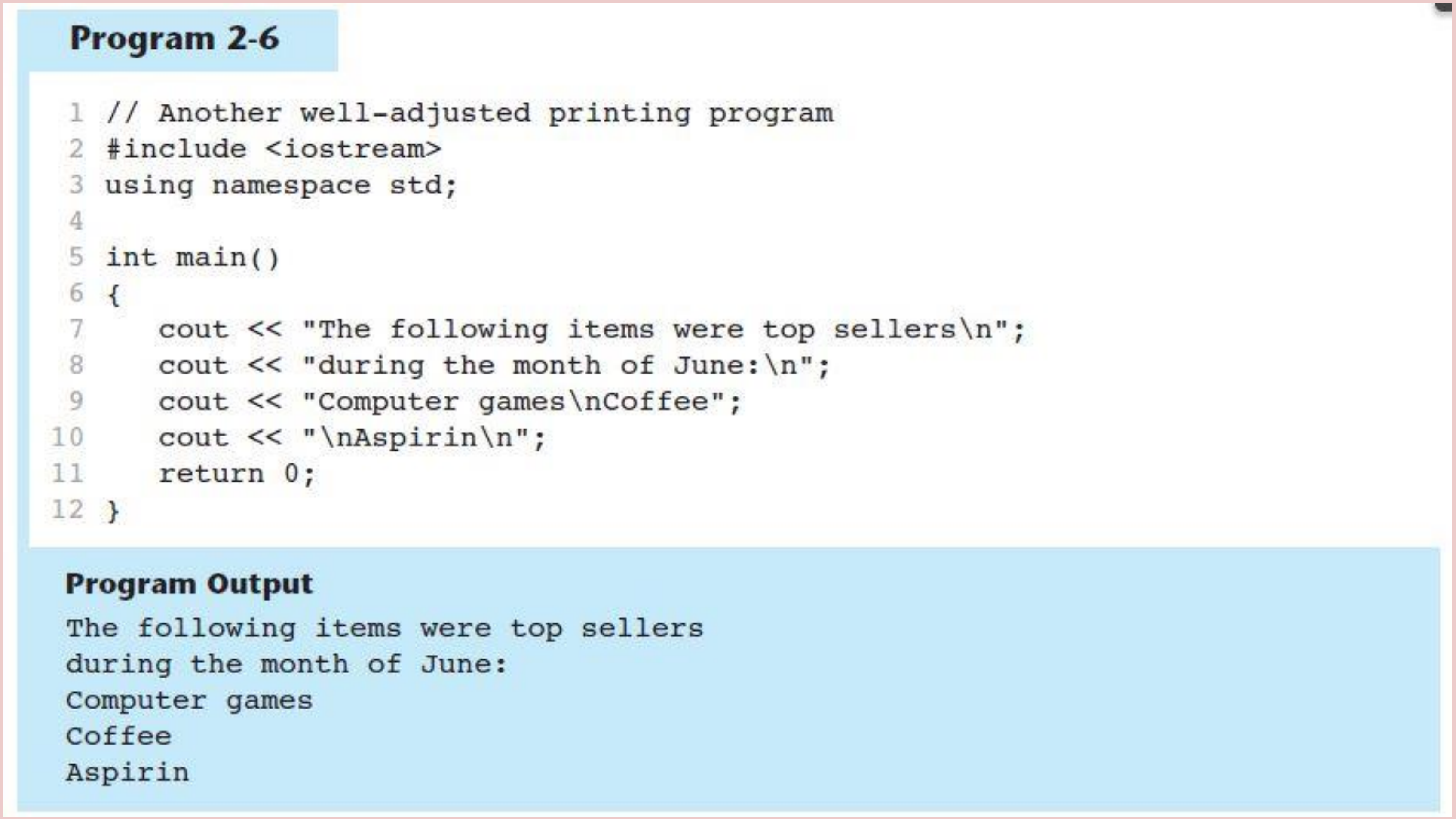
```cpp
// A well-adjusted printing program
#include <iostream>
using namespace std;

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;
}
```

Program Output

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

The endl (end “L”) is used to create a new line.

cout Examples



Program 2-6

```
// Another well-adjusted printing program
#include <iostream>
using namespace std;

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
```

The \n is an escape sequence and is also used to create a new line.

Escape Sequences

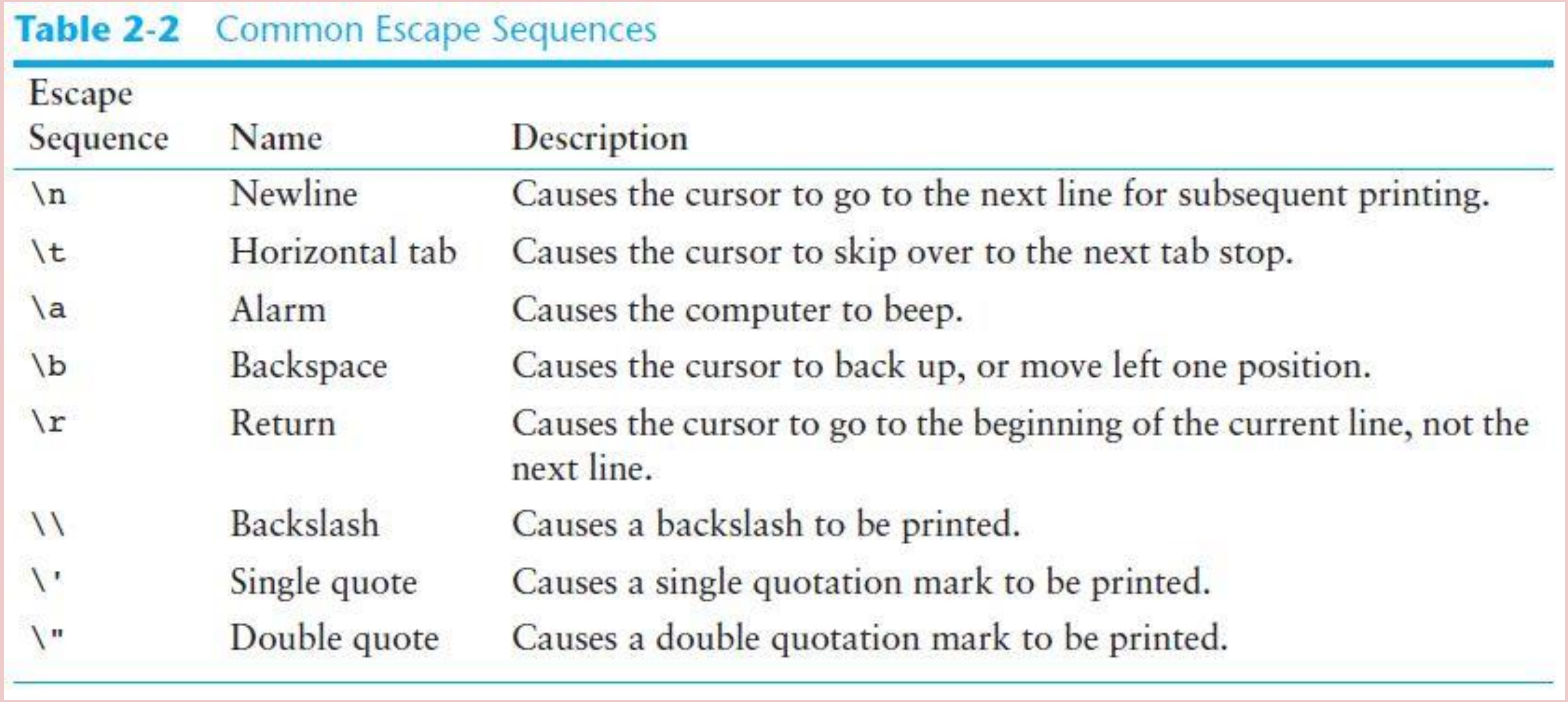


Table 2-2 Common Escape Sequences

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.

Escape sequences must be preceded by the "\" not the "/" and be included inside of double quotes.

The #include Directive

The #include directive causes the contents of another file to be inserted into the program.

The following line has appeared near the top of every example program.

```
#include <iostream>
```

The header file iostream must be included in any program that uses the cout object. This is because cout is not part of the “core” of the C++ language. Specifically, it is part of the input-output stream library. The header file, iostream, contains information describing iostream objects. Without it, the compiler will not know how to properly compile a program that uses cout.

The #include Directive

Preprocessor directives are not C++ statements. They are commands to the preprocessor, which runs prior to the compiler (hence the name “preprocessor”). The preprocessor’s job is to set programs up in a way that makes life easier for the programmer.

For example, any program that uses the cout object must contain the extensive setup information found in the iostream file. The programmer could type all this information into the program, but it would be too time consuming. An alternative would be to use an editor to “cut and paste” the information into the program, but that would still be inefficient. The solution is to let the preprocessor insert the contents of iostream automatically.

The #include Directive

An #include directive must always contain the name of a file. The preprocessor inserts the entire contents of the file into the program at the point it encounters the #include directive.

The compiler doesn’t actually see the #include directive. Instead it sees the code that was inserted by the preprocessor, just as if the programmer had typed it there.

The code contained in header files is C++ code. Typically it describes complex objects like

cout.

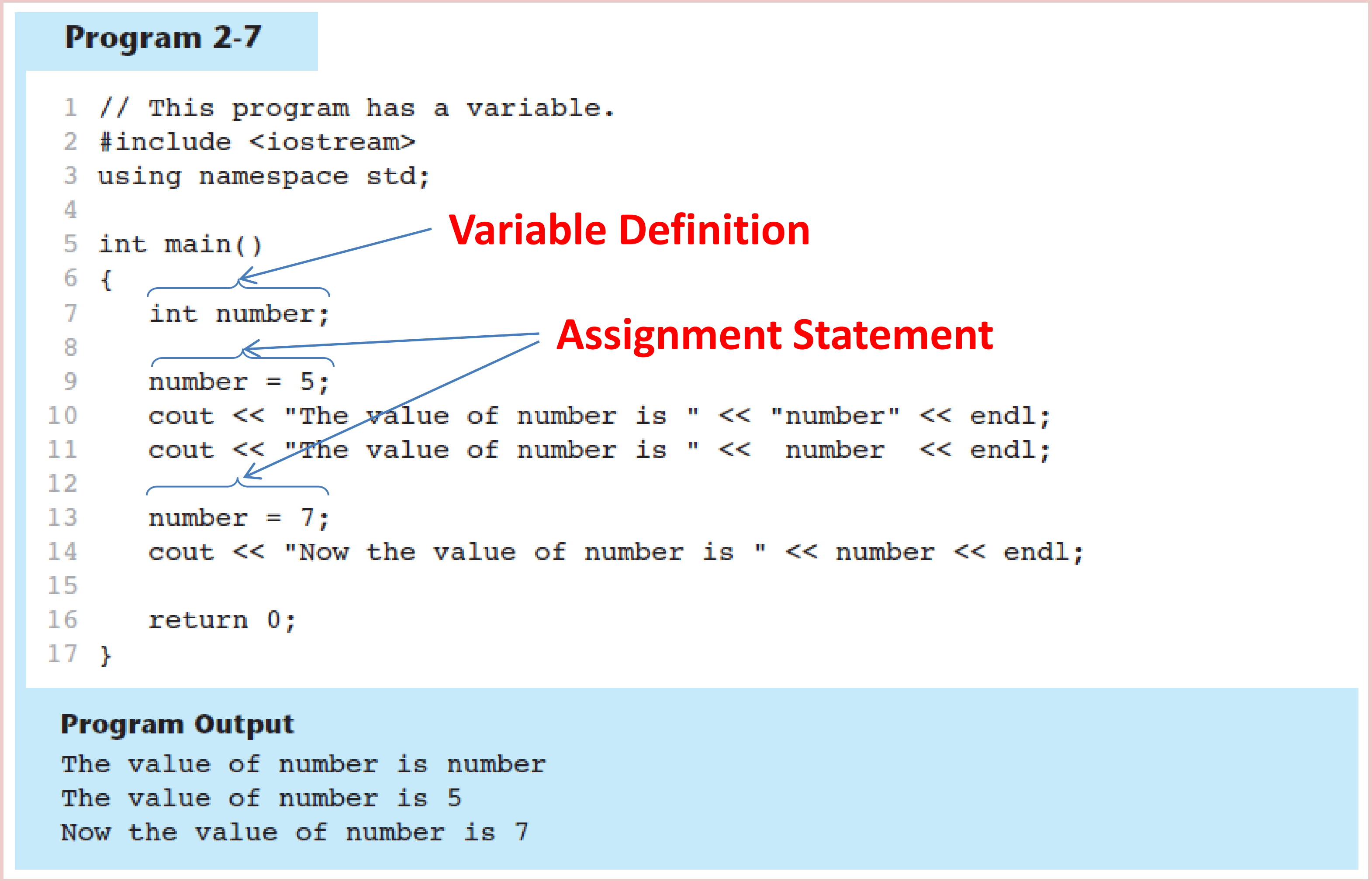
Variables, Constants, and the Assignment Statement

Variables represent storage locations in the computer’s memory.

Constants are data items whose values cannot change while the program is running.

The concept of a variable in computer programming is somewhat different from the concept of a variable in mathematics. In programming, a variable is a named storage location for holding data. Variables allow you to store and work with data in the computer’s memory. They provide an “interface” to RAM. Part of the job of programming is to determine how many variables a program will need and what type of information each will hold.

Variables, Constants, and the Assignment Statement


Variable Definition
Assignment Statement

Program 2-7

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

int main()
{
    int number;

    number = 5;
    cout << "The value of number is " << "number" << endl;
    cout << "The value of number is " <<  number  << endl;

    number = 7;
    cout << "Now the value of number is " << number << endl;

    return 0;
}
```

Program Output

```
The value of number is number
The value of number is 5
Now the value of number is 7
```

A variable definition tells the compiler the variable's name and the type of data it will hold.

Sometimes a Number Isn’t a Number

Placing quotation marks around a variable name made it a string constant, or string literal. When string literals are sent to cout, they are printed exactly as they appear inside the quotation marks.

NOTE: If we were to put the following line in a program, **it would print out the word endl**, rather than cause subsequent output to begin on a new line.

```
cout << "endl";                // Wrong!
```

Sometimes a Number Isn’t a Number

Placing double quotation marks around anything that is not intended to be a string literal will create an error of some type. For example, in Program 2-7 the number 5 was assigned to the variable number. It would have been incorrect to write the assignment this way:

```
number = "5"; // Wrong!
```

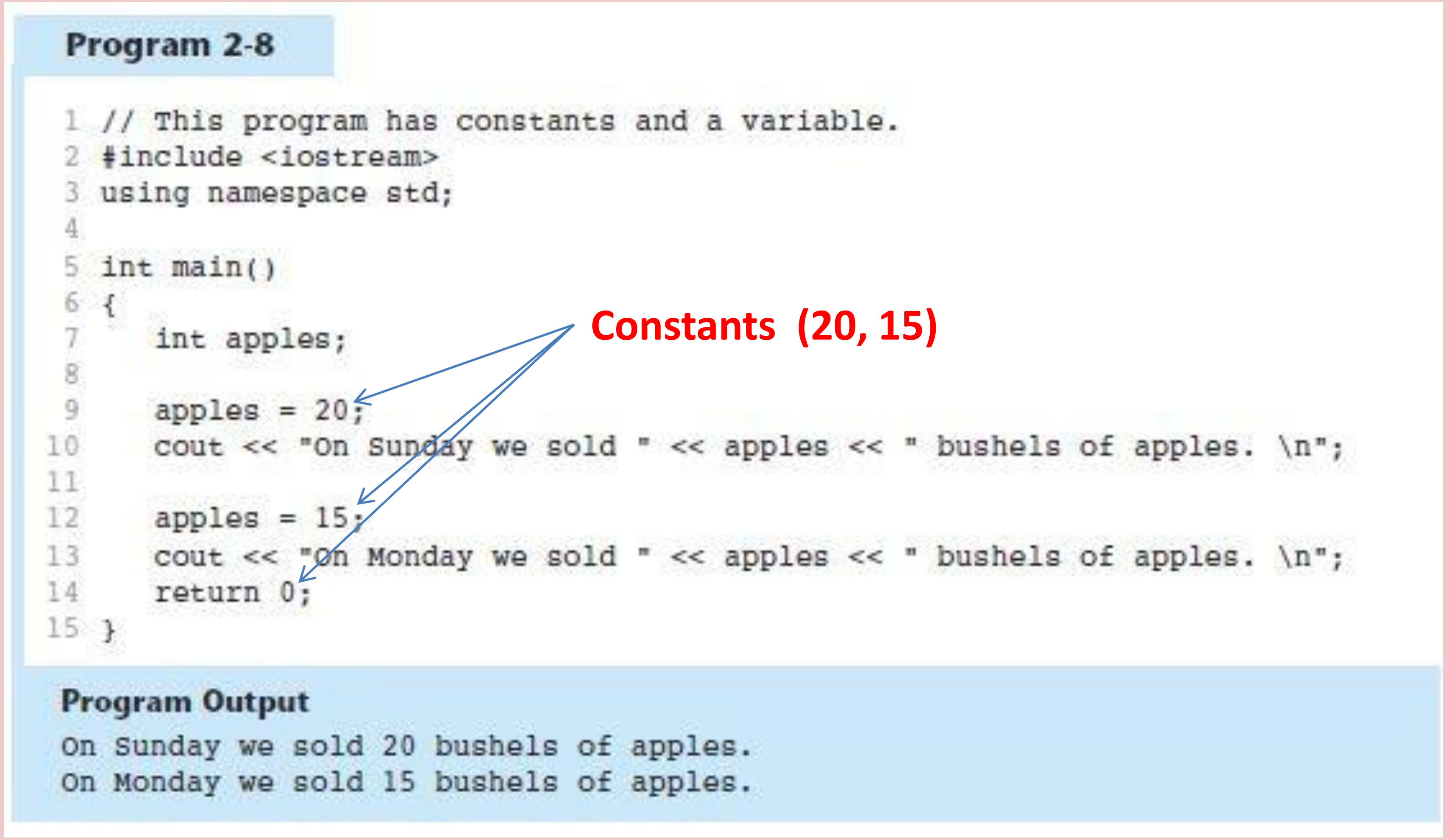
In this line, 5 is no longer an integer, but a string literal. Because number was defined to be an integer variable, you can only store integers in it. The integer 5 and the string literal “5” are not the same thing.

Sometimes a Number Isn’t a Number

The fact that numbers can be represented as strings frequently confuses people who are new to programming. Just remember that strings are intended for humans to read. They are to be printed on computer screens or paper. Numbers, however, are intended primarily for mathematical operations. You cannot perform math on strings, and you cannot display numbers on the screen without first converting them to strings. (*Fortunately, cout handles the conversion automatically when you send a number to it.*)

Constants

Unlike a variable, a constant is a data item whose value cannot change during the program's execution.



Program 2-8

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

int main()
{
   int apples;

   apples = 20;
   cout << "On Sunday we sold " << apples << " bushels of apples. \n";

   apples = 15;
   cout << "On Monday we sold " << apples << " bushels of apples. \n";
   return 0;
}
```

Program Output

```
On Sunday we sold 20 bushels of apples.
On Monday we sold 15 bushels of apples.
```

Constants

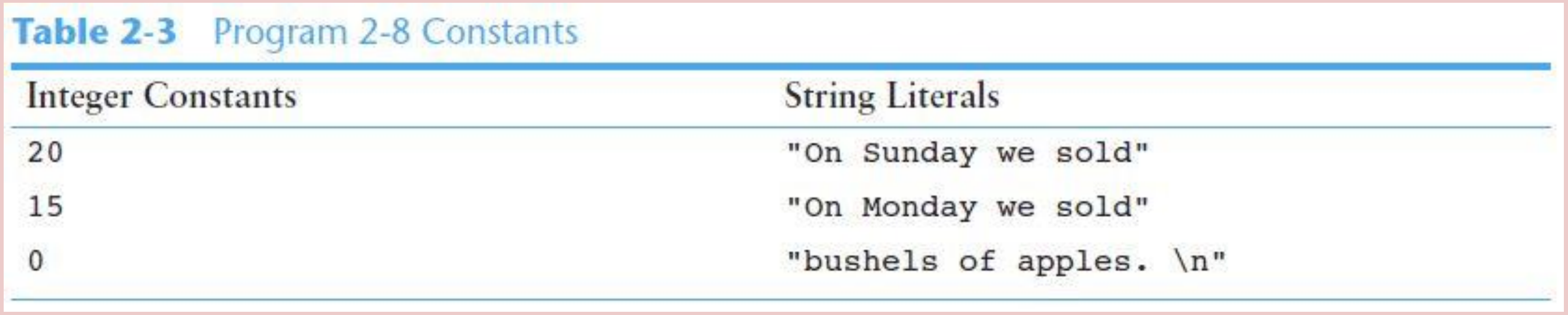


Table 2-3 Program 2-8 Constants

Integer Constants	String Literals
`20`	`"On Sunday we sold"`
`15`	`"On Monday we sold"`
`0`	`"bushels of apples. \n"`

Constants are commonly used to store known values in variables and to display messages on the screen.

Identifiers

NOTE: A variable name should indicate what the variable is used for.

An identifier is a programmer-defined name that represents some element of a program. Variable names are examples of identifiers. You may choose your own variable names in C++, as long as you do not use any of the C++ key words. The key words make up the “core” of the language and have specific purposes.

C++ Key Words

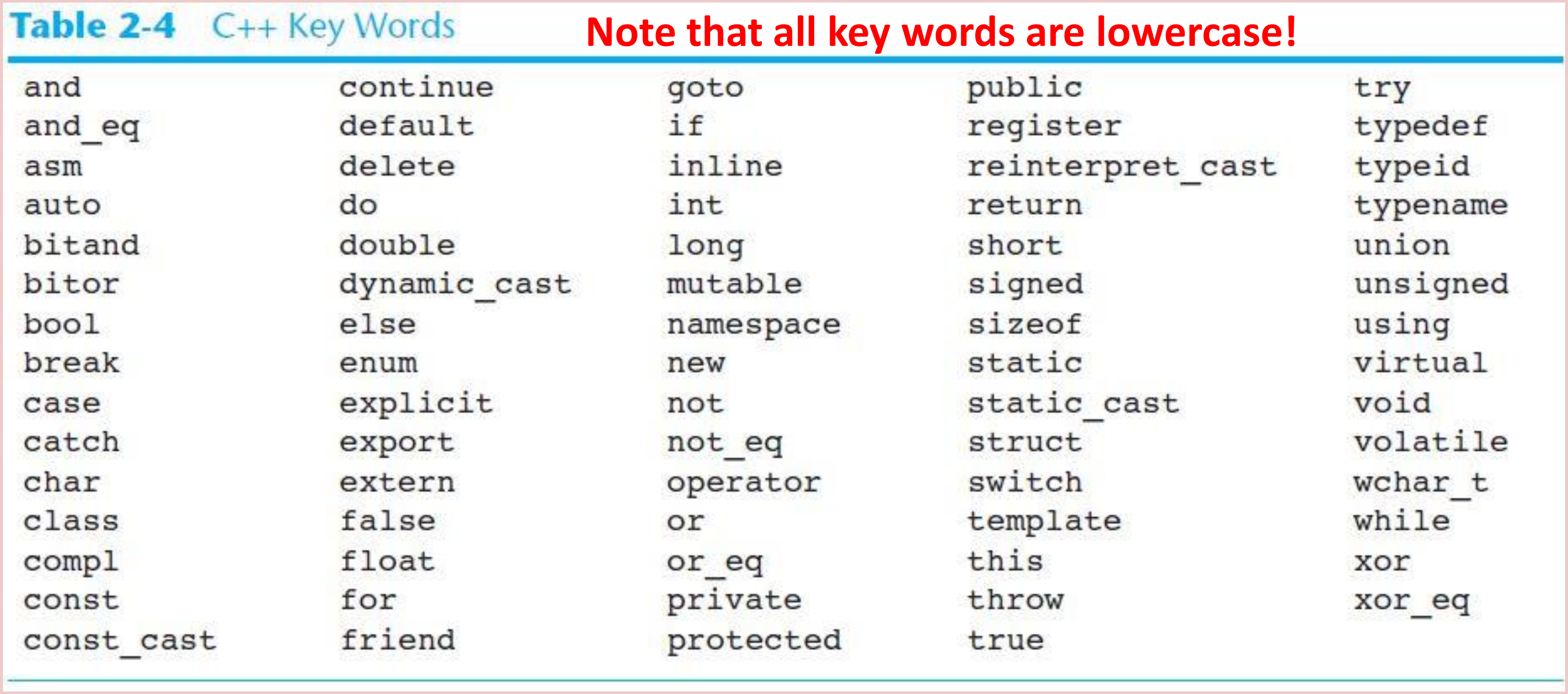


Table 2-4 C++ Key Words

Note that all key words are lowercase!

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	

Identifiers

You should always choose names for your variables that give an indication of what the variables are used for. You may be tempted to give variables names like this:

```
int x;
```

However, the rather nondescript name, x, gives no clue as to the variable's purpose. Here is a better example.

```
int itemsOrdered;
```

The name itemsOrdered gives anyone reading the program an idea of the variable's use. This way of coding helps produce self-documenting programs, which means you can get an understanding of what the program is doing just by reading its code.

Identifiers

Referring to **itemsOrdered**, it is OK to use uppercase and lowercase when naming variables. The O is capitalized for in the variable name for readability purposes.

Capitalization of the first letter of the second word and any succeeding words makes variable names like itemsOrdered easier to read and is the convention we use for naming variables in this book. ***However, this style of coding is not required.***

You are free to use all lowercase letters, all uppercase letters, or any combination of both. In fact, some programmers use the underscore character to separate words in a variable name. **int items_ordered;**

Legal Identifiers

Regardless of which style you adopt, be consistent and make your variable names as sensible as possible. Here are some specific rules that must be followed with all C++ 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**.

Legal Identifiers

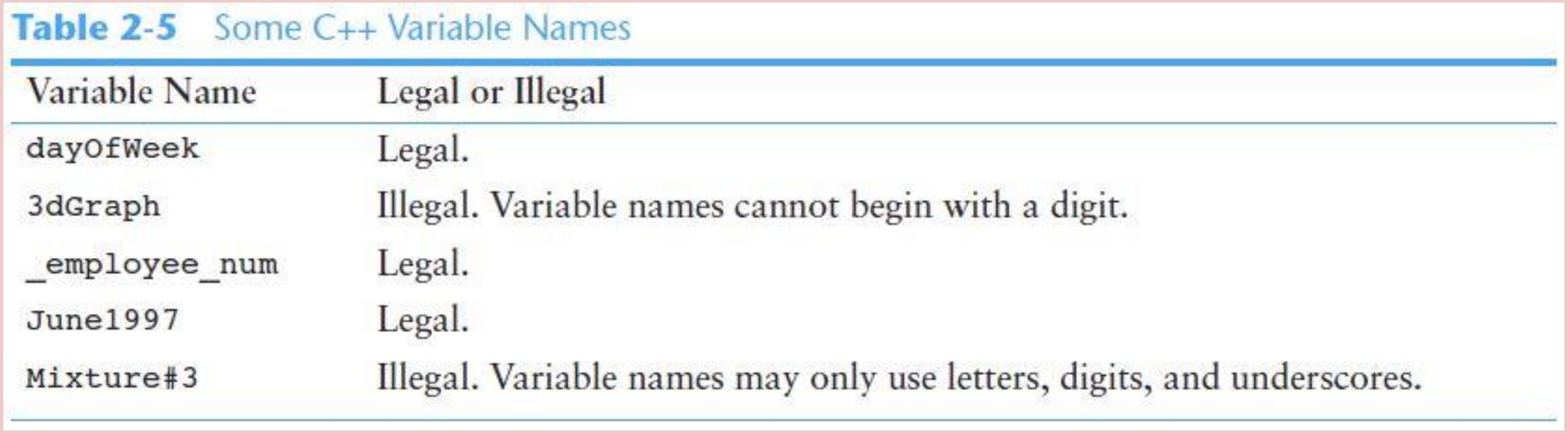


Table 2-5 Some C++ Variable Names

Variable Name	Legal or Illegal
`dayOfWeek`	Legal.
`3dGraph`	Illegal. Variable names cannot begin with a digit.
`_employee_num`	Legal.
`June1997`	Legal.
`Mixture#3`	Illegal. Variable names may only use letters, digits, and underscores.

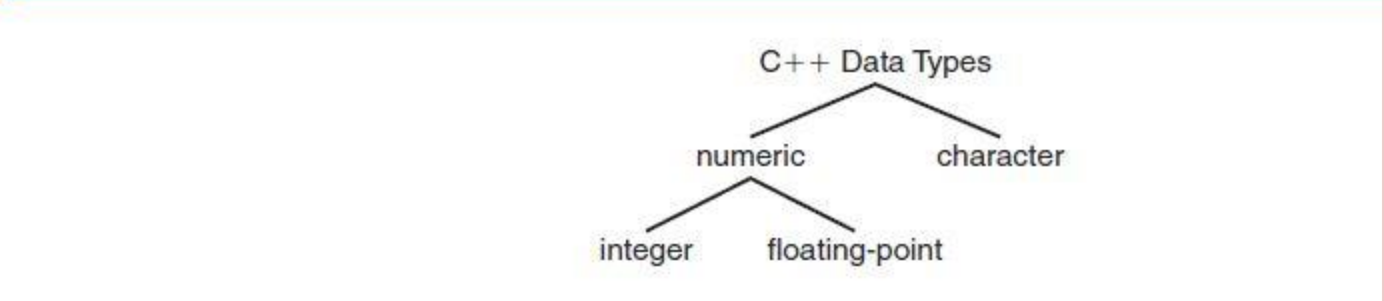
Integer Data Types

There are many different types of data. Variables are classified according to their data type, which determines the kind of information that may be stored in them. **Integer** ***variables can only hold*** **whole numbers.**

Although C++ offers many data types, in the very broadest sense there are only two: numeric and character. Numeric data types are broken into two additional categories:

integer and floating-point.

Figure 2-2 Basic C++ Data Types



Integer Data Types

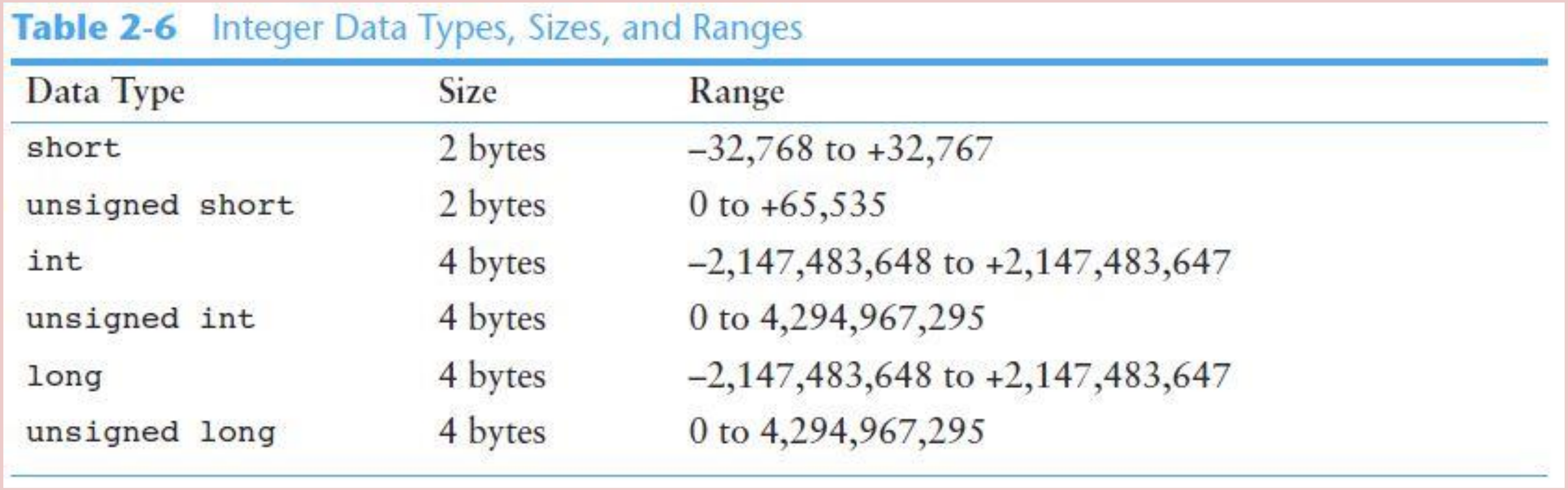
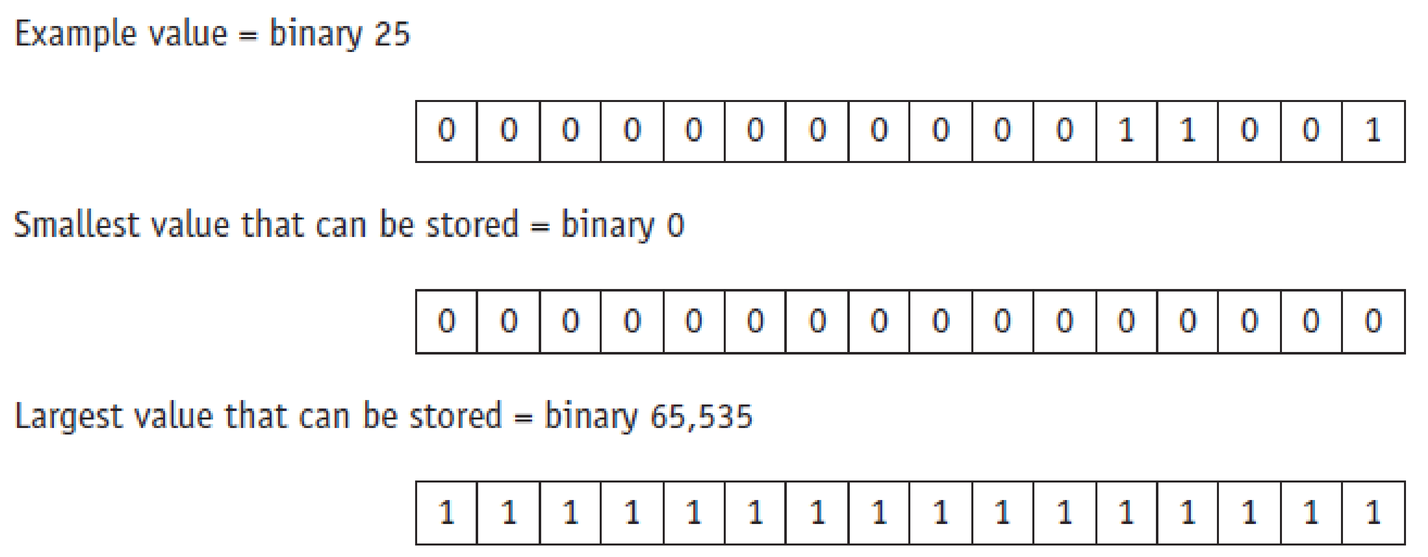


Table 2-6 Integer Data Types, Sizes, and Ranges

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

Figure 2-3 Unsigned Short Data Type Storage



Example value = binary 25

0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1

Smallest value that can be stored = binary 0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Largest value that can be stored = binary 65,535

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Integer Data Types

Here are some examples of integer variable definitions. Notice that an unsigned int variable

can also be defined using only the word unsigned, as shown below.

```
short count;
unsigned short age;
int speed;
unsigned int days;          // These two definitions
unsigned days;              // are equivalent.
long deficit;
unsigned long insects;
```

Integer Data Types

Notice also that in Table 2-6 the int and long data types have the same sizes and ranges, and the unsigned int data type has the same size and range as the unsigned long data type. This is not always true because the size of integers is dependent on the type of system you are using. Here are the only guarantees:

- Integers are at least as big as short integers.
- Long integers are at least as big as integers.
- Unsigned short integers are the same size as short integers.
- Unsigned integers are the same size as integers.
- Unsigned long integers are the same size as long integers.

Integer Data Types

In most programs you will need more than one variable of any given data type. If a program uses two integers, length and width, they can be defined separately, like this:

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

It is also possible to combine both variable definitions in a single statement:

```
int length, width;
```

Many instructors, however, prefer that each variable be placed on its own line:

```
int length,
width;
```

Integer and Long Integer Constants

Look at the following statements

```
int floors = 15,
rooms = 300,
suites = 30;
```

This statement contains three integer constants. In C++, integer constants are normally stored in memory just as an int.

One of the pleasing characteristics of the C++ language is that it allows you to control almost every aspect of your program.

Integer and Long Integer Constants

If you need to change the way something is stored in memory, the tools are provided to do that. For example, what if you are in a situation where you have an integer constant, but you need it to be stored in memory as a long integer? (Rest assured, this is a situation that does arise.) C++ allows you to force an integer constant to be stored as a long integer by placing the letter **L** at the end of the number.

Here is an example:

32L

On a computer that uses 2-byte integers and 4-byte long integers, this constant will use 4 bytes. This is called a long integer constant.

The char Data Type

Note: A variable of the char data type holds only a single character.

You might be wondering why there isn't a 1-byte integer data type. Actually there is. It is called the char data type, which gets its name from the word "character." A variable defined as a char can hold a single character, but strictly speaking, it is an integer data type.

On some systems the char data type is larger than 1 byte.

The reason an integer data type is used to store characters is because characters are internally represented by numbers. Each printable character, as well as many nonprintable characters, are assigned a unique number.

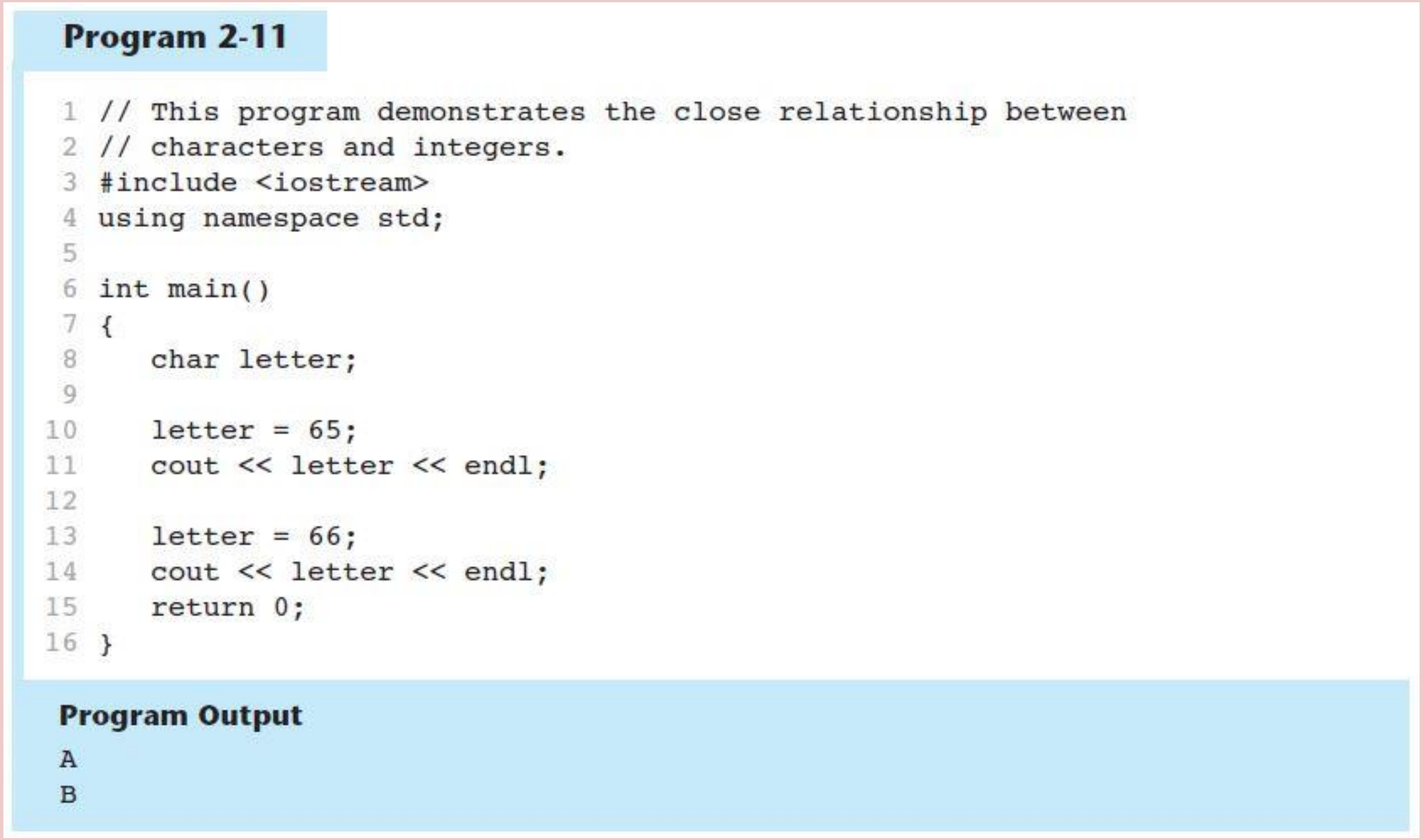
The char Data Type

The most commonly used method for encoding characters is ASCII, which stands for the American Standard Code for Information Interchange. (There are other codes, such as EBCDIC, which is used by many IBM mainframes.)

When a character is stored in memory, it is actually the numeric code that is stored. When the computer is instructed to print the value on the screen, it displays the character that corresponds with the numeric code.

Appendix A, shows the ASCII character set. The number 65 is the code for A, 66 is the code for B, and so on.

The char Data Type (Sample Program)



Program 2-11

```
// This program demonstrates the close relationship between
// characters and integers.
#include <iostream>
using namespace std;

int main()
{
   char letter;

   letter = 65;
   cout << letter << endl;

   letter = 66;
   cout << letter << endl;
   return 0;
}
```

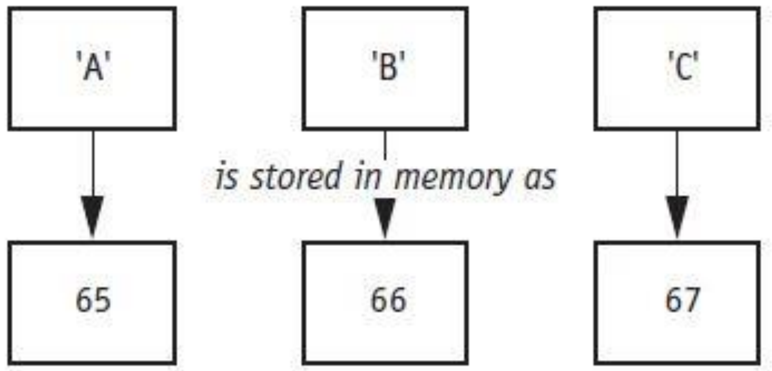
Program Output

```
A
B
```

The char Data Type (Sample Program)

Figure 2-4 illustrates that when you think of characters, such as A, B, and C, being stored in memory, it is really the numbers 65, 66, and 67 that are stored.

Figure 2-4


'A'
'B'
'C'
is stored in memory as
65
66
67