Table 1-2 Programming Language Elements

Language Element	Description
Key Words	Words that have a special meaning. Key words may only be used for their intended purpose. Key words are also known as reserved words.
Programmer-Defined Identifiers	Words or names defined by the programmer. They are symbolic names that refer to variables or programming routines.
Operators	Operators perform operations on one or more operands. An operand is usually a piece of data, like a number.
Punctuation	Punctuation characters that mark the beginning or ending of a statement, or separate items in a list.
Syntax	Rules that must be followed when constructing a program. Syntax dictates how key words and operators may be used, and where punctuation symbols must appear.

Key Words (reserved words)

Three of C++'s key words appear on lines 3 and 5: `using`, `namespace`, and `int`. The word `double`, which appears on line 7, is also a C++ key word. These words, which are always written in lowercase, each have a special meaning in C++ and can only be used for their intended purposes. As you will see, the programmer is allowed to make up his or her own

Programmer-Defined Identifiers

The words `hours`, `rate`, and `pay` that appear in the program on lines 7, 11, 15, 18, and 21 are programmer-defined identifiers. They are not part of the C++ language but rather are names made up by the programmer. In this particular program, these are the names of variables. As you will learn later in this chapter, variables are the names of memory locations that may hold data.

Operators

On line 18 the following statement appears:

```
pay = hours * rate;
```

The `=` and `*` symbols are both operators. They perform operations on pieces of data,

Punctuation

Notice that many lines end with a semicolon. A semicolon in C++ is similar to a period in English. It marks the end of a complete sentence (or statement, as it is called in programming). Semicolons do not appear at the end of every line in a C++ program, however. There are rules that govern where semicolons are required and where they are not. Part of learning C++ is learning where to place semicolons and other punctuation symbols.

A *statement* is a complete instruction that causes the computer to perform some action. Here is the statement that appears in line 10 of Program 1-1:

```
cout << "How many hours did you work? ";
```

Variables

A *variable* is a named storage location in the computer's memory for holding a piece of data. The data stored in variables may change while the program is running (hence the name "variable"). Notice that in Program 1-1 the words `hours`, `rate`, and `pay` appear in several places. All three of these are the names of variables. The `hours` variable is used to store the number of hours the user worked. The `rate` variable stores the user's hourly pay rate. The `pay` variable holds the result of `hours` multiplied by `rate`, which is the user's gross pay.

A *flowchart* is a diagram that shows the logical flow of a program. It is a useful tool for planning each operation a program must perform, and the order in which the operations are to occur. For more information see Appendix O, Introduction to Flowcharting.

Pseudocode is a cross between human language and a programming language. Although the computer can't understand pseudocode, programmers often find it helpful to write an algorithm using it. This is because pseudocode is similar to natural language, yet close enough to programming language that it can be easily converted later into program source code. By writing the algorithm in pseudocode first, the programmer can focus on just the logical steps the program must perform, without having to worry yet about syntax or about details such as how output will be displayed.

Pseudocode can be written at a high level or at a detailed level. Many programmers use both forms. High level pseudocode simply lists the steps a program must perform. Here is high level pseudocode for the pay-calculating program.

```
Get payroll data
Calculate gross pay
Display gross pay
```

Program 2-1

```
1 // A simple C++ program
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7    cout << "Programming is great fun!";
8    return 0;
9 }
```

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!
```

Let's examine the program line by line. Here's the first line:

```
// 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. That means you can type anything you want on that

Line 2 looks like this:

```
#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. Think of the preprocessor as a program that "sets up" your source code for the compiler.

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. 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. Its contents are included in the program at the point the `#include` statement appears. The `iostream` file is called a *header file*, so it should be included at the head, or top, of the program.

Line 3 reads

```
using namespace std;
```

every name created by the `iostream` file is part of that namespace. In order for a program to use the entities in `iostream`, it must have access to the `std` namespace. More

Screen clipping taken: 4/1/2013 1:06 PM

program entities that must have names. C++ uses *namespaces* to organize the names of program entities. The statement `using namespace std;` declares that the program will be accessing entities whose names are part of the namespace called `std`. (Yes, even namespaces have names.) The program needs access to the `std` namespace because every name created by the `iostream` file is part of that namespace. In order for a program to use the entities in `iostream`, it must have access to the `std` namespace. More information on namespaces can be found in Appendix F.

Line 5 reads

```
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.

Line 6 contains a single, solitary character:

```
{
```

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`.

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

After the opening brace you see the following statement in line 7:

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

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 "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*.

Notice that line 7 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

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.

The last line of the program, line 9, contains the closing brace:

```
}
```

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.

2.2 The cout Object

CONCEPT: `cout` is used to display information on the computer's screen.

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

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

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

```
#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
```

long line is that cout does not start a new line unless told to do so. There are two ways to instruct cout to start a new line. The first is to send cout a *stream manipulator* called endl (pronounced "end-line" or "end-L"). Program 2-5 does this.

Program 2-5

```
1  // A well-adjusted printing program
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     cout << "The following items were top sellers" << endl;
8     cout << "during the month of June:" << endl;
9     cout << "Computer games" << endl;
10    cout << "Coffee" << endl;
11    cout << "Aspirin" << endl;
12    return 0;
13 }
```

(program continues)

Program Output

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

Program 2-6

```
1  // Another well-adjusted printing program
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     cout << "The following items were top sellers\n";
8     cout << "during the month of June:\n";
9     cout << "Computer games\nCoffee";
10    cout << "\nAspirin\n";
11    return 0;
12 }
```

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.

2.5 Variables, Constants, and the Assignment Statement

CONCEPT: Variables represent storage locations in the computer's memory. Constants are data items whose values cannot change while the program is running.

Program 2-7

```
1  // This program has a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int number;
8
9     number = 5;
10    cout << "The value of number is " << "number" << endl;
11    cout << "The value of number is " <<  number  << endl;
12
13    number = 7;
14    cout << "Now the value of number is " << number << endl;
15
16    return 0;
17 }
```

Program Output

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

```
number = 5;
```

This is called an *assignment statement* and the = sign is called the *assignment operator.*

Checkpoint

2.6 Which of the following are legal C++ assignment statements?

```
a.  a = 7;
b.  7 = a;
c.  7 = 7;
```

2.7 List all the variables ~~and constants~~ that appear below.

```
int main()
{
   int little;
   int big;
   little = 2;
   big = 2000;
   cout << "The little number is " << little << endl;
   cout << "The big number is " << big << endl;
   return 0;
}
```

2.8 When the above program is run, what will display on the screen?

Solucion:
The little number is 2
The big number is 2000

2.9 What will the following program display on the screen?

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

int main()
{
   int number;

   number = 712;
   cout << "The value is " << "number" << endl;
   return 0;
}
```

The value is number

2.6 Identifiers

CONCEPT: 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. Table 2-4 shows a complete list of the C++ key words. Note that they are all lowercase.

Table 2-4 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

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`.

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.

2.7 Integer Data Types

CONCEPT: 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.

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

2.8 The `char` Data Type

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

```
int main()
{
   char letter;
```


rogram 2-12 *(continued)*

```
   letter = 'A';
   cout << letter << endl;
```

2.9 The C++ string Class

CONCEPT: Standard C++ provides a special data type for storing and working with strings.

Using the string Class

The first step in using the `string` class is to `#include` the `string` header file. This is accomplished with the following preprocessor directive:

```
#include <string>
```

The next step is to define a string type variable, called a string object. Defining a string object is similar to defining a variable of a primitive type. For example, the following statement defines a string object named `movieTitle`.

```
string movieTitle;
```

```
movieTitle = "Wheels of Fury";
```

2.10 Floating-Point Data Types

CONCEPT: Floating-point data types are used to define variables that can hold real numbers.

Table 2-8 Floating-Point Data Types on PCs

Data Type	Key Word	Size	Range	Significant Digits
Single precision	`float`	4 bytes	Numbers between ±3.4E-38 and ±3.4E38	7
Double precision	`double`	8 bytes	Numbers between ±1.7E-308 and ±1.7E308	16
Long double precision	`long double`	8 bytes*	Numbers between ±1.7E-308 and ±1.7E308	16

Program 2-15

```
1 // This program uses two floating-point data types, float and doub
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7    float distance = 1.496E8;          // in kilometers
8    double mass = 1.989E30;            // in kilograms
9
10   cout << "The Sun is " << distance << " kilometers away.\n";
11   cout << "The Sun\'s mass is " << mass << " kilograms.\n";
12   return 0;
13 }
```

Program Output

```
The Sun is 1.496e+008 kilometers away.
The Sun's mass is 1.989e+030 kilograms.
```

```
1   // This program uses two floating-point data types, float and double.
2   #include <iostream>
3   using namespace std;
4
5   int main(){
6   float distance = 1.496E8;  // in kilometers
7   double mass = 1.989E30;  // in kilograms
8
9   cout << "The Sun is " << distance << " kilometers away.\n";
10  cout << "The Sun\'s mass is " << mass << " kilograms.\n";
11  return 0;
12  }
```

```
C:\MinGW\bin>g++ examplefloat.cpp -o examplefloat.exe

C:\MinGW\bin>examplefloat.exe
The Sun is 1.496e+008 kilometers away.
The Sun's mass is 1.989e+030 kilograms.

C:\MinGW\bin>_
```

2.11 The bool Data Type

CONCEPT: Boolean variables are set to either `true` or `false`.

Program 2-16

```
1  // This program uses Boolean variables.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     bool boolValue;
8
9     boolValue = true;
10    cout << boolValue << endl;
11
12    boolValue = false;
13    cout << boolValue << endl;
14    return 0;
15 }
```

Program Output

```
1
0
```

Constants: se **definen** tal y como aparece abajo (el typedef puede ser int, bool, float, double, string, etc. Y se debe **inicializar** en el mismo statement. Luego no se puede modificar - compiler error-)

```
const double INTEREST_RATE = 0.129;
```