Chapter 2 Introduction to C++

2.1 The Parts of a C++ Program

Concept:

C++ programs have parts and components that serve specific purposes.

- Every C++ program has a structure, though the parts are not always in the same location. Learning these parts is the first step to learning C++.

Program 2-1

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

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

Program Output

- **Line 1:** `// A simple C++ program`
  - The `//` signifies a comment.
  - The compiler ignores all text from the `//` to the end of the line.
  - Comments are crucial for explaining complex code to human readers.
- **Line 2:** `#include <iostream>`
  - Lines starting with `#` are preprocessor directives.
  - The preprocessor runs before the compiler, setting up the source code.

- The `#include` directive includes a header file, in this case, `iostream`.
- The `<iostream>` file is required for screen output (`cout`) and keyboard input.

- **Line 3:** `using namespace std;`
  - C++ uses namespaces to organize the names of program entities like variables and functions.
  - This statement declares that the program will use entities from the `std` (standard) namespace.
  - Access to the `std` namespace is necessary because the names used from `iostream` are part of it.

- **Line 5:** `int main()`
  - This marks the beginning of a function named `main`.
  - A function is a named group of programming statements.
  - The `()` after `main` indicates it is a function.
  - `int` signifies that the function returns an integer value to the operating system upon completion.
  - Every C++ program must have a `main` function, as it is the program's starting point.

> **Note:**
> 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 "INT" the same as "int." This is true for all the C++ key words.

- **Line 6:** `{`
  - This is a left or opening brace.
  - It marks the beginning of the `main` function's body. All statements within the function are enclosed in braces.

> **Warning!**
> Make sure you have a closing brace for every opening brace in your program!

- **Line 7:** `cout << "Programming is great fun!";`
  - This statement displays a message on the screen.
  - The text inside the double quotation marks is called a string literal.
  - A semicolon `;` marks the end of a complete C++ statement.

> **Note:**
> This is the only line in the program that causes anything to be printed on the screen. The other lines, like `#include <iostream>` and `int main()`, are necessary for the framework of your program, but they do not cause any screen output. Remember, a program is a set of instructions for the computer. If something is to be displayed on the screen, you must use a programming statement for that purpose.

- **Line 8:** `return 0;`
  - This sends the integer value 0 back to the operating system.
  - A return value of 0 typically indicates that the program executed successfully.
- **Line 9:** `}`
  - This is a closing brace.
  - It marks the end of the `main` function.

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	Enclose 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	Enclose a group of statements, such as the contents of a function.
`" "`	Opening and closing quotation marks	Enclose a string of characters, such as a

CHARACTER	NAME	DESCRIPTION
		message that is to be printed on the screen.
`;`	Semicolon	Marks the end of a complete programming statement.

Checkpoint

2.1 The following C++ program will not compile because the lines have been mixed up.

```
int main()
}
return 0;
#include <iostream>
cout << "In 1492 Columbus sailed the ocean blue.";
{
using namespace std;
```

When the lines are properly arranged, the program should display the following on the screen:

```
In 1492 Columbus sailed the ocean blue.
```

Rearrange the lines in the correct order. Test the program by entering it on the computer, compiling it, and running it.

2.2 The `cout` Object

Concept:

Use the `cout` object to display information on the computer's screen.

- Console output refers to the plain text a program displays, typically in a dedicated window.
- In C++, the `cout` object is used to produce console output. It is a stream object, working with streams of data.
- The `<<` operator is the stream insertion operator. It sends data to `cout` to be displayed on the screen.

- The operator is written as two less-than signs (`<<`) and must point toward `cout`.

Using `cout`

Program 2-2

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

Program Output

Program 2-3

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

Program Output

- The `cout` object displays information in a continuous stream, without automatically adding spaces or new lines.

Program 2-4

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

```
3
4   int main()
5   {
6       cout << "The following items were top sellers";
7       cout << "during the month of June:";
8       cout << "Computer games";
9       cout << "Coffee";
10      cout << "Aspirin";
11      return 0;
12  }
```

Program Output

- To start a new line, you can use one of two methods:
  - `endl` **Stream Manipulator**: When `cout` encounters `endl`, it moves the cursor to the beginning of the next line.

Program 2-5

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

Program Output

Note:

The last character in `endl` is the lowercase letter L, *not* the number one.

- **Escape Sequence**: An escape sequence starts with a backslash (`\`) and is embedded inside a string to control output.
  - The newline escape sequence is `\n`. It tells `cout` to advance the cursor to the next line.

Program 2-6

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

Program Output

- Common mistakes include using a forward slash (`/n` instead of `\n`) or placing `\n` outside the quotation marks.

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.

ESCAPE SEQUENCE	NAME	DESCRIPTION
`\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.

Warning!

When using escape sequences, do not put a space between the backslash and the control character.

- An escape sequence is typed as two characters but is stored in memory as a single character.

2.3 The `#include` Directive

Concept:

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

- The `iostream` header file must be included in any program that uses the `cout` object because `cout` is part of the input-output stream library, not the core C++ language.
- Preprocessor directives are commands to the preprocessor, which runs before the compiler.

- The `#include` directive automatically inserts the necessary setup information from a header file, saving the programmer from typing it manually.
- The compiler sees the inserted code, not the `#include` directive itself.

Warning!

Do not put semicolons at the end of processor directives. Because preprocessor directives are not C++ statements, they do not require semicolons. In many cases, an error will result from a preprocessor directive terminated with a semicolon.

Checkpoint

2.2 The following C++ program will not compile because the lines have been mixed up.

```
cout << "Success\n";
cout << " Success\n\n";
int main()
cout << "Success";
}
using namespace std;
#include <iostream>
cout << "Success\n";
{
return 0;
```

When the lines are properly arranged, the program should display the following on the screen:

```
Success
Success Success

Success
```

Rearrange the lines in the correct order. Test the program by entering it on the computer, compiling it, and running it.

2.3 Study the following program and show what it will print on the screen:

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

```
   cout << "The works of Wolfgang\ninclude the following";
   cout << "\nThe Turkish March" << endl;
   cout << "and Symphony No. 40 ";
   cout << "in G minor." << endl;
   return 0;
}
```

2.4 Write a program that will display your name on the first line, your street address on the second line, your city, state, and ZIP code on the third line, and your telephone number on the fourth line. Place a comment with today's date at the top of the program. Test your program by compiling and running it.

2.4 Variables, Literals, and Assignment Statements

Concept:

Variables represent storage locations in the computer's memory. Literals are constant values that are assigned to variables.

- Variables allow programs to store and work with data in the computer's memory (RAM).
- Programmers must determine the number and types of variables a program needs.

Program 2-7

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int number;
7
8       number = 5;
9       cout << "The value in number is " << number << endl;
10      return 0;
11  }
```

Program Output

- **Variable Definition**: `int number;`
  - This statement defines a variable.

- It tells the compiler the variable's name (`number`) and the type of data it holds (`int` for integer).
- Variable definitions end with a semicolon.

Variable Definitions

> **Note:**
>
> You must have a definition for every variable you intend to use in a program. In C++, variable definitions can appear at any point in the program. Later in this chapter, and throughout the book, you will learn the best places to define variables.

- **Assignment Statement**: `number = 5;`
  - The equal sign (`=`) is the assignment operator.
  - It copies the value on its right into the variable on its left.

> **Note:**
>
> This line does not print anything on the computer's screen. It runs silently behind the scenes, storing a value in RAM.

- When a variable name is sent to `cout` without quotes, its value is printed.
- If the variable name is enclosed in quotes (`"number"`), it is treated as a string literal and the name itself is printed.

Program 2-8

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int number;
7
8       number = 5;
9       cout << "The value in number is " << "number" << endl;
10      return 0;
11  }
```

Program Output

- Numbers can be represented as numeric types (for math) or as strings (for display). You cannot perform math on a string representation of a number (e.g., `"5"`).

Literals

- A literal is a piece of data written directly into the program's code, like `100` or `"Welcome to my program."`.
- Literals are often used to assign values to variables or for display.

Program 2-9

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int apples;
7
8       apples = 20;
9       cout << "Today we sold " << apples << " bushels of apples.\n";
10      return 0;
11  }
```

Program Output

Table 2-3 Literals and Their Types

LITERAL	TYPE OF LITERAL
`20`	Integer literal
`"Today we sold"`	String literal
`"bushels of apples.\n"`	String literal
`0`	Integer literal

Note:

Literals are also called constants.

Checkpoint

2.5 Examine the following program:

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

List all the variables and literals that appear in the program.

2.6 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;
}
```

2.5 Identifiers

Concept:

Choose variable names that indicate what the variables are used for.

- An identifier is a programmer-defined name for a program element, such as a variable.

- You cannot use C++ key words as identifiers. Key words are reserved and have specific meanings.

Table 2-4 The C++ Key Words

`alignas`	`const`	`for`	`private`	`throw`
`alignof`	`constexpr`	`friend`	`protected`	`true`
`and`	`const_cast`	`goto`	`public`	`try`
`and_eq`	`continue`	`if`	`register`	`typedef`
`asm`	`decltype`	`inline`	`reinterpret_cast`	`typeid`
`auto`	`default`	`int`	`return`	`typename`
`bitand`	`delete`	`long`	`short`	`union`
`bitor`	`do`	`mutable`	`signed`	`unsigned`
`bool`	`double`	`namespace`	`sizeof`	`using`
`break`	`dynamic_cast`	`new`	`static`	`virtual`
`case`	`else`	`noexcept`	`static_assert`	`void`
`catch`	`enum`	`not`	`static_cast`	`volatile`
`char`	`explicit`	`not_eq`	`struct`	`wchar_t`
`char16_t`	`export`	`nullptr`	`switch`	`while`
`char32_t`	`extern`	`operator`	`template`	`xor`
`class`	`false`	`or`	`this`	`xor_eq`
`compl`	`float`	`or_eq`	`thread_local`	

- Choose meaningful variable names to make code self-documenting and easier to understand (e.g., `itemsOrdered` instead of `x`).
- Common naming conventions include CamelCase (`itemsOrdered`) or using underscores (`items_ordered`).

Legal Identifiers

- The first character must be a letter (a-z, A-Z) or an underscore (`_`).
- After the first character, you can use letters, digits (0-9), or underscores.
- Uppercase and lowercase characters are distinct (e.g., `ItemsOrdered` is different from `itemsordered`).

Table 2-5 Some 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, or underscores.

2.6 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.

- A variable's data type determines the kind of information it can hold.
- Data types can be broadly categorized as numeric (integer, floating-point) and character.
- When choosing a numeric data type, consider:
  - Largest and smallest possible values.
  - Memory usage.
  - Whether it needs to store negative numbers (signed vs. unsigned).
  - Required decimal precision.

Table 2-6 Integer Data Types

DATA TYPE	TYPICAL SIZE	TYPICAL RANGE
`short int`	2 bytes	−32,768 to +32,767
`unsigned short int`	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 int`	4 bytes	−2,147,483,648 to +2,147,483,647
`unsigned long int`	4 bytes	0 to 4,294,967,295
`long long int`	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`unsigned long long int`	8 bytes	0 to 18,446,744,073,709,551,615

Note:

The data type sizes and ranges shown in Table 2-6 are typical on many systems. Depending on your operating system, the sizes and ranges may be different.

- Most integer data types can be abbreviated (e.g., `short int` as `short`, `unsigned int` as `unsigned`).
- Unsigned data types can only store non-negative values.
- C++ has size guarantees: `long long` is at least 64 bits, `long` is at least as big as `int`, and `int` is at least as big as `short`.

Note:

The `long long int` and the `unsigned long long int` data types were introduced in C++ 11.

Program 2-10

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

int main()
{
    int checking;
    unsigned int miles;
    long days;

    checking = -20;
    miles = 4276;
    diameter = 100000;
    cout << "We have made a long journey of " << miles;
    cout << " miles.\n";
    cout << "Our checking account balance is " << checking;
    cout << "\nThe galaxy is about " << diameter;
    cout << " light years in diameter.\n";
    return 0;
}
```

Program Output

- You can define multiple variables of the same type in a single statement by separating them with commas.

Program 2-11

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

int main()
{
    int floors, rooms, suites;

    floors = 15;
    rooms = 300;
    suites = 30;
    cout << "The Grande Hotel has " << floors << " floors\n";
    cout << "with " << rooms << " rooms and " << suites;
```

```
13        cout << " suites.\n";
14        return 0;
15    }
```

Program Output

Integer and Long Integer Literals

- Numeric literals without a decimal point are typically treated as type `int`.
- To force a literal to be a `long`, append the letter `L` (e.g., `32L`).
- To force a literal to be a `long long`, append `LL` (e.g., `32LL`).

Tip:

When writing long integer literals or long long integer literals, you can use either an uppercase or a lowercase L. Because the lowercase l looks like the number 1, you should always use the uppercase L.

If You Plan to Continue in Computer Science: Hexadecimal and Octal Literals

- Programmers sometimes use hexadecimal (base 16) or octal (base 8) numbering systems.
- In C++, hexadecimal literals are prefixed with `0x` (e.g., `0xF4`).
- Octal literals are prefixed with `0` (e.g., `031`).

Note:

You will not be writing programs for some time that require this type of manipulation. It is important, however, that you understand this material. Good programmers should develop the skills for reading other people's source code. You may find yourself reading programs that use items like long integer, hexadecimal, or octal literals.

Checkpoint

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

```
X
99bottles
july97
theSalesFigureForFiscalYear98
r&d
grade_report
```

2.8 Is the variable name `Sales` the same as `sales`? Why or why not?

2.9 Refer to the data types listed in Table 2-6 for these questions.

If a variable needs to hold numbers in the range 32 to 6,000, what data type would be best?

If a variable needs to hold numbers in the range 240,000 to 140,000, what data type would be best?

Which of the following literals uses more memory? `20` or `20L`

2.10 On any computer, which data type uses more memory, an integer or an unsigned integer?

2.7 The `char` Data Type

- The `char` data type is used to store a single character.
- Character literals are enclosed in single quotation marks (e.g., `'g'`).
- You cannot assign a string (even one with a single character like `"g"`) to a `char` variable.

Program 2-12

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       char letter;
7
8       letter = 'A';
9       cout << letter << endl;
10      letter = 'B';
11      cout << letter << endl;
```

```
12        return 0;
13    }
```

Program Output

- The `char` type is an integer data type, typically 1 byte, because characters are stored internally as numeric codes.
- The most common character encoding is ASCII (American Standard Code for Information Interchange).
- For example, the character 'A' is stored as the ASCII code 65.

Program 2-13

```
1    #include <iostream>
2    using namespace std;
3
4    int main()
5    {
6        char letter;
7
8        letter = 65;
9        cout << letter << endl;
10       letter = 66;
11       cout << letter << endl;
12       return 0;
13   }
```

Program Output

The Difference between String Literals and Character Literals

- Strings are sequences of characters stored in consecutive memory locations.
- String literals stored in memory are always appended with a null terminator (`\0`), which marks the end of the string.
- The null terminator is ASCII code 0, not to be confused with the character `'0'`, which is ASCII code 48.

- Due to the null terminator, a string like `"Sebastian"` (9 characters) occupies 10 bytes of memory.

> **Note:**
> C++ automatically places the null terminator at the end of string literals.

- A character literal like `'A'` is stored as a single byte (its ASCII code).
- A string literal like `"A"` is stored as two bytes: the ASCII code for 'A' and the null terminator.
- Escape sequences like `\n` are represented by two characters in code but are stored internally as a single character.

Program 2-14

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       char letter;
7
8       letter = 'A';
9       cout << letter << '\n';
10      letter = 'B';
11      cout << letter << '\n';
12      return 0;
13  }
```

Program Output

- **Review of Key Points:**
  - Characters are represented internally by numeric codes (usually ASCII).
  - `char` variables typically occupy one byte.
  - Strings are consecutive sequences of characters.
  - String literals are terminated by a null character (`\0`) in memory.

- Character literals use single quotes (`'...'`), while string literals use double quotes (`"..."`).

2.8 The C++ `string` Class

Concept:

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

- Because `char` variables hold only one character, C++ provides the `string` class to work with variables that can hold entire strings.

Using the `string` Class

- **Step 1**: Include the `<string>` header file: `#include <string>`.
- **Step 2**: Define a `string` object (a variable of type `string`): `string movieTitle;`.
- **Step 3**: Assign a value using the assignment operator: `movieTitle = "Wheels of Fury";`.
- **Step 4**: Use the `string` object with `cout` just like any other variable.

Program 2-15

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main()
6   {
7       string movieTitle;
8
9       movieTitle = "Wheels of Fury";
10      cout << "My favorite movie is " << movieTitle << endl;
11      return 0;
12  }
```

Program Output

Checkpoint

2.11 What are the ASCII codes for the following characters? (Refer to Appendix A.)

```
C
F
W
```

2.12 Which of the following is a character literal?

```
'B'
"B"
```

2.13 Assuming the `char` data type uses 1 byte of memory, how many bytes do the following literals use?

```
'Q'
"Q"
"Sales"
'\n'
```

2.14 Write a program that has the following character variables: `first`, `middle`, and `last`. Store your initials in these variables then display them on the screen.

2.15 What is wrong with the following program statement?

```
char letter = "Z";
```

2.16 What header file must you include in order to use `string` objects?

2.17 Write a program that stores your name, address, and phone number in three separate `string` objects. Display the contents of the `string` objects on the screen.

2.9 Floating-Point Data Types

Concept:

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

- Floating-point numbers are values that can have fractional parts.
- They are stored internally in a format similar to scientific notation (e.g., 47,281.97 is 4.728197×10^4).
- Computers often use E notation, where 4.728197×10^4 is written as `4.728197E4`.

Table 2-7 Floating-Point Representations

DECIMAL NOTATION	SCIENTIFIC NOTATION	E NOTATION
247.91	2.4791×10^{2}	`2.4791E2`
0.00072	7.2×10^{-4}	`7.2E-4`
2,900,000	2.9×10^{6}	`2.9E6`

- C++ offers three floating-point data types:
  - `float`: Single precision.
  - `double`: Double precision (usually twice the size of `float`).
  - `long double`: Intended to be larger than or equal to `double`.
- Size guarantees: A `double` is at least as big as a `float`, and a `long double` is at least as big as a `double`.

Table 2-8 Floating-Point Data Types on PCs

DATA TYPE	KEY WORD	DESCRIPTION
Single precision	`float`	4 bytes. Numbers between ±3.4E–38 and ±3.4E38
Double precision	`double`	8 bytes. Numbers between ±1.7E–308 and ±1.7E308
Long double precision	`long double`	8 bytes*. Numbers between ±1.7E–308 and ±1.7E308
*Some compilers use 10 bytes for long `doubles`. This allows a range of ±3.4E–4932 to ±1.1E4832.		

Floating-Point Literals

- Floating-point literals can be written in E notation or standard decimal notation.
- By default, floating-point literals are treated as `double`s.
- You can force a literal to be a `float` by appending `F` or `f` (e.g., `1.2F`).
- You can force a literal to be a `long double` by appending `L` or `l` (e.g., `1034.56L`).

Program 2-16

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       float distance;
7       double mass;
8
9       distance = 1.495979E11;
10      mass = 1.989E30;
11      cout << "The Sun is " << distance << " meters away.\n";
12      cout << "The Sun\'s mass is " << mass << " kilograms.\n";
13      return 0;
14  }
```

Program Output

Note:

Because floating-point literals are normally stored in memory as `double`s, most compilers issue a warning message when you assign a floating-point literal to a `float` variable. For example, assuming `num` is a `float`, the following statement might cause the compiler to generate a warning message:

```
num = 14.725;
```

You can suppress the warning message by appending the f suffix to the floating-point literal, as shown below:

```
num = 14.725f;
```

Assigning Floating-Point Values to Integer Variables

- When a floating-point value is assigned to an integer variable, the fractional part is discarded (truncated).

- This is truncation, not rounding. For example, assigning 7.9 to an `int` variable will result in 7 being stored.

Note:

When a floating-point value is truncated, it is not rounded. Assigning the value 7.9 to an `int` variable will result in the value 7 being stored in the variable.

Warning!

Floating-point variables can hold a much larger range of values than integer variables can. If a floating-point value is being stored in an integer variable, and the whole part of the value (the part before the decimal point) is too large for the integer variable, an invalid value will be stored in the integer variable.

2.10 The `bool` Data Type

Concept:

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

- Expressions that result in a `true` or `false` value are called Boolean expressions.
- The `bool` data type creates variables that can hold `true` or `false` values.
- Internally, `true` is represented by the integer 1, and `false` is represented by 0.

Program 2-17

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

Program Output

2.11 Determining the Size of a Data Type

Concept:

The `sizeof` operator may be used to determine the size of a data type on any system.

- The size of data types can vary between different computer systems.
- The `sizeof` operator reports the number of bytes of memory used by any data type or variable.
- You place the data type or variable name inside parentheses following the operator (e.g., `sizeof(int)`).

Program 2-18

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       long double apple;
7
8       cout << "The size of an integer is " << sizeof(int);
9       cout << " bytes.\n";
10      cout << "The size of a long integer is " << sizeof(long);
11      cout << " bytes.\n";
12      cout << "An apple can be eaten in " << sizeof(apple);
13      cout << " bytes!\n";
14      return 0;
15  }
```

Program Output

Checkpoint

2.18 Yes or No: Is there an unsigned floating-point data type? If so, what is it?

2.19 How would the following number in scientific notation be represented in E notation?

$$6.31 \times 10^{17}$$

2.20 Write a program that defines an integer variable named `age` and a `float` variable named `weight`. Store your age and weight, as literals, in the variables. The program should display these values on the screen in a manner similar to the following:

```
My age is 26 and my weight is 180 pounds.
```

(Feel free to lie to the computer about your age and your weight—it'll never know!)

2.12 More about Variable Assignments and Initialization

Concept:

An assignment operation assigns, or copies, a value into a variable. When a value is assigned to a variable as part of the variable's definition, it is called an initialization.

- The `=` symbol is the assignment operator, which copies a value into a variable.
- The data that operators work on are called operands.
- The operand on the left side of `=` must be an *lvalue* (a memory location that can be modified, like a variable).
- The operand on the right side is an *rvalue* (an expression that has a value).
- **Initialization** is assigning a value to a variable as part of its definition.

Program 2-19

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
```

```
5   {
6       int month = 2, days = 28;
7
8       cout << "Month " << month << " has " << days << " days.\n";
9       return 0;
10  }
```

Program Output

Declaring Variables with the `auto` Key Word

- Introduced in C++ 11, the `auto` key word can be used to define a variable.
- It tells the compiler to automatically determine the variable's data type from its initialization value.
- Example: `auto amount = 100;` defines `amount` as an `int`. `auto interestRate = 12.0;` defines `interestRate` as a `double`.

Alternative Forms of Variable Initialization

- Besides using the assignment operator (`int value = 5;`), there are other ways to initialize variables.
- **Parenthesis Notation**: `int value(5);`
- **Brace Notation** (C++ 11): `int value {5};`
  - Brace notation is stricter and will produce a compiler error if you try to initialize a variable with a value of a different type that would cause data loss (e.g., `int value {4.9};`).

2.13 Scope

Concept:

A variable's scope is the part of the program that has access to the variable.

- A variable's scope is the part of the program where it may be used.
- **Rule of Scope**: A variable cannot be used in the program before its definition. The compiler reads code from top to bottom.

Program 2-20

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

int main()
{
    cout << value;

    int value = 100;
    return 0;
}
```

2.14 Arithmetic Operators

Concept:

There are many operators for manipulating numeric values and performing arithmetic operations.

- Operators are categorized by the number of operands they require:
  - **Unary**: One operand (e.g., the negation operator in `-5`).
  - **Binary**: Two operands (e.g., addition, assignment).
  - **Ternary**: Three operands (C++ has one, covered later).

Assignment Statements and Simple Math Expressions

Table 2-9 Fundamental Arithmetic Operators

OPERATOR	MEANING	TYPE	EXAMPLE
`+`	Addition	Binary	`total = cost + tax;`
`-`	Subtraction	Binary	`cost = total - tax;`
`*`	Multiplication	Binary	`tax = cost * rate;`
`/`	Division	Binary	`salePrice = original / 2;`
`%`	Modulus	Binary	`remainder = value % 3;`

- The modulus operator (`%`) works only with integers and returns the remainder of a division.

Program 2-21

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

int main()
{
    double regularWages,
             basePayRate = 18.25,
             regularHours = 40.0,
             overtimeWages,
             overtimePayRate = 27.78,
             overtimeHours = 10,
             totalWages;

    regularWages = basePayRate * regularHours;

    overtimeWages = overtimePayRate * overtimeHours;

    totalWages = regularWages + overtimeWages;

    cout << "Wages for this week are $" << totalWages << endl;
    return 0;
}
```

Program Output

Integer Division

- When both operands of the division operator (`/`) are integers, the result is also an integer. Any fractional part is truncated (discarded).
- For example, `5 / 2` results in `2`.
- To perform a floating-point division, at least one of the operands must be a floating-point type (e.g., `5.0 / 2`).

In the Spotlight:

Calculating Percentages and Discounts

Determining percentages is a common calculation in computer programming. Although the % symbol is used in general mathematics to indicate a percentage, most programming languages (including C++) do not use the % symbol for this purpose. In a program, you have to convert a percentage to a floating-point number, just as you would if you were using a calculator. For example, 50 percent would be written as 0.5, and 2 percent would be written as 0.02.

Let's look at an example. Suppose you earn $6,000 per month and you are allowed to contribute a portion of your gross monthly pay to a retirement plan. You want to determine the amount of your pay that will go into the plan if you contribute 5 percent, 7 percent, or 10 percent of your gross wages. To make this determination, you write the program shown in Program 2-22.

Program 2-22

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

int main()
{
    double monthlyPay = 6000.0, contribution;

    contribution = monthlyPay * 0.05;
    cout << "5 percent is $" << contribution
           << " per month.\n";

    contribution = monthlyPay * 0.07;
    cout << "7 percent is $" << contribution
           << " per month.\n";

    contribution = monthlyPay * 0.1;
    cout << "10 percent is $" << contribution
           << " per month.\n";

    return 0;
}
```

Program Output

Line 11 defines two variables: `monthlyPay` and `contribution.` The `monthlyPay` variable, which is initialized with the value 6000.0, holds the amount of your monthly pay. The `contribution` variable will hold the amount of a contribution to the retirement plan.

The statements in lines 14 through 16 calculate and display 5 percent of the monthly pay. The calculation is done in line 14, where the `monthlyPay` variable is multiplied by 0.05. The result is assigned to the `contribution` variable, which is then displayed in line 15.

Similar steps are taken in lines 18 through 21, which calculate and display 7 percent of the monthly pay, and lines 24 through 26, which calculate and display 10 percent of the monthly pay.

Calculating a Percentage Discount

Another common calculation is determining a percentage discount. For example, suppose a retail business sells an item that is regularly priced at $59.95, and is planning to have a sale where the item's price will be reduced by 20 percent. You have been asked to write a program to calculate the sale price of the item.

To determine the sale price, you perform two calculations:

- First, you get the amount of the discount, which is 20 percent of the item's regular price.
- Second, you subtract the discount amount from the item's regular price. This gives you the sale price.

Program 2-23 shows how this is done in C++.

Program 2-23

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       double regularPrice = 59.95, discount, salePrice;
7
8       discount = regularPrice * 0.2;
9
10      salePrice = regularPrice - discount;
```

```
11
12        cout << "Regular price: $" << regularPrice << endl;
13        cout << "Discount amount: $" << discount << endl;
14        cout << "Sale price: $" << salePrice << endl;
15        return 0;
16    }
```

Program Output

Line 11 defines three variables. The `regularPrice` variable holds the item's regular price, and is initialized with the value 59.95. The `discount` variable will hold the amount of the discount once it is calculated. The `salePrice` variable will hold the item's sale price.

Line 14 calculates the amount of the 20 percent discount by multiplying `regularPrice` by 0.2. The result is stored in the `discount` variable. Line 18 calculates the sale price by subtracting `discount` from `regularPrice.` The result is stored in the `salePrice` variable. The `cout` statements in lines 21 through 23 display the item's regular price, the amount of the discount, and the sale price.

In the Spotlight:

Using the Modulus Operator and Integer Division

The modulus operator (`%`) is surprisingly useful. For example, suppose you need to extract the rightmost digit of a number. If you divide the number by 10, the remainder will be the rightmost digit. For instance, 123 ÷ 10 = 12 with a remainder of 3. In a computer program, you would use the modulus operator to perform this operation. Recall that the modulus operator divides an integer by another integer, and gives the remainder. This is demonstrated in Program 2-24. The program extracts the rightmost digit of the number 12345.

Program 2-24

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int number = 12345;
7       int rightMost = number % 10;
```

```
8
9       cout << "The rightmost digit in "
10              << number << " is "
11              << rightMost << endl;
12
13      return 0;
14  }
```

Program Output

Interestingly, the expression `number % 100` will give you the rightmost two digits in `number`, the expression `number % 1000` will give you the rightmost three digits in `number`, and so on.

The modulus operator (`%`) is useful in many other situations. For example, Program 2-25 converts 125 seconds to an equivalent number of minutes, and seconds.

Program 2-25

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int totalSeconds = 125;
7
8       int minutes, seconds;
9
10      minutes = totalSeconds / 60;
11
12      seconds = totalSeconds % 60;
13
14      cout << totalSeconds << " seconds is equivalent to:\n";
15      cout << "Minutes: " << minutes << endl;
16      cout << "Seconds: " << seconds << endl;
17      return 0;
18  }
```

Program Output

Let's take a closer look at the code:

- Line 8 defines an `int` variable named `totalSeconds`, initialized with the value 125.
- Line 11 declares the `int` variables `minutes` and `seconds`.
- Line 14 calculates the number of minutes in the specified number of seconds. There are 60 seconds in a minute, so this statement divides `totalSeconds` by 60. Notice we are performing integer division in this statement. Both `totalSeconds` and the numeric literal 60 are integers, so the division operator will return an integer result. This is intentional because we want the number of minutes with no fractional part.
- Line 17 calculates the number of remaining seconds. There are 60 seconds in a minute, so this statement uses the `%` operator to divide the `totalSeconds` by 60, and get the remainder of the division. The result is the number of remaining seconds.
- Lines 20 through 22 display the number of minutes and seconds.

Checkpoint

2.21 Is the following assignment statement valid or invalid? If it is invalid, why?

```
72 = amount;
```

2.22 How would you consolidate the following definitions into one statement?

```
int x = 7;
int y = 16;
int z = 28;
```

2.23 What is wrong with the following program? How would you correct it?

```
#include <iostream>
using namespace std;
int main()
{
  number = 62.7;
  double number;
  cout << number << endl;
```

```
  return 0;
}
```

2.24 Is the following an example of integer division or floating-point division? What value will be stored in `portion`?

```
portion = 70 / 3;
```

2.15 Comments

Concept:

Comments are notes of explanation that document lines or sections of a program. Comments are part of the program, but the compiler ignores them. They are intended for people who may be reading the source code.

- Comments are crucial for program documentation, though they have no effect on how the program runs.
- Developing the habit of writing thorough comments is important for maintaining complex code, especially when returning to it after a long time.

Single-Line Comments

- A single-line comment starts with `//`.
- The compiler ignores everything from the `//` to the end of the line.
- They can be used to explain the program's purpose, variable definitions, or complex code sections.

Program 2-26

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6     double payRate;
7     double hours;
8     int employNumber;
9      (The remainder of this program is left out.)
```

Multi-Line Comments

- A multi-line comment starts with `/*` and ends with `*/`.
- Everything between these two markers is ignored by the compiler.
- This type of comment can span several lines, making it convenient for writing large blocks of text.

Program 2-27

```
1   /*
2       PROGRAM: PAYROLL.CPP
3       Written by Herbert Dorfmann
4       This program calculates company payroll
5       Last modification: 8/20/2017
6   */
7
8   #include <iostream>
9   using namespace std;
10
11  int main()
12  {
13      double payRate;
14      double hours;
15      int employNumber;
16     (The remainder of this program is left out.)
```

Note:

Many programmers use a combination of single-line comments and multi-line comments in their programs. Convenience usually dictates which style to use.

2.16 Named Constants

Concept:

Literals may be given names that symbolically represent them in a program.

- Using unnamed literal values (magic numbers) in code can cause two problems:
  - It makes the code hard to understand (e.g., what does `0.069` represent?).

    - If the value needs to be changed, you must find and update every occurrence of it throughout the program.
- **Named constants** solve these problems. A named constant is a variable whose content is read-only and cannot be changed while the program runs.
- Define a named constant using the `const` qualifier before the data type.
- A value must be assigned upon definition.
- By convention, named constants are written in all uppercase letters to distinguish them from regular variables.
- Using named constants makes programs self-documenting and easy to update.

Program 2-28

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       const double PI = 3.14159;
7       const double DIAMETER = 10.0;
8
9       double circumference;
10
11      circumference = PI * DIAMETER;
12
13      cout << "The circumference is: " << circumference << endl;
14      return 0;
15  }
```

Program Output

Checkpoint

2.25 Write statements using the `const` qualifier to create named constants for the following literal values:

LITERAL VALUE	DESCRIPTION
2.71828	Euler's number (known in mathematics as *e*)

LITERAL VALUE	DESCRIPTION
5.256E5	Number of minutes in a year
32.2	The gravitational acceleration constant (in ft/s^2)
9.8	The gravitational acceleration constant (in m/s^2)
1609	Number of meters in a mile

2.17 Programming Style

Concept:

Programming style refers to the way a programmer uses identifiers, spaces, tabs, blank lines, and punctuation characters to visually arrange a program's source code. These are some, but not all, of the elements of programming style.

- While syntax rules are mandatory for the compiler, programming style is for human readability.
- The compiler processes code as a continuous stream, ignoring stylistic elements like indentation and spacing.
- A good programming style uses visual cues to make the code easier for people to read and understand.

Program 2-29

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

Program Output

Program 2-30

```cpp
#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**

- Elements of good style include:
  - Indenting lines of code within braces (`{}`).
  - Using blank lines to visually separate logical sections of code (e.g., variable definitions from executable statements).
  - Breaking long statements across multiple lines for better readability.

Note:

Although you are free to develop your own style, you should adhere to common programming practices. By doing so, you will write programs that visually make sense to other programmers.

Review Questions and Exercises

Short Answer

1. How many operands does each of the following types of operators require?
   - _____ Unary
   - _____ Binary
   - _____ Ternary

2. How may the `double` variables `temp`, `weight`, and `age` be defined in one statement?

3. How may the `int` variables `months`, `days`, and `years` be defined in one statement, with `months` initialized to 2 and `years` initialized to 3?

4. Write assignment statements that perform the following operations with the variables `a`, `b`, and `c`:

    1. Adds 2 to `a` and stores the result in `b`.

    2. Multiplies `b` by 4 and stores the result in `a`.

    3. Divides `a` by 3.14 and stores the result in `b`.

    4. Subtracts 8 from `b` and stores the result in `a`.

    5. Stores the value 27 in a.

    6. Stores the character 'K' in `c`.

    7. Stores the ASCII code for 'B' in `c`.

5. Is the following comment written using single-line or multi-line comment symbols?

```
/* This program was written by M. A. Codewriter*/
```

6. Is the following comment written using single-line or multi-line comment symbols?

```

```

7. Modify the following program so it prints two blank lines between each line of text.

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Two mandolins like creatures in the";
    cout << "dark";
    cout << "Creating the agony of ecstasy.";
    cout << "                   - George Barker";
    return 0;
}
```

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

1.
```
#include <iostream>
using namespace std;
int main()
{
    int freeze = 32, boil = 212;
    freeze = 0;
    boil = 100;
    cout << freeze << endl << boil << endl;
    return 0;
}
```

2.
```
#include <iostream>
using namespace std;
int main()
{
    int x = 0, y = 2;
    x = y * 4;
    cout << x << endl << y << endl;
    return 0;
}
```

3.
```
#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;
}
```

4.
```
#include <iostream>
using namespace std;
int main()
{
    cout << "Be careful\n";
    cout << "This might/n be a trick ";
    cout << "question\n";
    return 0;
}
```

5.
```cpp
#include <iostream>
using namespace std;
int main()
{
    int a, x = 23;
    a = x % 2;
    cout << x << endl << a << endl;
    return 0;
}
```

Multiple Choice

1. Every complete statement ends with a(n) _____.
    1. `period`
    2. `# symbol`
    3. `semicolon`
    4. `ending brace`
2. Which of the following statements is correct?
    1. `#include (iostream)`
    2. `#include {iostream}`
    3. `#include <iostream>`
    4. `#include [iostream]`
    5. All of the above
3. Every C++ program must have a _____.
    1. `cout` statement
    2. function `main`
    3. `#include` statement
    4. All of the above
4. Preprocessor directives begin with _____.

    1. `#`
    2. `!`
    3. `<`
    4. `*`
    5. None of the above
5. The following data

```
72
'A'
"Hello World"
2.8712
```

   are all examples of _____.

    1. variables
    2. literals or constants
    3. strings
    4. none of the above

6. A group of statements, such as the contents of a function, is enclosed in _____.

    1. braces `{}`
    2. parentheses `()`
    3. brackets `<>`
    4. all of the above will do

7. Which of the following are *not* valid assignment statements? (Select all that apply.)

    1. `total = 9;`
    2. `72 = amount;`
    3. `profit = 129`
    4. `letter = 'W';`

8. Which of the following are *not* valid `cout` statements? (Select all that apply.)
    1. `cout << "Hello World";`
    2. `cout << "Have a nice day"\n;`
    3. `cout < value;`
    4. `cout << Programming is great fun;`
9. Assume `w` = 5, `x` = 4, `y` = 8, and `z` = 2. What value will be stored in `result` in each of the following statements?
    1. `result = x + y;`
    2. `result = z * 2;`
    3. `result = y / x;`
    4. `result = y - z;`
    5. `result = w % 2;`
10. How would each of the following numbers be represented in E notation?
    1. 3.287×10^{6}
    2. -978.65×10^{12}
    3. 7.65491×10^{-3}
    4. -58710.23×10^{-4}
11. The negation operator is _____.
    1. unary
    2. binary
    3. ternary
    4. none of the above
12. A(n) _____ is like a variable, but its value is read-only and cannot be changed during the program's execution.

    1. secure variable
    2. uninitialized variable
    3. named constant
    4. locked variable
13. When do preprocessor directives execute?
    1. Before the compiler compiles your program
    2. After the compiler compiles your program
    3. At the same time as the compiler compiles your program
    4. None of the above

True or False

1. T F A variable must be defined before it can be used.
2. T F Variable names may begin with a number.
3. T F Variable names may be up to 31 characters long.
4. T F A left brace in a C++ program should always be followed by a right brace later in the program.
5. T F You cannot initialize a named constant that is declared with the `const` modifier.

Algorithm Workbench

1. 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.
2. Convert the following pseudocode to C++ code. Be sure to define the appropriate variables.

- Store 172.5 in the `force` variable.
- Store 27.5 in the `area` variable.
- Divide area by `force` and store the result in the `pressure` variable.
- Display the contents of the `pressure` variable.

Find the Error

1. There are a number of syntax errors in the following program. Locate as many as you can.

```
*/ What's wrong with this program? /*
#include iostream
using namespace std;
int main();
}
   int a, b, c \\ Three integers
   a = 3
   b = 4
   c = a + b
   Cout < "The value of c is %d" < C;
   return 0;
{
```

Programming Challenges

Visit www.myprogramminglab.com to complete many of these Programming Challenges online and get instant feedback.

1. Sum of Two Numbers

   Write a program that stores the integers 50 and 100 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 58 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 $8.6 million in sales this year.

3. Sales Tax

   Write a program that will compute the total sales tax on a $95 purchase. Assume the state sales tax is 4 percent, and the county sales tax is 2 percent.

   Solving the Restaurant Bill Problem

4. Restaurant Bill

   Write a program that computes the tax and tip on a restaurant bill for a patron with a $88.67 meal charge. The tax should be 6.75 percent of the meal cost. The tip should be 20 percent of the total after adding the tax. Display the meal cost, tax amount, tip amount, and total bill on the screen.

5. Average of Values

   To get the average of a series of values, you add the values up then divide the sum by the number of values. Write a program that stores the following values in five different variables: 28, 32, 37, 24, and 33. The program should first calculate the sum of these five variables and store the result in a separate variable named `sum`. Then, the program should divide the `sum` variable by 5 to get the average. Display the average on the screen.

    **Tip:**

   Use the `double` data type for all variables in this program.

6. Annual Pay

   Suppose an employee gets paid every two weeks and earns $2,200 each pay period. In a year, the employee gets paid 26 times. Write a program that defines the following variables:

`payAmount`	This variable will hold the amount of pay the employee earns each pay period. Initialize the variable with 2200.0.

`payPeriods`	This variable will hold the number of pay periods in a year. Initialize the variable with 26.
`annualPay`	This variable will hold the employee's total annual pay, which will be calculated.

The program should calculate the employee's total annual pay by multiplying the employee's pay amount by the number of pay periods in a year and store the result in the `annualPay` variable. Display the total annual pay on the screen.

7. Ocean Levels

   Assuming the ocean's level is currently rising at about 1.5 millimeters per year, write a program that displays:

   - The number of millimeters higher than the current level that the ocean's level will be in 5 years.
   - The number of millimeters higher than the current level that the ocean's level will be in 7 years.
   - The number of millimeters higher than the current level that the ocean's level will be in 10 years.

8. Total Purchase

   A customer in a store is purchasing five items. The prices of the five items are as follows:

   - Price of item 1 = $15.95
   - Price of item 2 = $24.95
   - Price of item 3 = $6.95
   - Price of item 4 = $12.95
   - Price of item 5 = $3.95

   Write a program that holds the prices of the five items in five variables. Display each item's price, the subtotal of the sale, the amount of sales tax, and the total. Assume the sales tax is 7 percent.

9. Cyborg Data Type Sizes

You have been given a job as a programmer on a Cyborg supercomputer. In order to accomplish some calculations, you need to know how many bytes the following data types use: `char`, `int`, `float`, and `double`. You do not have any technical documentation, so you can't look this information up. Write a C++ program that will determine the amount of memory used by these types and display the information on the screen.

10. Miles per Gallon

    A car holds 15 gallons of gasoline and can travel 375 miles before refueling. Write a program that calculates the number of miles per gallon the car gets. Display the result on the screen.

    Hint: Use the following formula to calculate miles per gallon (MPG):

    > MPG 5 Miles Driven/Gallons of Gas Used

11. Distance per Tank of Gas

    A car with a 20-gallon gas tank averages 23.5 miles per gallon when driven in town, and 28.9 miles per gallon when driven on the highway. Write a program that calculates and displays the distance the car can travel on one tank of gas when driven in town and when driven on the highway.

    Hint: The following formula can be used to calculate the distance:

    > Distance 5 Number of Gallons 3 Average Miles per Gallon

12. Land Calculation

    One acre of land is equivalent to 43,560 square feet. Write a program that calculates the number of acres in a tract of land with 391,876 square feet.

13. Circuit Board Price

    An electronics company sells circuit boards at a 35 percent profit. Write a program that will calculate the selling price of a circuit board that costs $14.95. Display the result on the screen.

14. Personal Information

    Write a program that displays the following pieces of information, each on a separate line:

- Your name
- Your address, with city, state, and ZIP code
- Your telephone number
- Your college major

Use only a single `cout` statement to display all of this information.

15. Triangle Pattern

    Write a program that displays the following pattern on the screen:

```
   *
  ***
 *****
*******
```

16. Diamond Pattern

    Write a program that displays the following pattern:

```
    *
   ***
  *****
 *******
  *****
   ***
    *
```

17. Stock Commission

    Kathryn bought 750 shares of stock at a price of $35.00 per share. She must pay her stockbroker 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 16,500 of its customers and found that approximately 15 percent of those surveyed purchase one or more energy drinks per week. Of those customers who purchase energy drinks, approximately 58 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.

19. Annual High Temperatures

    The average July high temperature is 85 degrees Fahrenheit in New York City, 88 degrees Fahrenheit in Denver, and 106 degrees Fahrenheit in Phoenix. Write a program that calculates and reports what the new average July high temperature would be for each of these cities if temperatures rise by 2 percent.

20. How Much Paint

    A particular brand of paint covers 340 square feet per gallon. Write a program to determine and report approximately how many gallons of paint will be needed to paint two coats on a wooden fence that is 6 feet high and 100 feet long.