Chapter 2: Introduction to C++

2.1 The Parts of a C++ Program

The Parts of a C++ Program

// sample C++ program —— comment
#include <iostream> —— preprocessor directive
using namespace std; —— which namespace to use
int main() —— beginning of function named main
{
    cout << "Hello, there!"; —— output statement
    return 0; —— Send 0 to operating system
} —— end of block for main

Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>//</td>
<td>Double slash</td>
<td>Beginning of a comment</td>
</tr>
<tr>
<td>#</td>
<td>Pound sign</td>
<td>Beginning of preprocessor directive</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Open/close brackets</td>
<td>Enclose filename in #include</td>
</tr>
<tr>
<td>()</td>
<td>Open/close parentheses</td>
<td>Used when naming a function</td>
</tr>
<tr>
<td>{}</td>
<td>Open/close brace</td>
<td>Encloses a group of statements</td>
</tr>
<tr>
<td>&quot; &quot;</td>
<td>Open/close quotation marks</td>
<td>Encloses string of characters</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon</td>
<td>End of a programming statement</td>
</tr>
</tbody>
</table>

2.2 The cout Object

Displays output on the computer screen

You use the stream insertion operator << to send output to cout:

```cpp
cout << "Programming is fun!";
```
The cout Object

- Can be used to send more than one item to cout:
  
  ```cpp
  cout << "Hello " << "there!";
  ```

  Or:
  
  ```cpp
  cout << "Hello ";
  cout << "there!";
  ```

The endl Manipulator

- You can use the `endl` manipulator to start a new line of output. This will produce two lines of output:
  
  ```cpp
  cout << "Programming is " << endl;
  cout << "fun!";
  ```

You do NOT put quotation marks around `endl`

- The last character in `endl` is a lowercase L, not the number 1.

  `endl` — This is a lowercase L

The \n Escape Sequence

- You can also use the `\n` escape sequence to start a new line of output. This will produce two lines of output:

  ```cpp
  cout << "Programming is\n";
  cout << "fun!";
  ```

Notice that the `\n` is INSIDE the string.
The \n Escape Sequence

```cpp
#include <iostream>

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

2.3

The \#include Directive

- Inserts the contents of another file into the program
- This is a preprocessor directive, not part of C++ language
- \#include lines not seen by compiler
- Do not place a semicolon at end of \#include line

2.4

Variables and Literals

- Variable: a storage location in memory
  - Has a name and a type of data it can hold
  - Must be defined before it can be used:

```
int item;
```
Literals

- **Literal**: a value that is written into a program’s code.

  "hello, there" (string literal)

  12 (integer literal)

Integer Literal in Program 2-9

```cpp
# Program 2-9
1 // This program has literals and a variable.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int apples;
8 
9     cout << "Today we sold " << apples << " bushels of apples."
10     return 0;
11 }
```

Program Output

Today we sold 20 bushels of apples.

String Literals in Program 2-9

```cpp
# Program 2-9
1 // This program has literals and a variable.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int apples;
8 
9     cout << "Today we sold " << apples << " bushels of apples."
10     return 0;
11 }
```

Program Output

Today we sold 20 bushels of apples.

Identifiers

- An identifier is a programmer-defined name for some part of a program: variables, functions, etc.

C++ Key Words

<table>
<thead>
<tr>
<th>C++ Key Words</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>break</code></td>
</tr>
<tr>
<td><code>infinite</code></td>
</tr>
<tr>
<td><code>register</code></td>
</tr>
<tr>
<td><code>template</code></td>
</tr>
<tr>
<td><code>void</code></td>
</tr>
</tbody>
</table>

You cannot use any of the C++ key words as an identifier. These words have reserved meaning.
Variable Names

- A variable name should represent the purpose of the variable. For example:

  \texttt{itemsOrdered}

  The purpose of this variable is to hold the number of items ordered.

Identifier Rules

- The first character of an identifier must be an alphabetic character or and underscore (_).
- After the first character you may use alphabetic characters, numbers, or underscore characters.
- Upper- and lowercase characters are distinct.

Valid and Invalid Identifiers

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>VALID?</th>
<th>REASON IF INVALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalSales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total_Sales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total.Sales</td>
<td>No</td>
<td>Cannot contain .</td>
</tr>
<tr>
<td>4thQtrSales</td>
<td>No</td>
<td>Cannot begin with digit</td>
</tr>
<tr>
<td>totalSale$</td>
<td>No</td>
<td>Cannot contain $</td>
</tr>
</tbody>
</table>

Integer Data Types

- Integer variables can hold whole numbers such as 12, 7, and -99.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Typical Size</th>
<th>Typical Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>short int</td>
<td>2 bytes</td>
<td>-32,768 to +32,767</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>2 bytes</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648 to +2,147,483,647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4 bytes</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>long int</td>
<td>8 bytes</td>
<td>-2,147,483,648 to +2,147,483,647</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>8 bytes</td>
<td>0 to 4,294,967,295</td>
</tr>
</tbody>
</table>

Defining Variables

- Variables of the same type can be defined on separate lines:
  - int length;
  - int width;
  - unsigned int area;
- On the same line:
  - int length, width;
  - unsigned int area;
- Variables of different types must be in different definitions.
Integer Types in Program 2-10

Program 2.10

This program has three variables: checking, miles, and days

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

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

  checking = 10;
  miles = 276;
  days = 12390;

  cout << "We have made a long journey of " << miles;
  cout << " miles.\n";
  cout << "Our checking account balance is " << checking;
  cout << " dollars and " << days << " days ago\n";
}
```

Integer Literals in Program 2-10

- Integer literals are stored in memory as `int` by default
- To store an integer constant in a long memory location, put 'L' at the end of the number: `1234L`
- To store an integer constant in a long long memory location, put 'LL' at the end of the number: `3241LL`
- Constants that begin with '0' (zero) are base 8: `075`
- Constants that begin with '0x' are base 16: `0x75A`

The `char` Data Type

- Used to hold characters or very small integer values
- Usually 1 byte of memory
- Numeric value of character from the character set is stored in memory:
Character Literals

- Character literals must be enclosed in single quote marks. Example:
  
  'A'

Character Strings

- A series of characters in consecutive memory locations: "Hello"
- Stored with the null terminator, \0, at the end:
- Comprised of the characters between the " 

Character Literals in Program 2-14

Program 2-14

```c++
1 // This program uses character literals.
2 #include <iostream>
3 using namespace std;
4 int main()
5 {
6     char letter;
7     letter = 'A';
8     cout << letter << 'a';
9     letter = 'B';
10    cout << letter << 'b';
11    return 0;
12 }
```

Program Output

A B

Character Strings

- A series of characters in consecutive memory locations: "Hello"
- Stored with the null terminator, \0, at the end:
- Comprised of the characters between the " 

The C++ string Class

- Special data type supports working with strings
  ```c++
  #include <string>
  ```
- Can define string variables in programs:
  ```c++
  string firstName, lastName;
  ```
- Can receive values with assignment operator:
  ```c++
  firstName = "George";
  lastName = "Washington";
  ```
- Can be displayed via cout
  ```c++
  cout << firstName << " " << lastName;
  ```

The string class in Program 2-15

Program 2-15

```c++
1 // This program demonstrates the string class.
2 #include <iostream>
3 #include <string> // Required for the string class.
4 using namespace std;
5 int main()
6 {
7     string movieTitle;
8     movieTitle = "Wheels of Fury";
9     cout << "My favorite movie is " << movieTitle << endl;
10    return 0;
11 }
```

Program Output

My favorite movie is Wheels of Fury
2.9 Floating-Point Data Types

The floating-point data types are:
- `float`
- `double`
- `long double`

They can hold real numbers such as:
- `12.45`
- `-3.8`

Stored in a form similar to scientific notation

All floating-point numbers are signed

Floating-Point Literals

- Can be represented in
  - Fixed point (decimal) notation:
    - `31.4159`
    - `0.0000625`
  - E notation:
    - `3.14159E1`
    - `6.25e-5`

- Are double by default

- Can be forced to be float (3.14159f) or long double (0.0000625L)

Floating-Point Data Types in Program 2-16

Program 2-16

```cpp
1 // This program uses floating point data types.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7    float distance;
8    double mass;
9    
10    distance = 1.495978E11;
11    mass = 1.989E30;
12    cout << "The Sun is " << distance << " meters away.\n";
13    cout << "The Sun's mass is " << mass << " kilograms.\n";
14    return 0;
15 }
```

Program Output

The Sun is 1.49598e+11 meters away.
The Sun's mass is 1.989e+30 kilograms.

2.10 The `bool` Data Type
The bool Data Type

- Represents values that are true or false
- bool variables are stored as small integers
- false is represented by 0, true by 1:

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

Boolean Variables in Program 2-17

```cpp
Program 2-17
1  // This program demonstrates bool variables.
2 #include <iostream>
3 using namespace std;
4 int main()
5 {
6    bool boolValue;
7    bool boolValue = true;
8    cout << boolValue << endl;
9    boolValue = false;
10   cout << boolValue << endl;
11   return 0;
12 }
```

Program Output

```
1 0
```

Determining the Size of a Data Type

- The sizeof operator gives the size of any data type or variable:

```cpp
double amount;
cout << "A double is stored in "
                   << sizeof(double) "bytes\n";
cout << "Variable amount is stored in "
                   << sizeof(amount) "bytes\n";
```

Variable Assignments and Initialization

- An assignment statement uses the = operator to store a value in a variable.

```cpp
item = 12;
```
- This statement assigns the value 12 to the item variable.
Assignment

- The variable receiving the value must appear on the left side of the = operator.
- This will NOT work:
  ```cpp
  // ERROR!
  12 = item;
  ```

Variable Initialization

- To initialize a variable means to assign it a value when it is defined:
  ```cpp
  int length = 12;
  ```
- Can initialize some or all variables:
  ```cpp
  int length = 12, width = 5, area;
  ```

Variable Initialization in Program 2-19

```cpp
Program 2-19
1 // This program shows variable initialization.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7   int month = 2, days = 28;
8   cout << “Month “ << month << “ has “ << days << “ days.
9   return 0;
10 }
```

Program Output

Month 2 has 28 days.

Declaring Variables With the `auto` Key Word

- C++11 introduces an alternative way to define variables, using the `auto` key word and an initialization value. Here is an example:
  ```cpp
  auto amount = 100; // int
  auto interestRate = 12.0; // double
  auto stockCode = 'D'; // char
  auto customerNum = 459L; // long
  ```

Scope

- The scope of a variable: the part of the program in which the variable can be accessed.
- A variable cannot be used before it is defined.
Variable Out of Scope in Program 2-20

Program 2-20

```cpp
1. // This program can't find its variable.
2. #include <iostream>
3. using namespace std;
4.
5. int main()
6. {
7.  cout << value; // ERROR! value not defined yet!
8.  int value = 600;
9.  return 0;
10. }
```

2.14 Arithmetic Operators

- **Arithmetic Operators**
  - Used for performing numeric calculations
  - C++ has unary, binary, and ternary operators:
    - **unary (1 operand)**
      - `-`
    - **binary (2 operands)**
      - `13 - 7`
    - **ternary (3 operands)**
      - `exp1 ? exp2 : exp3`

**Binary Arithmetic Operators**

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>OPERATION</th>
<th>EXAMPLE</th>
<th>VALUE OF <code>ans</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td><code>ans = 7 + 3;</code></td>
<td>10</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td><code>ans = 7 - 3;</code></td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td><code>ans = 7 * 3;</code></td>
<td>21</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td><code>ans = 7 / 3;</code></td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
<td><code>ans = 7 % 3;</code></td>
<td>1</td>
</tr>
</tbody>
</table>

A Closer Look at the `/` Operator

- `/` (division) operator performs integer division if both operands are integers
  - `cout << 13 / 5; // displays 2`
  - `cout << 91 / 7; // displays 13`
- If either operand is floating point, the result is floating point
  - `cout << 13 / 5.0; // displays 2.6`
  - `cout << 91.0 / 7; // displays 13.0`
A Closer Look at the % Operator

- % (modulus) operator computes the remainder resulting from integer division
  
  \[ \text{cout} \ll 13 \% 5; \quad \text{// displays 3} \]

- % requires integers for both operands
  
  \[ \text{cout} \ll 13 \% 5.0; \quad \text{// error} \]

Comments

- Used to document parts of the program
- Intended for persons reading the source code of the program:
  - Indicate the purpose of the program
  - Describe the use of variables
  - Explain complex sections of code
- Are ignored by the compiler

Single-Line Comments

- Begin with // through to the end of line:
  
  int length = 12; \quad \text{// length in inches} 
  int width = 15; \quad \text{// width in inches} 
  int area; \quad \text{// calculated area} 
  
  \text{// calculate rectangle area} 
  area = length * width;

Multi-Line Comments

- Begin with /*, end with */
- Can span multiple lines:
  
  /* this is a multi-line comment */

- Can begin and end on the same line:
  
  int area; \quad /* calculated area */

Named Constants
Named Constants

- **Named constant (constant variable):** variable whose content cannot be changed during program execution.
- **Used for representing constant values with descriptive names:**
  - `const double TAX_RATE = 0.0675;`
  - `const int NUM_STATES = 50;`
- **Often named in uppercase letters**

### Named Constants in Program 2-28

```c++
1 // This program calculates the circumference of a circle.
2 #include <iostream>
3 using namespace std;
4 
5 int main()
6 {
7    // Constants
8    const double PI = 3.14159;
9    const double DIAMETER = 10.0;
10   
11    // Variable to hold the circumference
12    double circumference;
13   
14    // Calculate the circumference.
15    circumference = PI * DIAMETER;
16   
17    // Display the circumference.
18    cout << "The circumference is: " << circumference << endl;
19    return 0;
20 }
```

### Program Output

```
The circumference is: 31.4159
```

Programming Style

- **The visual organization of the source code**
- **Includes the use of spaces, tabs, and blank lines**
- **Does not affect the syntax of the program**
- **Affects the readability of the source code**

2.17 Programming Style