Basic Elements of C++ Programming Language

Identifiers

- An identifier is the name of C/C++ entities such as variables and functions.
- Rules for writing valid identifiers:
  1. An identifier must begin with a letter of the alphabet (a to z or A to Z), or the underscore character (_).
  2. After the first character, an identifier may include any letter of the alphabet, any digit (0 to 9), or the underscore character (_).
  3. There is no limit on the number of characters that an identifier may have. But, most compilers only recognize the first 31 characters as significant.
- C/C++ is case-sensitive: identifiers Sum, sum, and SUM are all distinct.
- By convention, if an identifier consists of two or more words you may do one of the following:
  - Separate the words with the underscore characters.
    Examples: interest_rate sum_of_dice
  - Concatenate all the words and capitalize the first character of each word except the first.
    Examples: interestRate sumOfDice

Keywords

- Keywords are words that already have a meaning for the compiler.
- Keywords must not be used as identifiers.
- Examples of keywords are: auto, do, while, int, float, long, and const.

Standard identifiers

- A standard identifier is a word used to name an object that comes with the compiler.
  Examples: scanf and printf
- You may use a standard identifier as an identifier only if you do not want to use the object that comes with the compiler.
Variables and Basic Data Types

- The functions of a digital computer are:
  - To input information
  - To process it,
  - And to produce the result(s)
- The information input by the computer is stored in its (main) memory into a **memory location**.
- In a high-level language program, a memory location is represented by a **variable**.
- A variable has:
  - A name,
  - A data type
  - And an address.
- The data type of a variable lets the compiler know the type of data that will be stored in the corresponding memory location.
- The number of bytes reserved for a memory location depends on the data type of the corresponding variable.
- The following table provides the basic data types and their ranges of values on IBM PC and compatible computers.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Type of Data</th>
<th>Range of Values</th>
<th>Size on PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>Boolean value</td>
<td>true, false</td>
<td>1 byte</td>
</tr>
<tr>
<td>char</td>
<td>A single character</td>
<td>ASCII character representations</td>
<td>1 byte</td>
</tr>
<tr>
<td>short int</td>
<td>integers</td>
<td>-2¹⁵ (-32,768) to 2¹⁵ – 1 (32,767)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>integers</td>
<td>0 to 2¹⁶ – 1 (65,535)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>int</td>
<td>integers</td>
<td>-2³¹ (-2,147,483,648) to 2³¹ – 1 (2,147,483,647)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>unsigned int</td>
<td>integers</td>
<td>0 to 2³² – 1 (4,294,967,295)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>long int</td>
<td>integers</td>
<td>-2³¹ (-2,147,483,648) to 2³¹ – 1 (2,147,483,647)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>integers</td>
<td>0 to 2³² – 1 (4,294,967,295)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>float</td>
<td>Floating point decimals</td>
<td>Negative range: -3.4028235E+38 to -1.4E-45</td>
<td>4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Positive range: 1.4E-45 to 3.4028235E+38</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>Floating point decimals</td>
<td>Negative range: -1.7976931348623157E+308 to -4.9E-324</td>
<td>8 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Positive range: 4.9E-324 to 1.7976931348623157E+308</td>
<td></td>
</tr>
<tr>
<td>long double</td>
<td>Floating point decimals</td>
<td>Negative range: -1.18E+4932 to -3.37E-4932</td>
<td>10 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Positive range: 3.37E-4932 to 1.18E+4932</td>
<td></td>
</tr>
</tbody>
</table>
The size and the range of values of a basic data type depend on the computer that you are using.

You can use the `sizeof()` function to find out the size of a basic data type on a computer. The following statement displays the size of `int`, `long`, and `double` data types:

```c
cout << sizeof(int) << ‘t’ << sizeof(long) << ‘t’ << sizeof(double);
```

Floating point decimal values are approximated inside a computer. The **precision of a floating point data type** specifies the number of significant digits in the representation of a value in a memory location of that data type. The precisions of floating-point data types are provided as follows:

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>7 digits</td>
</tr>
<tr>
<td>double</td>
<td>15 digits</td>
</tr>
<tr>
<td>Long double</td>
<td>19 digits</td>
</tr>
</tbody>
</table>

### Declaration Statement

- You define a variable by using a **declaration statement** as follows:

  ```c
  <data-type> <variable-name>;
  ```

  **Examples:**
  ```c
  int num;
  unsigned index;
  bool flag;
  double taxRate;
  ```

- Two or more variables with the same data type can be defined in the same declaration statement: for example, integer variables `num`, `number`, and `sum` can be defined as follows:

  ```c
  int num, number, sum;
  ```

- It is a good programming practice to follow each variable definition with a comment in order to remind yourself and anybody else reading the program the purpose of that variable:

  ```c
  int num, // to hold the first value
       number, // to hold the second value
              sum;   // to hold their sum
  ```

- A **declaration statement** tells the compiler to reserve memory location(s) for the specified variable(s).
Constant Data

- In addition to the input data, the data to be processed in a C/C++ program can also be specified in the text of the program.
- The data that you specify in the text of a program are referred to as constant data.
- There are five types of constant data in C++:
  - character constants,
  - integer constants,
  - floating-point constants,
  - strings constants, and
  - Boolean constants: true, false.

Character Constants

- A character constant is either a single printable character enclosed between single quotes, or an escape sequence enclosed between single quotes.
  
  Examples of printable characters: 'A', '$', '8', ‘ ’ (space bar), 'z'.
  
  Examples of escape sequences: '\n', '\t', '\', '\w', ‘\''.

- Escape sequences are used to represent characters that you can not type from the keyboard or characters that have a meaning for the compiler such as single quote, double quote, . . . , etc.

Commonly used Escape Sequences

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Character</th>
<th>ASCII Code</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>BEL</td>
<td>0000 0111</td>
<td>Audible bell</td>
</tr>
<tr>
<td>\b</td>
<td>BS</td>
<td>0000 1000</td>
<td>Backspace</td>
</tr>
<tr>
<td>\f</td>
<td>FF</td>
<td>0000 1100</td>
<td>Formfeed</td>
</tr>
<tr>
<td>\n</td>
<td>LF</td>
<td>0000 1010</td>
<td>Newline (linefeed)</td>
</tr>
<tr>
<td>\r</td>
<td>CR</td>
<td>0000 1101</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>HT</td>
<td>0000 1001</td>
<td>Tab (horizontal)</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>0101 1100</td>
<td>Backslash</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
<td>0010 1100</td>
<td>Single quote (apostrophe)</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>0010 0010</td>
<td>Double quote</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>0011 1111</td>
<td>Question mark</td>
</tr>
<tr>
<td>\0</td>
<td>0</td>
<td>0000 0000</td>
<td>Null character</td>
</tr>
<tr>
<td>%</td>
<td>%</td>
<td>0010 0101</td>
<td>percent sign</td>
</tr>
</tbody>
</table>
Integer Constants

- An integer constant is a positive or negative whole number.
- A positive integer constant may be preceded with a plus (+) sign.
- Commas, decimal point and special symbols such as the dollar sign are not allowed.
- Examples of valid decimal integer constants are 0, -5, 643, -72, and +23.
- A leading zero such as in 024 is not allowed in a decimal integer constant.
- The range of integer values that may be represented depends on the computer.
- On IBM PC and compatible computers, the range of integer values that may be represented is -2,147,483,648 to 2,147,483,647.

Floating-point constants

- A floating-point constant may either be written in
  - decimal notation or
  - exponential notation.
- Decimal Notation:  \(<\text{decimal integer}>.\text{<decimal fraction>}\>
- Exponential Notation:
  \(<\text{decimal integer}>.\text{<decimal fraction>}E\text{<decimal integer>}>
  \text{or}
  \(<\text{decimal integer}>.\text{<decimal fraction>}e\text{<decimal integer>}>

- Either the <\text{decimal integer}> part, or the <\text{decimal fraction}> part, may be omitted, but not both.
- The letter e or E in the exponential notation stands for 10 and the number following it is a signed integer that represents the exponent of the number.
- The period and the <\text{decimal fraction}> part may also be omitted in the exponential notation.

Examples of Valid Floating-Point Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>.5</td>
<td>0.5</td>
</tr>
<tr>
<td>38.</td>
<td>38.0</td>
</tr>
<tr>
<td>-6.2</td>
<td>-6.2</td>
</tr>
<tr>
<td>0.</td>
<td>0</td>
</tr>
<tr>
<td>.0</td>
<td>0</td>
</tr>
<tr>
<td>23.45e6</td>
<td>23.45 x 10^6</td>
</tr>
<tr>
<td>2E-5</td>
<td>2.0 x 10^-5</td>
</tr>
<tr>
<td>-3.45E8</td>
<td>-3.45 x 10^8</td>
</tr>
<tr>
<td>.45E+12</td>
<td>0.45 x 10^{12}</td>
</tr>
</tbody>
</table>
String constants

- A string constant is a sequence of zero or more characters (including escape sequences) enclosed between double quotes.
- String constants form a special class of constants because they do not correspond to a basic data type.
- Inside the computer, the characters of a string constant are represented in consecutive bytes with a null (\0) terminating character.

Examples of String Constants

<table>
<thead>
<tr>
<th>String Constants</th>
<th>Length</th>
<th>Representation in Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A&quot;</td>
<td>1</td>
<td>A \0</td>
</tr>
<tr>
<td>&quot;5g&quot;</td>
<td>2</td>
<td>5 g \0</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>0</td>
<td>\0</td>
</tr>
<tr>
<td>&quot;\nJo's mug&quot;</td>
<td>9</td>
<td>\n J o ' s m u g \0</td>
</tr>
</tbody>
</table>

- The length of a string constant is the number of characters that it contains.
- The Null string is the string with no character. Its length is zero.

Assignment Statement

- The assignment statement: <variable-name> = <value>;
  is used to store <value> into the memory location represented by the variable <variable-name>.

Examples:

```
char letter;
int num1;
double dnum;
letter = 'A';
num1 = 25;
dnum = 4.25;
um1 = 10;
```

- When you store a new value into a memory location, any value that was previously there is destroyed.
Initial value of a Variable

- The **initial value** of a variable is the value that you specify when that variable is defined as follows:

  ```
  <data-type> <variable-name> = <initial-value>;
  ```

**Example:**  The statement:

```
int num1 = 25;
```

says the same thing as the following sequence of two statements:

```
int num1;
num1 = 25;
```

- Initial values may also be specified in a declaration statement in which two or more variables are defined as follows:

  a)  ```
  int num1 = 25, num2, sum = 0;
  ```

  b)  ```
  char letter, grade = 'A';
  ```

**Quiz #1:** Do exercise 1.

### Arithmetic Expressions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
<th>Example</th>
<th>Operand</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>A + B</td>
<td>Both integers</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>One operand is not integer</td>
<td>Double precision floating-point</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>A – B</td>
<td>Both integers</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>One operand is not integer</td>
<td>Double precision floating-point</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>A * B</td>
<td>Both integers</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>One operand is not integer</td>
<td>Double precision floating-point</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>A / B</td>
<td>Both integers</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>One operand is not integer</td>
<td>Double precision floating-point</td>
</tr>
<tr>
<td>Modulus (Remainder)</td>
<td>%</td>
<td>A % B</td>
<td>Both integers</td>
<td>Integer</td>
</tr>
<tr>
<td>Negation</td>
<td>-</td>
<td>-A</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>is not integer</td>
<td>double precision floating-point</td>
</tr>
</tbody>
</table>
Rules for Writing Valid Arithmetic Expressions

- A basic element is one of the following:
  - An integer or a floating-point constant. For example 24, -2.57, and 17.3.
  - A single variable. For example num1 and fnum.
  - An arithmetic expression enclosed in parentheses
    For example: (num + 5), (fnum * 3), and ((fnum1 - fnum2) * (num / 6)).

- An arithmetic expression is one of the following:
  - A basic element
  - A basic element that is preceded by the minus sign (-). For example: -num, -2 and -(num/5).
  - Two or more basic elements separated by single arithmetic operators.
  
  **Example:**
  - num1 + 2 - num2
  - num1 + num2 * 5 - 6
  - (num1 + 3) * (num2 + (num1 % 5) - 24) + 10.

Rules for Evaluating Arithmetic Expressions

1. First evaluate all sub-expressions in parentheses, starting with the inner most parentheses.
2. Then, evaluate unary operations from right to left.
3. Then, evaluate multiplication, division and modulus operations from left to right.
4. And finally, evaluate addition and subtraction operations from left to right.

Evaluation of Arithmetic Expressions

**a.**  
3 + 11 * 2 – 5  

<table>
<thead>
<tr>
<th>Step</th>
<th>Expression</th>
<th>New Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>step 1</td>
<td>11 * 2 = 22</td>
<td>3 + 22 - 5</td>
</tr>
<tr>
<td>step 2</td>
<td>3 + 22 = 25</td>
<td>25 - 5</td>
</tr>
<tr>
<td>step 3</td>
<td>25 - 5 = 20</td>
<td></td>
</tr>
</tbody>
</table>

**b.**  
81 - 4 * 13 / 2 * 3  

<table>
<thead>
<tr>
<th>Step</th>
<th>Expression</th>
<th>New Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>step 1</td>
<td>4 * 13 = 52</td>
<td>81 – 52 / 2 * 3</td>
</tr>
<tr>
<td>step 2</td>
<td>52 / 2 = 26</td>
<td>81 – 26 * 3</td>
</tr>
<tr>
<td>step 3</td>
<td>26 * 3 = 78</td>
<td>81 - 78</td>
</tr>
<tr>
<td>step 4</td>
<td>81 - 78 = 3</td>
<td></td>
</tr>
</tbody>
</table>
c.  \[ 25.3 + 13 / 2 - 4 \]
New Expression

Step 1:  
\[ 13 / 2 = 6 \]
Step 2:  
\[ 25.3 + 6 = 31.3 \]
Step 3:  
\[ 31.3 - 4 = 27.3 \]

\[ 34 - ((21 + 6) / 3 - (37 \% 5 + 4) + 7) \]
New Expression

Step 1:  
\[ 21 + 6 = 27 \]
Step 2:  
\[ 37 \% 5 = 2 \]
Step 3:  
\[ 2 + 4 = 6 \]
Step 4:  
\[ 27 / 3 = 9 \]
Step 5:  
\[ 9 - 6 = 3 \]
Step 6:  
\[ 3 + 7 = 10 \]
Step 7:  
\[ 34 - 10 = 24 \]

e.  \[ 25 + 13.0 / 2 - 4 \]
New Expression

Step 1:  
\[ 13.0 /2 = 6.5 \]
Step 2:  
\[ 25 + 6.5 = 31.5 \]
Step 3:  
\[ 31.5 - 4 = 27.5 \]

Operations on Character Values

- Assuming that variables `ch1`, `ch2`, `ch3`, and `num` are defined as follows:

  ```
  char ch1 = 'P', ch2, ch3;
  int num;
  ```

- The execution of the following statements will have the specified effects:
  a)  \( ch2 = ch1 + 2; \)  // the new value in Ch2 is the character ‘R’
  b)  \( ch3 = ch1 - 3; \)  // the new value in ch3 is the character ‘M’
  c)  \( num = 'E' - 'A'; \)  // the new value in num is the integer value 4

Quiz #1: Do exercise 2.

Arithmetic Expressions and the Assignment Statement

- The assignment statement:

  ```
  <variable-name> = <arithmetic-expression>;
  ```

  says to evaluate the arithmetic expression, <arithmetic-expression>, and to store its result in the memory location that corresponds to the variable, <variable-name>.
Example:

With the following declaration of variables `num1` and `num2`:
```
int num1 = 3, num2;
```
the assignment statement: `num2 = num1 * 4;`
will store the value 12 into the memory location that corresponds to the variable `num2`.

➢ To trace the execution of one or more statements is to show how the execution of these statements affects the contents of memory locations.

Example:

<table>
<thead>
<tr>
<th>Statements</th>
<th>Memory Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>num1 = 5, num2, num3;</td>
<td>num1</td>
</tr>
<tr>
<td>num2 = - num1;</td>
<td>5</td>
</tr>
<tr>
<td>num3 = num1 * 6;</td>
<td>5</td>
</tr>
<tr>
<td>num1 = num3 + num2;</td>
<td>25</td>
</tr>
<tr>
<td>num3 = num3 - 12;</td>
<td>25</td>
</tr>
<tr>
<td>num2 = num1 - num3;</td>
<td>25</td>
</tr>
<tr>
<td>num2 = num2 + 1;</td>
<td>25</td>
</tr>
</tbody>
</table>

Quiz #1:  Do exercise 3.

Stream Output with cout Object

➢ Output to the monitor may be performed by using the output stream object `cout`.
➢ Its declaration statement is provided in the header file `iostream`.
➢ You must include this file in any source file in which you perform output using the stream object `cout`.

A. Using cout Statements to Output String Constants

a) `cout << "Hello World!";`

   Output `Hello World!`

b) `cout << "Enter a number:\n";`

   Output `Enter a number: _`

c) `cout << "John's tape\n";`

   Output `|
   | John's tape
   | `_
d) cout << "\nI love" << "candies";

Output | I lovecandies

e) cout << "\nI love";
cout << "candies";

Output | I lovecandies

f) cout << "\nI love" << "uncandies";

Output | I love
| candies

g) cout << "\nDid you have fun\nBye bye";

Output | Did you have fun?
| Bye bye

B. Using cout Statements to Output the Results of Arithmetic Expressions

Assume that the variables are defined and initialized as follows:

```cpp
int inum = 125;
char ch = 'Z';
float fnum = 42.75;
double dnum = 347.874;
```

a) cout << "\nThe value in inum is:\t" << inum;

Output | The value in inum is: 125

b) cout << "\nI have chosen the value:\t" << 47 << " and the value of ch is:\t" << ch;

Output | I have chosen the value:47 and the value of ch is: Z

c) cout << "\nThe value of fnum is:\t" << fnum << "\n and that of dnum is:\t" << dnum;

Output | The value of fnum is: 42.75000
| and that of dnum is: 347.8740

d) cout << "\n12 + 23 =\t" << (12 + 23);

Output | 12 + 23 = 35

e) cout << "\ninum - 25 =\t" << (inum - 25);

Output | inum - 25 = 100

f) cout << inum << ch << " " << fnum;

Output | 125Z 42.750000

➢ When the name of a variable appears in a string constant, it is not replaced by its value in the output stream.

➢ Arithmetic expressions are replaced with their values in the output stream.

➢ Floating-point values are printed with the maximum number of digits after the decimal point.
In order to prevent operator precedence conflicts between the operators in an arithmetic expression and the stream insertion operator <<, arithmetic expressions are usually enclosed in parentheses.

**Formatted Output**

- The **manipulator setw(n)** can be used to set the (minimum) field width of the next output value to \( n \), and the **manipulator setprecision(n)** can be used to set the number of digits to be displayed after the decimal point for all floating-point values to be output (in the fixed-point format) to \( n \).

  - In order to use either the manipulator setw() or setprecision(), you must first include the header file `<iomanip>` in your program:

    ```cpp
    #include <iomanip>
    ```

  - After the appearance of the manipulator setprecision(n) in a program, all floating-point values are output with \( n \) digits after the decimal point until another setprecision( ) with a different value appears in the program.

  - setw(n) causes the next output value to be printed with at least \( n \) characters with additional spaces added to the left of the value.

**Using setw(n) and setprecision(n) Manipulators**

In the following output, the character \( b \) represents a single space.

1. ```cpp
cout << endl << “Start” << setw(4) << 10 << setw(4) << 20 << setw(6) << 30;
```
```cpp
Output
Start b / b / 10 b / 20 b / / 30
```

2. ```cpp
cout << setw(10) << “n LIST OF PRICES\n” << setw(15) << “ITEM” << setw(15) << “PRICE”;
```
```cpp
Output
LIST OF PRICES
ITEM / / PRICE
```

3. ```cpp
cout << setprecision(2) << 10.3 << endl << 20.16545;
```
```cpp
Output
10.30
20.17
```

4. ```cpp
cout << setprecision( 2 );
cout << 10.341;
cout << 20.16545;
```
```cpp
Output
10.34
20.17
```

- Note that unlike the setw( ) manipulator, the effect of a setprecision( ) manipulator
remains until it is changed to some other value by another setprecision( ) manipulator.

The fixed Manipulator

- Floating-point values are printed either in fixed-point notation (or decimal) or in scientific notation.
- The stream manipulator fixed forces cout to print floating-point values in fixed-point notation.
- When the fixed manipulator is used in a cout statement, all floating values that are subsequently printed will be displayed in fixed-point notation.

Examples:

1. cout << setprecision(2) << fixed << 10.341 << endl << 20.16545;
   Output  |10.34
           |20.17

2. cout << setprecision(2) << fixed;
   cout << 10.341;
   cout << 20.16545;
   Output  |10.34
           |20.17

The showpoint Manipulator

- By default, floating-point values are not printed with trailing zeroes, and floating-point values that do not have a fractional part are not displayed with a decimal point.

Example:

    cout << setprecision(2) << fixed << 10.3 << endl << 25.0;
    Output  |10.3
             |25
When the `showpoint` manipulator is used in a `cout` statement, all floating values that are subsequently printed will be displayed with padded zeroes (if necessary).

**Examples:**

```
cout << setprecision(3) << fixed << showpoint << 10.3 << endl << 25.0;
```

**Output**

```
|10.300
|25.000
```

**The left and right Manipulators**

- When the `setw()` manipulator is used in a `cout` statement to specify the number of spaces to use for an output value, that output value is right justified (aligned to the right): that means that any extra spaces are added to the left of the output value.

- When the `left` manipulator is used in a `cout` statement, all values that are subsequently printed will be left justified (aligned to the left): any extra spaces will be added to the right of the output value.

- The `right` manipulator causes subsequent output to be right justified (aligned to the right) as in the default case.

**Examples:**

1. ```
cout << setprecision(2) << fixed << showpoint;
cout << endl << “Start” << setw(4) << 10
<< endl << setw(4) << 20 << endl << setw(6) << 30
<< endl << setw(7) << 10.3 << endl << setw(8) << 25.0;
```

**Output**

```
|Start|
|10|
|20|
|30|
|10.3|
|25.0|
```
2.

```cpp
cout << setprecision(2) << fixed << showpoint << left;
cout << endl << “Start” << setw(4) << 10
 << endl << setw(4) << 20 << endl << setw(6) << 30
 << endl << setw(7) << 10.3 << endl << setw(8) << 25.0;
```

**Output**

```
Start 10
20
30
10.3
25.0
```

Quiz #1: Do exercises 4, 5, and 6.
Stream Input with cin Object

- In the C++ programming language, input from the keyboard may be performed by using the input stream object cin.
- Its declaration statement is provided in the header file iostream.
- You must include this file in any source file in which you perform input using the object cin.
- Its syntax is:

  ```
  cin >> <variable-name1> >> <variable-name2> >> ...;
  ```

Examples:

In the following examples, it is assumed that the variables are defined as follows:

```
char ch1 , ch2;
int inum1, inum2;
float fnum;
double dnum1, dnum2;
```

For each cin statement, we show the status of the input stream before a value is extracted from it, and the contents of the memory location(s) where the input value(s) extracted from the input stream are stored.

a) cin >> ch1;

```
Input | K
Memory | ch1 K
```

b) cin >> num1;

```
Input | 123
Memory | inum1 123
```

c) cin >> fnum1;

```
Input | 123.45
Memory | fnum 123.45 00
```

d) cin >> dnum1;

```
Input | 123.45
Memory | dnum1 123.450000000000
```

e) cin >> inum1 >> inum2 >> dnum1;

```
Input | 123 123 6.75
Memory | inum1 123
        | inum2 45
        | dnum1 6.750000000000000
```


\[ f) \quad \text{cin} \gg \text{inum1} \gg \text{ch1} \gg \text{fnum}; \]
\[ \text{cin} \gg \text{inum2}; \]

\begin{array}{|c|c|}
\hline
\text{Input Stream After the Input of all Values:} & \text{B78} \\
\hline
\end{array}

Quiz #1: Do exercises 7 and 8.

**Compound Assignments and the Assignment Operator**

- A **compound assignment** is a shorthand notation of an assignment statement in which the name of a variable appears in both sides of the assignment operator as in the following examples:

\[
\text{num} = \text{num} + 5; \quad \text{num} = \text{num} - 5; \quad \text{num} = \text{num} * 5; \quad \text{num} = \text{num} / 5; \quad \text{num} = \text{num} \% 5;
\]

- It is specified as follows: \(<\text{variable-name}> <\text{operator}> = <\text{expression}>;\)

- Which has the same effect as the regular assignment:

\[
<\text{variable-name}> = <\text{variable-name}> <\text{operator}> <\text{expression}>;
\]

**Examples:**

<table>
<thead>
<tr>
<th>Compound Assignments</th>
<th>Equivalent Simple Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter += 5;</td>
<td>counter = counter + 5;</td>
</tr>
<tr>
<td>power *= value;</td>
<td>power = power * value;</td>
</tr>
<tr>
<td>average /= count;</td>
<td>average = average / count;</td>
</tr>
<tr>
<td>total -= value;</td>
<td>total = total - value;</td>
</tr>
<tr>
<td>remain %= 8;</td>
<td>remain = remain % 8;</td>
</tr>
<tr>
<td>result /= num - 25</td>
<td>result = result / (num - 25);</td>
</tr>
<tr>
<td>number *= -1;</td>
<td>number = -number;</td>
</tr>
</tbody>
</table>

- In C++, the assignment is an operator with right-to-left associativity. For example the following statement has the effect of storing the integer value 5 to the variable \(\text{number}\), then \(\text{num2}\), then \(\text{num1}\):

\[
\text{num1} = \text{num2} = \text{number} = 5;
\]

Quiz #1: Do exercise 9.
Increment and Decrement Operators

Increment Operator

- The **increment operator** specifies the following two actions:
  1. Add 1 to the current value of the variable.
  2. Return the current value of the variable.

- The order in which these two actions are performed depends on how the operator is specified.

- The **pre-increment** operator which is specified as follows:

  \[ ++\text{<variable-name>} \]

  Says to perform the first action before the second.

- The **post-increment** operator which is specified as follows:

  \[ \text{<variable-name>}++ \]

  Says to perform the second action before the first.

Decrement Operator

- The **decrement operator** specifies the following two actions:

  1. Subtract 1 from the current value of the variable.
  2. Return the current value of the variable.

- Similarly to the increment operator, the order in which these two actions are performed depends on how the operator is specified.

- The **pre-decrement** operator which is specified as follows:

  \[ --\text{<variable-name>} \]

  Says to perform the first action before the second.

- The **post-decrement** operator which is specified as follows:

  \[ \text{<variable-name>}-- \]

  Says to perform the second action before the first.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a)  int num = 5;</td>
<td></td>
</tr>
<tr>
<td>num++; // the value of variable num is not used</td>
<td></td>
</tr>
<tr>
<td>cout &lt;&lt; endl &lt;&lt; &quot;num =&quot; &lt;&lt; num;</td>
<td>num = 6</td>
</tr>
<tr>
<td>b)  int num = 5;</td>
<td></td>
</tr>
<tr>
<td>++num; // the value of variable num is not used</td>
<td></td>
</tr>
<tr>
<td>cout &lt;&lt; endl &lt;&lt; &quot;num =&quot; &lt;&lt; num;</td>
<td>num = 6</td>
</tr>
</tbody>
</table>
c) int num1 = 5, num2 = 5, result1, result2;
   result1 = ++num1 - 3;
   // add 1 to the value of num1, then use it
   result2 = num2++ - 3;
   // use the value of num2, then add 1 to it
   cout << "num1=\t" << num1 << "\tresult1=\t" << result1
   << "\tnum2=\t" << num2 << "\tresult2=\t" << result2;

   num1 = 6
   result1 = 3
   num2 = 6
   result2 = 2

---

d) int num = 5;
   num - - ;       // the value of variable num is not used
   cout << endl << "num = " << num;
   num = 4

---

e) int num = 5;
   - - num;       // the value of variable num is not used
   cout << endl << "num = " << num;
   num = 4

---

f) int num1 = 5, num2 = 5, result1, result2;
   result1 = - - num1 + 3;
   // subtract 1 from the value of num1, then use it
   result2 = num2 - - + 3;
   // use the value of num2, then subtract 1 from it
   cout << "\tnum1=\t" << num1 << "\tresult1=\t" << result1
   << "\tnum2=\t" << num2 << "\tresult2=\t" << result2;

   num1 = 4  result1 = 7
   num2 = 4  result2 = 8

Quiz #1: Do exercises 10 and 11.

**Comments**

- Comments are used in a program in one of the following ways:
  a. To specify the object of the program or a function.
  b. To introduce a step of the computation
  c. To specify the purpose of a variable.

Examples of these different usages of comments are provided in the program that follows.
C++ Program

- A C++ program consists of one or more files called source modules (or source files).
- A C++ source module consists of one or more functions.
- Every C++ program must have a unique function called main.
- The execution of a program starts with the first executable statement in function main.
- Statements in any other functions are executed only if that function is called or invoked by function main or a function that is called by function main.
- The statements in the following source module are executed in the following order: 13, 14, 15, 16, 3, 4, 5, 6, 7, 8, 9, 10, 17, 18, 19.

---

An Example of Function Definitions and Function Call

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

/* function to read a character and an integer value, to compute the double of the value, and to print it with the character read. */
void test()
{
    Char letter;  // to hold the character
    int num,      // to hold the integer value
                result;  // to hold the double of the value
    /*---------- read a character and an integer value -------*/
    cout << endl << "Enter a character please: ";
    cin >> letter;
    cout << endl << "Enter an integer value: ";
    cin >> num;

    /*compute the double of the value and print the results */
    result = 2 * num;
    cout << endl << "The character is: " << letter;
    cout << endl << "The double of the value is: " << result;
    cout << endl << "Bye bye";
}

/*---------------- function main ----------------------*/
int main()
{
    int num1;    // the first number
    int num2;    // the second number
    /*----------read two integer values ----------*/
    cout << endl << "Enter two integer values: ";
    cin >> num1 >> num2;
```
/*read a character and an integer value and compute the double of the value*/
15  cout  <<  endl  <<  "I am about to enter function test( )";
16  test();
17  cout  <<  endl  <<  "I am back in function main";

/*------ compute the sum of the two values and print it */
18  cout  <<  endl  <<  "The sum of the two values is:  
19     (num1 + num2);
20  return (0);
 }

Functions That Return a Value

The output of the following program is:

result1 = 5
result2 = 15

#include   <iostream.h>
using namespace std;

int tester2( )
{
  return (5);
}

int main( )
{
  int num;
  int result1;
  int   result2;
  num = 10;
  result1 = tester2( );
  result2 = num + tester2( );
  cout  <<  endl  <<  "result1 ="  <<  result1
19     <<  endl  <<  "result2 ="  <<  result2;
  return ( 0 );
}

Quiz #1:  Do exercises 12 and 13.

Library Functions and the Preprocessor

- A C/C++ compiler also comes with a collection of functions that are already translated into machine language. These functions are referred to as library functions and may be called in any program.

- Library functions are held in library files and their declaration statements are held in header files that come with the compiler software package.

  Examples of header files are iostream and iomanip.
In order to call a library function in a source module, you must first insert the content of a header file that contains the declaration of that library function into the source module.

You can insert the content of a header file into a program source module by using the preprocessor directive, `#include`.

The preprocessor is a program that comes with the compiler software package. It first processes and completes a program source file before it is translated into machine language by the compiler.

You instruct the preprocessor to copy the contents of a file into a program source file by using the `#include` directive as follows:

```
#include <filename>                       // for header files that come with the compiler
or    #include "file-specified"        // for the user defined header files
```

Example: `#include <iostream>`

The following arithmetic and random number generator library functions are available in all C++ implementations.

Library functions `abs( )`, `rand( )` and `srand( )` are declared in the header file `cstdlib`; and the others are declared in the header file `cmath`.

<table>
<thead>
<tr>
<th>Function Declaration</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>double abs (double d );</td>
<td>returns the absolute value of d</td>
<td>cout &lt;&lt; abs( 25);</td>
</tr>
<tr>
<td>double sqrt (double d);</td>
<td>returns the square root of d</td>
<td>cout &lt;&lt; sqrt (num + 5);</td>
</tr>
<tr>
<td>double pow(double d , int i);</td>
<td>returns d to the power of i</td>
<td>cout &lt;&lt; pow (5.25, 7);</td>
</tr>
<tr>
<td>double ceil (double d);</td>
<td>returns the smallest integer value not less than d</td>
<td>cout &lt;&lt; ceil(17.24);</td>
</tr>
<tr>
<td>double floor (double d);</td>
<td>returns the largest integer value not greater than d</td>
<td>cout &lt;&lt; floor(24.21);</td>
</tr>
<tr>
<td>double exp (double d);</td>
<td>returns the exponent (base e) of d</td>
<td>cout &lt;&lt; exp(24.2);</td>
</tr>
<tr>
<td>double log (double d);</td>
<td>returns the (base e) logarithm of d</td>
<td>cout &lt;&lt; log(5.25);</td>
</tr>
<tr>
<td>double log10 (double d);</td>
<td>returns the (base 10) logarithm of d</td>
<td>cout &lt;&lt; log10(35.4);</td>
</tr>
<tr>
<td>int rand( );</td>
<td>returns a pseudo random number between 0 and RAND_MAX (predefined integer constant)</td>
<td>double num; num = rand( ); cout &lt;&lt; num / RAND_MAX;</td>
</tr>
<tr>
<td>void srand(int s);</td>
<td>reinitializes the random number generator with seed s (positive value)</td>
<td>srand (10);</td>
</tr>
</tbody>
</table>
**Calling a Library Function**

The following output are produced by the following program for the input values 5, and -8, respectively:

**A.** Enter an integer value please:  5
   Its absolute value is:  5

**B.** Enter an integer value please  -8
   Its absolute value is:  8

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

int main()
{
    int num;  /* to hold an integer value */
    int result;  /* to hold its absolute value */
    cout  <<  endl  <<
        "Enter an integer value please:  
        " << num;
    cin     >>     num;
    result = abs( num );
    cout  <<  endl  <<
        "Its absolute value is:   
        " << result;
    return (0);
}
```

Quiz #1:  Do exercises 14 and 15.

**Processing the include Directive**

Suppose that the content of the file *myheader.h* is as follows:

```
/*-------------------------- myheader.h--------------------------------------------*/
int   gnum = 10;
int   tester ( );  // a new function
char letter = ‘A’;
```

And that the content of the file *myprog.cpp* is as follows:

```
/*--------------------------myprog.cpp----------------------------------------*/
#include “myheader.h”
int main( )
{
    int num = 15;
    int result;
    result = num + tester( );
    cout  <<  endl  <<
        “The result is: ”  
        << result;
    return ( 0 );
}
```
The content of the file myprog.cpp after it has been processed by the preprocessor is as follows:

```cpp
/*-----------------------------------------------*/
/*------------------------*/
int gnum = 10;
int tester ( ); // a new function
char letter = 'A';
int main() {
    int num = 15;
    int result;
    result = num + tester ( );
    cout    <<    endl    <<
            "The result is: " << result;
    return ( 0 );
}
```

### Naming Constants

#### Using `#define` Directive and Symbolic Constants

- The `#define` directive is used to associate a name to a constant as follows:

  ```cpp
  #define   <name>   <constant>
  ```

- After this association is done, `<name>` can be used in the program in any place where `<constant>` will normally be used.

- Before a program is compiled, all occurrences of `<name>` in the program are replaced with `<constant>` by the preprocessor. The following program illustrates the use of the `#define` directive.

#### Line Number

1  /**********************************************************************************
2  Program to read the radius of a circle and to compute its area and perimeter.
3  **********************************************************************************/
4  #include  <iostream>
5  #include  <iomanip>
6  using namespace std;
7  #define  PI   3.14
8
9  int main()
10 {  
11     Cout.setf(ios :: fixed);
12     Cout.setf(ios :: showpoint);
13     double radius; // to hold the radius of a circle
14     /***************read the radius of the circle**************/
15     cout    <<    "\nEnter the radius of the circle:\t";
16     cin    >>    radius;
17     /***********compute and print the area of the circle***********/
18     Cout    <<    setprecision(2);
19     cout    <<    "\n\nThe area of the circle is:\t"
20     <<    radius * radius * PI;
```
24  /*----------compute and print its perimeter-------------*/
25  cout << "\n\nThe perimeter of the circle is: \t"
26  <<  2 * PI * radius;
27  return 0;
28 }

➤ The statements of function main after they have been processed by the preprocessor are shown as follows:

9  int main()
10  {
11      Cout.setf(ios :: fixed);
12      Cout.setf(ios :: showpoint);
13      double radius;  // to hold the radius of a circle
14      /*-----------read the radius of the circle-------------*/
15      cout << "\nEnter the radius of the circle: \t";
16      cin >>  radius;
17      /*----------compute and print the area of the circle--------*/
18      Cout <<  setprecision(2);
19      cout <<  "\n\nThe area of the circle is: \t"
20      << radius * radius * 3.14;
21      /*---------compute and print its perimeter----------------*/
22      cout <<  "\n\nThe perimeter of the circle is: \t"
23      <<  2 * 3.14 * radius;
24      return 0;
25 }

const Variables

➤ A const variable is a variable that is defined and initialized as follows:

    const <data-type> <variable-name> = <constant>;

<constant> is the constant value to be named, and
<data-type> is its data type.

➤ The compiler creates a memory location that corresponds to <variable-name>, and initializes it with that constant value.

➤ The value of a const variable can not be changed in a program.

➤ The following program illustrates the use of const variables.
Line Number
1  /***************************************************************
2  Program to read the radius of a circle and to compute its area
   and  perimeter.
3  ***************************************************************/
4  #include  <iostream>
5  #include  <iomanip>
6  using namespace std;
7
8  int main()
9  {
10     Cout.setf(ios :: fixed);
11     Cout.setf(ios :: showpoint);
12     Const double PI = 3.14;  // the value of pi
13     double radius;  // to hold the radius of a circle
14     /*----------------------
15      read the radius of the circle----------------------*/
16     cout   <<   "\nEnter the radius of the circle:\t";
17     cin   >>   radius;
18
19     /*----------------------compute and print the area of the circle----------------------*/
20     Cout   <<   setprecision(2);
21     cout   <<   "\nThe area of the circle is:\t"  
22     <<   radius * radius * PI;
23
24     /*----------------------compute and print its perimeter----------------------*/
25     cout   <<   "\nThe perimeter of the circle is:\t" 
26     <<   2 * PI * radius;
27     return 0;
28 }

Quiz #1:  Do exercises 16 and 17.

Class string

➤ The C++ programming language does not have a basic data type to store and manipulate character strings. However, the string class may be used to store and process character strings in a way similar to a basic data type.

➤ A class is a data type that is defined by a C++ programmer. In order to use the string class, you must first include the header file string in your program:

    #include   <string>

➤ You declare string variables in the same way you declare variables of any other data types.

    Examples:

    string  firstName,
            lastName,
            name1,
            name2,
            day = “Monday”,
            address;

©2013 Gilbert Ndjatou
You can use an assignment statement to store a character string into a *string* variable.

**Example:**
```
address = "300 Pompton Road, Wayne, NJ 07470";
```

You can use a `cout` statement to output the content of a *string* variable.

**Examples:**
```
cout << endl << "Today is:\t" << day;
cout << "\nMy address is:\t" << address;
```

**OUTPUT**
```
Today is: Monday
My address is: 300 Pompton Road, Wayne, NJ 07470
```

You can use a `cin` statement to read a word (sequence of characters that does not contain a *whitespace*) into a *string* variable.

**Examples:**
```
cin >> name1 >> name2:
```

<table>
<thead>
<tr>
<th><strong>Input Stream after Input:</strong></th>
<th>Doe</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Memory</strong></th>
<th>name1</th>
<th>123</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name2</td>
<td>John</td>
</tr>
</tbody>
</table>

**Note:** A *whitespace* character is a character that you produce when you press the *space bar*, the *Return key*, or the *Tab key*.

You can combine two character strings into one by using the *concatenation operator* `+`.

**Example:**
```
string lastName = "Doe"
firstName = "John"
name;
nname = "Mr. " + firstName + " " + lastName;
cout << name;
```

**OUTPUT**
```
Mr. John Doe
```

Quiz #1: Do exercises 18 and 19.