C++ Statements

- Instructions in C++ are called “statements”
- We will cover these statements in Chapter 2:
  - Output statement using `cout`
  - Variable definition statement
  - Assignment statement
  - Variable initialization statement
  - Named constant declaration (or definition)

2.2 The `cout` Object

- `cout`: short for “console output”
  - a stream object: represents the contents of the screen
- `<<`: the stream insertion operator
  - use it to send data to `cout` (to be output to the screen)
    - `cout << “This is an example.”;`
- when this instruction is executed, the console (screen) looks like this:

  This is an example.

  Note: the “ ” do not show up in the output

The `endl` manipulator

- `endl`: short for “end line”
  - send it to `cout` when you want to start a new line of output.
    - `cout << “Hello ” << endl << “there!”;`
  - or you can use the newline character: `\n`
    - `cout << “Hello \nthere!”;`
- Either way the output to the screen is:

  Hello
  there!
more examples

cout << “Hello ” << “there!”;
Hello there!
cout << “Hello ”;
cout << “there!”;
Hello there!
cout << “The best selling book on Amazon\n is “The Help””; The best selling book on Amazon
is “The Help”

2.4 Variables and Literals

- Variable: named location in main memory
- A variable definition has a name and a datatype
  - <datatype> <identifier>;
  - The data type indicates the kind of data it can contain.
  - The identifier is a name of your choosing.
- A variable must be defined before it can be used!!
- Example variable definitions:
  - int someNumber;
  - char firstLetter;

2.5 Identifiers

- An identifier is a name for some program element (like a variable).
- Rules:
  - May not be a keyword (see Table 2.4 in the book)
  - First character must be a letter or underscore
  - Following characters must be letters, numbers or underscores.
- Identifiers are case-sensitive:
  - myVariable is not the same as MyVariable

Literals

- A literal represents a constant value used in a program statement.
- Numbers: 0, 34, 3.14159, -1.8e12, etc.
- Strings (sequence of keyboard symbols):
  - “Hello”, “This is a string”
  - “100 years”, “100”, “Y”, etc.
- NOTE: These are different: 5 “5”
2.12 Variable Assignments and Initialization

- An **assignment statement** uses the = operator to store a value in an already defined variable.
  - `someNumber = 12;`
- When this statement is executed, the computer stores the value 12 in memory, in the location named “someNumber”.
- The variable receiving the value must be on the left side of the = (the following does NOT work):
  - `12 = someNumber;  //This is an ERROR`

Example program using a variable

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

int main() {
    int number;
    number = 100;
    cout << "The value of the number is " << number << endl;
    return 0;
}
```

Output screen: The value of the number is 100

Variable Initialization

- To initialize a variable means to assign it a value when it is defined:
  - `int length = 12;`
- You can define and initialize multiple variables at once (and change them later):
  ```cpp
  int length = 12, width = 5, area;
  area = 35;
  length = 10;
  area = 40;
  ```

Data Types

- Variables are classified according to their data type.
- The data type determines the kind of information that may be stored in the variable.
- A data type is a set of values.
- Generally two main (types of) data types:
  - Numeric
  - Character-based
C++ Data Types

- int, short, long
  ‣ whole numbers (integers)
- float, double
  ‣ real numbers (with fractional amounts, decimal points)
- bool
  ‣ logical values: true and false
- char
  ‣ a single character (keyboard symbol)
- string
  ‣ any text, a sequence of characters

2.6 Integer Data Types

- Whole numbers such as 12, 7, and -99
- Typical ranges (may vary on different systems):

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

- Example variable definitions:
  short dayOfWeek;
  unsigned long distance;
  int xCoordinate;

2.9 Floating-Point Data Types

- Real numbers such as 12.45, and -3.8
- Typical ranges (may vary on different systems):

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>+/- 3.4e +/- 38 (~7 digits of precision)</td>
</tr>
<tr>
<td>double</td>
<td>+/- 1.7e +/- 308 (~15 digits of precision)</td>
</tr>
<tr>
<td>long double</td>
<td>+/- 1.7e +/- 308 (~15 digits of precision)</td>
</tr>
</tbody>
</table>

- Floating-point literals can be represented in
  - Fixed point (decimal) notation:
    31.4159  0.0000625
  - E (scientific) notation:
    3.14159E1  6.25e-5

Example program using floating-point data types

```cpp
// This program uses floating point data types.
#include <iostream>
using namespace std;

int main() {
    float distance;
    double mass;

    distance = 1.495979E11;
    mass = 1.989E30;
    cout << "The Sun is " << distance << " meters away.\n";
    cout << "The Sun\'s mass is " << mass << " kilograms.\n";
    return 0;
}
```

Output screen:
The Sun is 1.49598e+11 meters away.
The Sun's mass is 1.989e+30 kilograms.
2.10 The bool Data Type

- The values true and false.
- Literal values: true, false
- (false is equivalent to 0, true is equivalent to 1)

```c
int main() {
    bool boolValue;
    boolValue = true;
    cout << boolValue << endl;
    boolValue = false;
    cout << boolValue << endl;
    return 0;
}
```

2.7 The char Data Type

- All the keyboard and printable symbols.
- Literal values: ‘A’ ‘5’ ‘?’ ‘b’
  - characters are indicated using single quotes
- Numeric value of character from the ASCII character set is stored in memory:

```
CODE: char letter;
letter = 'C';
cout << letter << endl;

MEMORY: letter

OUTPUT: C
```

Appendix B shows the ASCII code values

2.8 The C++ string class

- Sequences of characters
- Requires the string header file: #include <string>
- To define string variables in programs:

```
string firstName, lastName;
```

- To assign literals to variables:

```
firstName = "George";
lastName = "Washington";
```

- To display via cout

```
cout << firstName << " " << lastName;
```

```
OUTPUT: George Washington
```
2.13 Scope

- The scope of a variable is the part of the program in which the variable can be accessed.
- A variable cannot be used before it is defined.

```cpp
// This program can't find its variable.
#include <iostream>
using namespace std;

int main() {
    cout << value; // ERROR! value not defined yet!
    int value = 100;
    return 0;
}
```

2.15 Comments

- Used to document parts of the program
- Intended for humans 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

2.16 Named Constants

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

Single and Multi-Line Comments

- Single-Line comments begin with `//` through to the end of line:
  ```cpp
  int length = 12; // length in inches
  int width = 15; // width in inches
  int area; // calculated area
  // calculate rectangle area
  area = length * width;
  ```

- Multi-Line comments begin with `/*`, end with `*/`
  ```cpp
  /* this is a multi-line comment */
  int area; /* calculated area */
  ```
2.17 Programming Style

- The visual organization of the source code
- Includes the use of spaces, tabs, and blank lines
- Includes naming of variables, constants.
- Includes where to use comments.
- Purpose: improve the readability of the source code

Programming Style

Common elements to improve readability:
- Braces \{ \} aligned vertically
- Indentation of statements within a set of braces
- Blank lines between declaration and other statements
- Long statements intentionally broken up over multiple lines.

See the Style Guidelines on the class website.
You must follow these in your programming assignments.