Excerpts from Chapter 2: Introduction to C++

slides from Gaddis, Walters & Muganda (2017). Starting Out with C++ Early Objects 9th Ed.

Topics

2.1 The Parts of a C++ Program

2.2 The `cout` Object

2.3 The `#include` Directive

2.4 Variables and the Assignment Statement

2.5 Literals

2.6 Identifiers

2.7 Integer Data Types

2.8 Floating-Point Data Types

2.9 The `char` Data Type

Topics (continued)

2.10 The C++ **`string`** Class

2.11 The **`bool`** Data Type

2.13 More on Variable Assignments and Initialization

2.15 Arithmetic Operators

2.16 Comments

2.1 The Parts of a C++ Program

Statement	Purpose
`// sample C++ program`	comment
`#include <iostream>`	preprocessor directive
`int main() {`	`main` function prototype; beginning of function
`   std::cout << "Hello, there!";`	output statement
`   return 0;`	send 0 back to the OS signifying successful termination
`}`	

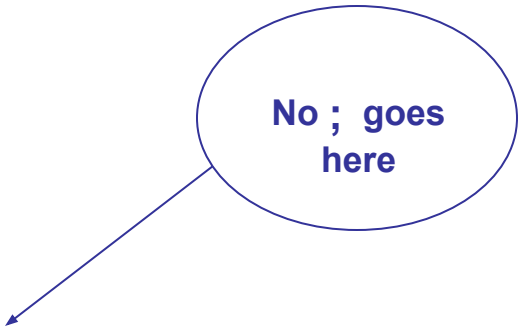
Important Details

- C++ is case-sensitive. Uppercase and lowercase characters are different characters. '`Main`' is not the same as '`main`'.
- Every `{` must have a corresponding `}`, and vice-versa.

2.3 The `#include` Directive

- Inserts the contents of another file into the program
- It is a preprocessor directive
  - Not part of the C++ language
  - Not seen by compiler
- Example:

```
#include <iostream>
```



2.2 The `cout` Object

- Displays information on computer screen
- Use << to send information to cout

```
std::cout << "Hello, there!";
```

- You can use << to send multiple items to cout

```
std::cout << "Hello, " << "there!";
```

Or

```
std::cout << "Hello, ";
std::cout << "there!";
```

Starting a New Line

- To get multiple lines of output on screen
  - Use **`\n`** in an output string (preferred)

```
std::cout << "Hello, there!\n";
```

  - You can also use **`endl`**

```
std::cout << "Hello, there!" << std::endl;
```

Escape Sequences – More Control Over Output

Escape Sequence	Name	Description
`\n`	Newline	Causes the cursor to go to the next line for subsequent printing.
`\t`	Horizontal tab	Causes the cursor to skip over to the next tab stop.
`\a`	Alarm	Causes the computer to beep.
`\b`	Backspace	Causes the cursor to back up, or move left one position.
`\r`	Return	Causes the cursor to go to the beginning of the current line, not the next line.
`\\`	Backslash	Causes a backslash to be printed.
`\'`	Single quote	Causes a single quotation mark to be printed.
`\"`	Double quote	Causes a double quotation mark to be printed.

Common Escape Sequence Mistakes

1) Don't confuse `"\"` (a back slash) and `"/"` (a forward slash)
2) Remember to put `\n` in single quotation marks (character literal) double quotation marks (string literal)

2.4 Variables and the Assignment Statement

A Variable

- Is used to refer to a location in memory where a value can be stored.
- An assignment statement is used to store a value.
- The value that is stored can be changed, *i.e.*, it can "vary".
- You must define the variable (indicate the name and the type of value that it can hold) before you can use it to store a value.

Variables

- If a new value is stored in the variable, it replaces the previous value
- The previous value is overwritten and can no longer be retrieved

```
int age;
age = 17;                 // Assigns 17
std::cout << age;         // Displays 17
age = 18;                 // Now age is 18
std::cout << age;         // Displays 18
```

Assignment Statement

- Uses the = operator
- Has a single variable on the left side and a value on the right side
- Copies the value on the right into the location in memory that is associated with the variable on the left

```
item = 12;
```

2.5 Literals

A Literal is a piece of data that is written directly in the source code of the program.

```
'A'         // character literal
"Hello"     // string literal
12          // integer literal
"12"        //  string literal (yes!)
3.14        // floating-point literal
```

2.6 Identifiers

- Programmer-chosen names to represent parts of the program, such as variables
- Name should indicate the use of the identifier
- Cannot use C++ key words as identifiers
- Must begin with alphabetic character or _, followed by any number of alphabetic, numeric, or _ characters.
- Alphabetic characters may be upper- or lowercase

Multi-word Variable Names

- A variable name should reflect its purpose
- Descriptive variable names may include multiple words
- Two conventions to use in naming variables:
  - **[PREFERRED] Use the underscore _ character as a space (snake case):**

    ```
    quantity_on_order    total_sales
    ```

  - Capitalize all words but the first letter of first word. Run words together (camel case):

    ```
    quantityOnOrder    totalSales
    ```

- Use one convention consistently throughout a program

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
`totalSales`	Yes	
`total_sales`	Yes	
`total.Sales`	No	Cannot contain period
`4thQtrSales`	No	Cannot begin with digit
`total$Sales`	No	Cannot contain $

2.7 Integer Data Types

- Designed to hold whole (non-decimal) numbers
- Can be **`signed`** or **`unsigned`**

```
12      -6      +3
```

- Available in different sizes (*i.e.*, number of bytes): **`short int`**, **`int`**, **`long int`**, and **`long long int`**
- **`long long int`** was introduced in C++ 11.

Signed vs. Unsigned Integers

- C++ allocates one bit for the sign of the number. The rest of the bits are for data.
- If your program will never need negative numbers, you can declare variables to be **`unsigned`**. All bits in unsigned numbers are used for data.
- A variable is signed unless the **`unsigned`** keyword is used at variable definition.

Defining Variables

- Variables of the same type can be defined
  - In separate statements

```
int length;
int width;
```

  - In the same statement

```
int length, width;
```

- Variables of different types must be defined in separate statements

Integer Literals

- To store an integer literal in a long memory location, put '**L**' at the end of the number:
  ```
  long rooms = 234L;
  ```
- Use '**LL**' at the end to put an integer literal in a long long memory location.
- Literals that begin with '**0**' (zero) are octal, or base 8: `075`
- Literals that begin with '**0x**' are hexadecimal, or base 16: `0x75A`

2.8 Floating-Point Data Types

- Designed to hold real numbers

  `12.45` `-3.8`
- Stored in a form similar to scientific notation
- Numbers are all signed
- Available in different sizes (number of bytes): **`float`**, **`double`**, and **`long double`**
- Size of **`float`** ≤ size of **`double`** ≤ size of **`long double`**

Floating-point Literals

- Can be represented in
  - Fixed point (decimal) notation:

    `31.4159  0.0000625`
  - E notation (scientific notation):

    `3.14159E1  6.25e-5`
- Are **`double`** by default
- Can be forced to be float **`3.14159F`** or long double **`0.0000625L`**

Assigning Floating-point Values to Integer Variables

If a floating-point value (a literal or a variable) is assigned to an integer variable

- The fractional part will be truncated (*i.e.*, "chopped off" and discarded)
- The value is not rounded

```
int rainfall = 3.88;
std::cout << rainfall;
// Displays 3
```

2.9 The `char` Data Type

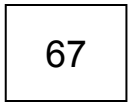
- Used to hold single characters or very small integer values
- Usually occupies 1 byte of memory
- A numeric code representing the character is stored in memory

SOURCE CODE

```
char letter = 'C';
```

MEMORY

letter



Character Literal

- A character literal is a single character
- When referenced in a program, it is enclosed in single quotation marks:

```
std::cout << 'Y' << endl;
```

- The quotation marks are not part of the literal, and are not displayed

String Literals

- Can be stored as a series of characters in consecutive memory locations

```
"Hello"
```

- Is comprised of characters between the `"  "`

A character or a string literal?

- A character literal is a single character, enclosed in single quotes:

```
'C'
```

- A string literal is a sequence of characters enclosed in double quotes:

```
"Hello, there!"
```

- A single character in double quotes is a string literal, not a character literal:

```
"C"
```

2.10 The C++ `string` Class

- Must `#include <string>` to create and use string objects
- Can define `string` variables in programs

```
std::string name;
```

- Can assign values to string variables with the assignment operator

```
name = "George";
```

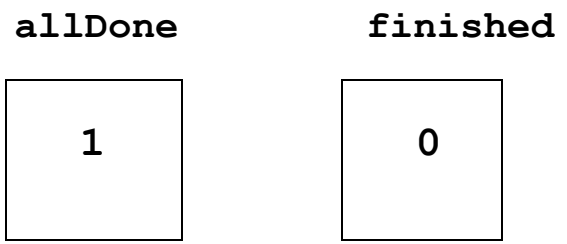
- Can display them with `cout`

```
std::cout << "My name is " << name;
```

2.11 The `bool` Data Type

- Represents values that are **`true`** or **`false`**
- **`bool`** values are stored as integers
- **`false`** is represented by 0, **`true`** by 1

```
bool all_done = true;
bool finished = false;
```



*Any non-zero value is **`true`**

2.13 More on Variable Assignments and Initialization

Assigning a value to a variable

- Assigns a value to a previously created variable
- A single variable name must appear on left side of the = symbol

```
int size;
size = 5+2;   // legal
5 = size;     // not legal
```

2.14 Scope

- The scope of a variable is that part of the program where the variable may be used
- A variable cannot be used before it is defined

```
int num1 = 5;
std::cout << num1;    // legal
std::cout << num2;    // illegal
int num2 = 12;
```

Variable Assignment vs. Initialization

Initializing a variable

- Gives an initial value to a variable at the time it is defined
- Some or all of the variables being defined can be initialized

```
int length = 12;
int width = 7, height = 5,
    area = 3;
```

2.15 Arithmetic Operators

- Used for performing numeric calculations
- C++ has unary, binary, and ternary operators
  - unary (1 operand) `-5`
  - binary (2 operands) `13 - 7`
  - ternary (3 operands) `exp1 ? exp2 : exp3`

Binary Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	ans
+	addition	`ans = 7 + 3;`	10
-	subtraction	`ans = 7 - 3;`	4
*	multiplication	`ans = 7 * 3;`	21
/	division	`ans = 7 / 3;`	2
%	modulus	`ans = 7 % 3;`	1

/ Operator

- C++ division operator (/) performs integer division if both operands are integers

```
std::cout << 13 / 5;    // displays 2
std::cout <<  2 / 4;    // displays 0
```

- If either operand is floating-point, the result is floating-point

```
std::cout << 13 / 5.0;  // displays 2.6
std::cout << 2.0 / 4;   // displays 0.5
```

% Operator

- C++ modulus operator (%) computes the remainder resulting from integer division

```
std::cout << 9 % 2;   // displays ?
```

- % requires integers for both operands

```
std::cout << 9 % 2.0; // displays ?
```

% Operator

- C++ modulus operator (%) computes the remainder resulting from integer division

```
std::cout << 9 % 2;   // displays 1
```

- % requires integers for both operands

```
std::cout << 9 % 2.0; // error
```

2.16 Comments

- Are used to document parts of a program
- Are written 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 `//` and continue to the end of line

```
double side_a = 5;
double side_b = 3;
double hypotenuse;
//
//
hypotenuse = sqrt(side_a * side_a +
                  side_b * side_b);
```

Single-Line Comments

- Begin with `//` and continue to the end of line

```
double side_a = 5;
double side_b = 3;
double hypotenuse;
// Calculate hypotenuse using the
// pythagorean theorem
hypotenuse = sqrt(side_a * side_a +
                  side_b * side_b);
```

Multi-Line Comments

- Begin with `/*` and end with `*/`
- Can span multiple lines

```
/*----------------------------
   Here's a multi-line comment
  ----------------------------*/
```

- Can also be used as single-line comments

```
int area;    /* Single line */
```

Excerpts from Chapter 2: Introduction to C++

slides from Gaddis, Walters & Muganda (2017). Starting Out with C++ Early Objects 9th Ed.