1.x — Chapter 1 summary and quiz

Chapter Review

A **statement** is a type of instruction that causes the program to perform some action. Statements are often terminated by a semicolon.

A **function** is a collection of statements that execute sequentially. Every C++ program must include a special function named *main*. When you run your program, execution starts at the top of the *main* function.

In programming, the name of a function (or object, type, template, etc…) is called its **identifier**.

The rules that govern how elements of the C++ language are constructed is called **syntax**. A **syntax error** occurs when you violate the grammatical rules of the language.

Comments allow the programmer to leave notes in the code. C++ supports two types of comments. Line comments start with a `//` and run to the end of the line. Block comments start with a `/*` and go to the paired `*/` symbol. Don't nest block comments.

You can use comments to temporarily disable lines or sections of code. This is called commenting out your code.

Data is any information that can be moved, processed, or stored by a computer. A single piece of data is called a **value**. Common examples of values include letters (e.g. `a`), numbers (e.g. `5`), and text (e.g. `Hello`).

A variable is a named piece of memory that we can use to store values. In order to create a variable, we use a statement called a **definition statement**. When the program is run, each defined variable is **instantiated**, which means it is assigned a memory address.

A **data type** tells the compiler how to interpret a piece of data into a meaningful value. An **integer** is a number that can be written without a fractional component, such as 4, 27, 0, -2, or -12.

Copy assignment (via operator=) can be used to assign an already created variable a value.

The process of specifying an initial value for an object is called **initialization**, and the syntax used to initialize an object is called an **initializer**.

Simplified, C++ supports 6 basic types of initialization:

Initialization Type	Example	Note
Default initialization	int x;	In most cases, leaves variable with indeterminate value
Copy initialization	int x = 5;	
Direct initialization	int x (5);	
Direct list initialization	int x { 5 };	Narrowing conversions disallowed
Copy list initialization	int x = { 5 };	Narrowing conversions disallowed
Value initialization	int x {};	Usually performs zero initialiation

Direct initialization is sometimes called parenthesis initialization, and list initialization (including value initialization) is sometimes called uniform initialization or brace initialization. You should prefer brace initialization over the other initialization forms, and prefer initialization over assignment.

Although you can define multiple variables in a single statement, it’s better to define and initialize each variable on its own line, in a separate statement.

std::cout and `operator<<` allow us to output the result of an expression to the console.

std::endl outputs a newline character, forcing the console cursor to move to the next line, and flushes any pending output to the console. The `'\n'` character also outputs a newline character, but lets the system decide when to flush the output. Be careful not to use `'/n'` (forward slash).

std::cin and `operator>>` allow us to get a value from the keyboard.

A variable that has not been given a value is called an **uninitialized variable**. Trying to get the value of an uninitialized variable will result in **undefined behavior**, which can manifest in any number of ways.

C++ reserves a set of names called **keywords**. These have special meaning within the language and may not be used as variable names.

A **literal constant** is a fixed value inserted directly into the source code. Examples are 5 and “Hello world!”.

An **operation** is a process involving zero or more input values, called **operands**. The specific operation to be performed is denoted by the provided **operator**. The result of an operation produces an output value.

Unary operators take one operand. **Binary** operators take two operands, often called left and right. **Ternary** operators take three operands. **Nullary** operators take zero operands.

An **expression** is a sequence of literals, variables, operators, and function calls that are evaluated to produce a single output value. The calculation of this output value is called **evaluation**. The value produced is the **result** of the expression.

An **expression statement** is an expression that has been turned into a statement by placing a semicolon at the end of the expression.

When writing programs, add a few lines or a function, compile, resolve any errors, and make sure it works. Don't wait until you've written an entire program before compiling it for the first time!

Focus on getting your code working. Once you are sure you are going to keep some bit of code, then you can spend time removing (or commenting out) temporary/debugging code, adding comments, handling error cases, formatting your code, ensuring best practices are followed, removing redundant logic, etc…

First-draft programs are often messy and imperfect. Most code requires cleanup and refinement to get to great!

Quiz time

Question #1

What is the difference between initialization and assignment?

Show Solution

Question #2

When does undefined behavior occur? What are the consequences of undefined behavior?

Show Solution

Question #3

Write a program that asks the user to enter a number, and then enter a second number. The program should tell the user what the result of adding and subtracting the two numbers is.

The output of the program should match the following (assuming inputs of 6 and 4):

```
Enter an integer: 6
Enter another integer: 4
6 + 4 is 10.
6 - 4 is 2.
```

Hint: To print a period and a newline, use `".\n"`, not `'.\n'`.

Show Solution