1.3 — Introduction to objects and variables

Data and values

In lesson 1.1 -- Statements and the structure of a program, you learned that the majority of instructions in a program are statements, and that functions are groups of statements that execute sequentially. The statements inside the function perform actions that (hopefully) generate whatever result the program was designed to produce.

But how do programs actually produce results? They do so by manipulating (reading, changing, and writing) data. In computing, **data** is any information that can be moved, processed, or stored by a computer.

Key insight

Programs are collections of instructions that manipulate data to produce a desired result.

A program can acquire data to work with in many ways: from a file or database, over a network, from the user providing input on a keyboard, or from the programmer putting data directly into the source code of the program itself. In the “Hello world” program from the aforementioned lesson, the text “Hello world!” was inserted directly into the source code of the program, providing data for the program to use. The program then manipulates this data by sending it to the monitor to be displayed.

Data on a computer is typically stored in a format that is efficient for storage or processing (and is thus not human readable). Thus, when the “Hello World” program is compiled, the text “Hello world!” is converted into a more efficient format for the program to use (binary, which we’ll discuss in a future lesson).

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`).

Random Access Memory

The main memory in a computer is called **Random Access Memory** (often called **RAM** for short). When we run a program, the operating system loads the program into RAM. Any data that is hardcoded into the program itself (e.g. text such as “Hello, world!”) is loaded at this point.

The operating system also reserves some additional RAM for the program to use while it is running. Common uses for this memory are to store values entered by the user, to store data read in from a file or network, or to store values calculated while the program is running (e.g.

the sum of two values) so they can be used again later.

You can think of RAM as a series of numbered boxes that can be used to store data while the program is running.

In some older programming languages (like Applesoft BASIC), you could directly access these boxes (e.g. you could write a statement to “go get the value stored in mailbox number 7532”).

Objects and variables

In C++, direct memory access is discouraged. Instead, we access memory indirectly through an object. An **object** is a region of storage (usually memory) that can store a value, and has other associated properties (that we’ll cover in future lessons). How the compiler and operating system work to assign memory to objects is beyond the scope of this lesson. But the key point here is that rather than say “go get the value stored in mailbox number 7532”, we can say, “go get the value stored by this object”. This means we can focus on using objects to store and retrieve values, and not have to worry about where in memory those objects are actually being placed.

Although objects in C++ can be unnamed (anonymous), more often we name our objects using an identifier. An object with a name is called a **variable**.

Key insight

An object is used to store a value in memory. A variable is an object that has a name (identifier).

Naming our objects let us refer to them again later in the program.

Nomenclature

In general programming, the term *object* typically refers to an unnamed object in memory, a variable, or a function. In C++, the term *object* has a narrower definition that excludes functions.

Variable instantiation

In order to create a variable, we use a special kind of declaration statement called a **definition** (we’ll clarify the difference between a declaration and definition later).

Here’s an example of defining a variable named `x`:

```
int x; // define a variable named x, of type int
```

At compile time, when the compiler sees this statement, it makes a note to itself that we are defining a variable, giving it the name `x`, and that it is of type `int` (more on types in a moment). From that point forward (with some limitations that we'll talk about in a future lesson), whenever the compiler sees the identifier `x`, it will know that we're referencing this variable.

When the program is run (called **runtime**), the variable will be instantiated. **Instantiation** is a fancy word that means the object will be created and assigned a memory address. Variables must be instantiated before they can be used to store values. For the sake of example, let's say that variable *x* is instantiated at memory location 140. Whenever the program uses variable x, it will access the value in memory location 140. An instantiated object is sometimes called an **instance**.

Data types

So far, we've covered that variables are a named region of storage that can store a data value (how exactly data is stored is a topic for a future lesson). A **data type** (more commonly just called a **type**) tells the compiler what type of value (e.g. a number, a letter, text, etc…) the variable will store.

In the above example, our variable `x` was given type `int`, which means variable x will represent an integer value. An **integer** is a number that can be written without a fractional component, such as `4`, `27`, `0`, `-2`, or `-12`. For short, we can say that `x` is an `integer variable`.

In C++, the type of a variable must be known at **compile-time** (when the program is compiled), and that type can not be changed without recompiling the program. This means an integer variable can only hold integer values. If you want to store some other kind of value, you'll need to use a different type.

Integers are just one of many types that C++ supports out of the box. For illustrative purposes, here's another example of defining a variable using data type `double`:

```
double width; // define a variable named width, of type double
```

C++ also allows you to create your own custom types. This is something we'll do a lot of in future lessons, and it's part of what makes C++ powerful.

For these introductory chapters, we'll stick with integer variables because they are conceptually simple, but we'll explore many of the other types C++ has to offer (including `double`) soon.

Defining multiple variables

It is possible to define multiple variables *of the same type* in a single statement by separating the names with a comma. The following 2 snippets of code are effectively the same:

```
int a;
int b;
```

is the same as:

```
int a, b;
```

When defining multiple variables this way, there are two common mistakes that new programmers tend to make (neither serious, since the compiler will catch these and ask you to fix them):

The first mistake is giving each variable a type when defining variables in sequence.

```
int a, int b; // wrong (compiler error)

int a, b; // correct
```

The second mistake is to try to define variables of different types in the same statement, which is not allowed. Variables of different types must be defined in separate statements.

```
int a, double b; // wrong (compiler error)

int a; double b; // correct (but not recommended)

// correct and recommended (easier to read)
int a;
double b;
```

Best practice

Although the language allows you to do so, avoid defining multiple variables of the same type in a single statement. Instead, define each variable in a separate statement on its own line (and then use a single-line comment to document what it is used for).

Summary

In C++, we use objects to access memory. A named object is called a variable. Variables have an identifier, a type, and a value (and some other attributes that aren't relevant here). A variable's type is used to determine how the value in memory should be interpreted.

In the next lesson, we'll look at how to give values to our variables and how to actually use them.

Quiz time

Question #1

What is data?

Show Solution

Question #2

What is a value?

Show Solution

Question #3

What is a variable?

Show Solution

Question #4

What is an identifier?

Show Solution

Question #5

What is a type?

Show Solution

Question #6

What is an integer?

Show Solution