

1.3 — Introduction to objects and variables

ALEX SEPTEMBER 16, 2022

Data

In lesson [1.1 -- Statements and the structure of a program](#) (<https://www.learncpp.com/cpp-tutorial/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).

Objects and variables

All computers have memory, called **RAM** (short for random access memory), that is available for your programs to use. You can think of RAM as a series of numbered mailboxes that can each be used to hold a piece of data while the program is running. A single piece of data, stored in memory somewhere, is called a **value**.

In some older programming languages (like Apple Basic), you could directly access these mailboxes (a statement could say something like go get the value stored in mailbox number 7532).

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 has a value and 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 they’re actually being placed.

Objects can be named or unnamed (anonymous). A named object is called a **variable**, and the name of the object is called an **identifier**. In our programs, most of the objects we create and use will be variables.

Author’s note

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`:

```
1 | 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 also 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 variable.

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`:

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

C++ also allows you to create your own user-defined 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 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:

```
1 | int
   | a;
   | int
   | b;
```

is the same as:

```
1 | 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.

```
1 | 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.

```
1 | 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 variables to access memory. 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](#) (javascript:void(0))

Question #2

What is a value?

[Show Solution](#) (javascript:void(0))

Question #3

What is a variable?

[Show Solution](#) (javascript:void(0))

Question #4

What is an identifier?

[Show Solution](#) (javascript:void(0))

Question #5

What is a type?

[Show Solution](#) (javascript:void(0))

Question #6

What is an integer?

[Show Solution](#) (javascript:void(0))



[Next lesson](#)

1.4 [Variable assignment and initialization](#)



[Back to table of contents](#)





[Previous lesson](#)

1.2 [Comments](#)

Leave a comment...

 Name*

 Email* | 

 Find a mistake? Leave a comment! | 

Notify me about replies:



POST COMMENT

Avatars from <https://gravatar.com/> are connected to your provided email address.

505 COMMENTS

Newest ▼