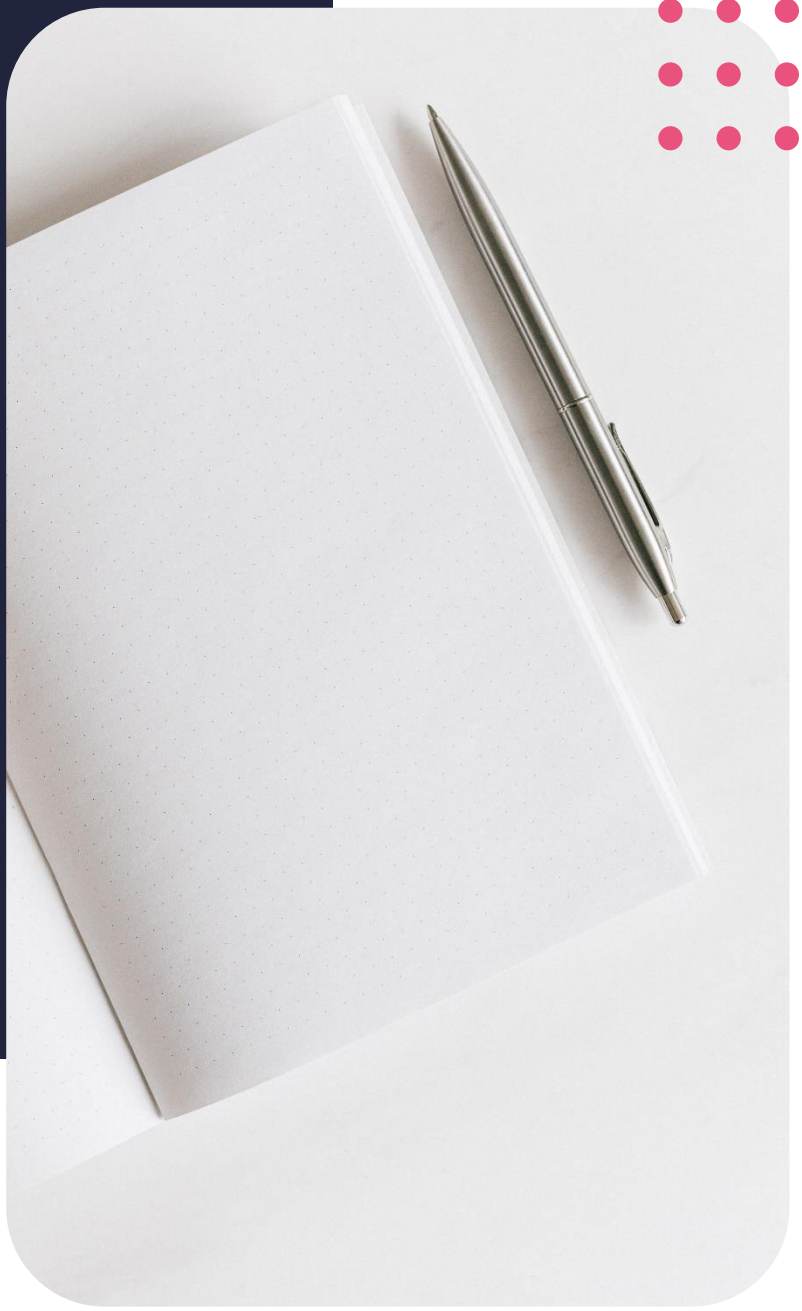
Diploma in Computer Science

Variables and constants

Contents



Lesson outcomes

By the end of this lesson, you should be able to:

- Develop an understanding of variables,
- Appreciate the role of variables in behaviour of your code at runtime
- Discuss the difference between constants and variables
- Identify good programming practices when using variables and constants

In this lesson, we will look at the components of C that allow us to manipulate data in the amazing ways that we see in our daily lives. For you to be able to view this lesson, a lot of variables are holding the data, acting as buffers, receiving data from the internet and reassembling it, and caching the video just to mention a few. We could even go as far as to say the power of programming lies in variables. In our previous lesson, we looked at the various types of data that C supports, and today we are going to see how these come together with variables. Spoiler alert, at the end of this lesson, we will write a program using all that we have learnt in C so far! Let's go!

Variables

Like we just mentioned, variables are at the centre of programming. A more formal definition would be a memory location. It has a name that is associated with that location. The memory location is used to hold data. The content and organisation of a computer's memory is not fixed - so neither is the value that is pointed at by a variable. When data is read from a variable, the content of the memory location is copied and used in calculations.

Variable types

In the previous lesson, we covered the various forms data can take in C. these are also the same forms which variables can take. to understand how these two are related, let's take a look at the general syntax of a variable.

```
data_type variable_name;
```

the area in memory is "formatted" into a specific type to accommodate the specified data type. We will take a closer look at this in a bit. Variables can take 6 forms, namely Types of variables

- local variables
- global variables
- static variables
- automatic variables and
- external variables

we will also look at these in detail in a bit

Variable declaration

When declaring variables, the same rules that we learnt in lesson 3 apply here. The same syntax that we described earlier is used to declare a variable. For the sake of emphasis, we will take another look at it.

```
data_type variable_name;
```

the computer will assign a memory location of a size defined by the data type and label it with the variable name. the computer whenever you use the variable, the computer knows where to store that data and where to retrieve it from.

Typically, you can declare a variable anywhere you want, even right before you use it. you need to keep in mind, however, that the position of your variable affects its visibility to other components of the program, as we shall see shortly.

Naming conventions

When naming your variables, it is essential to make sure that the variable name makes sense. You could, for example, name your variable cfdrths. Of course, it's not a keyword, so the syntactic analyser will not scream at you, and it makes perfect sense to the compiler, and so your program will run fine. Now imagine you find this variable and a dozen other similarly named variables in a block of lengthy code written by someone else. This is where we will start to sing the programming anthem, clarity, clarity, clarity! Such names might end up confusing you, the original author of the program! The best practice would be to name your variables with names that are related to the program`s, function, or procedure. This will ensure that whoever reads the program will be able to follow along and quickly see the flow and logic of the program. For example, if you are writing a program that calculates the ages of people in a population sample based on their date of birth, your variable names will read better if they include the word age.

A few ground rules also apply here. The convention is to separate words in a variable name without the use of white space. White space within variables is usually the source of errors in programming languages in general, not just C. Because of this, variables must be delimited in some other way. Generally, if you want to space out you have three options

- Snakecase: Words are delimited by an underscore, for example,

```
variable_one
variable_two
```

- Pascalcase: Words are delimited by capital letters. For example,

```
VariableOne
VariableTwo
```

- Camelcase: Words are delimited by capital letters, except the initial word.

```
variableOne
variableTwo
```

- Hungarian Notation: This notation describes the variable type or purpose at the start of the variable name, followed by a descriptor that indicates the variable's function. The Camelcase notation is used to delimit words.

```
arrShapeNames  //Array called "Shape Names"
sUserName           //String called "User Name"
iShapeLength        //Integer called "Shape Length"
```

following these naming conventions, you can see how and why a code reviewer will be able to easily snap into the flow of your code. The variable name already tells half of the story, before you even read the rest of the code. Apart form naming variables following these naming conventions you can also add comments to make it even clearer

lvalues and rvalues

The position of your variable is significant to the modification of the variable. The left side of an expression refers to an object that can hold a value. The right side of the expression is where the calculation is performed.

Constants

The key difference when comparing a constant to a variable is that the value associated with a variable name may change during program execution

constants and literals

All along, we have been referring to values that change during the execution of the program, but there's a scenario that we haven't really addressed. What if you want to declare a variable, and you don't want it to change, no matter what? Hmm? You could just declare a variable and not provide a way to modify it in the program. Of course, that could work, but there's an even better way of doing it, that ids, declaring it as a constant. A constant works in exactly the same way as a variable, except that once it is allocated a specific value, it doesn't change no matter what. This is useful for values that you don't need to change, for example, if you are performing calculations with a value such as pi, for example. We all know, from maths, that pi =22/7 or 3.14 and there's no reason why you would want to change it to any other value.

There are a number of reasons why you would want to use constants. Constants are used when you want to assign a value that doesn't change. This comes in handy if, by accident, you try to change the constant's value; you will receive an error.

As a programmer, you would want to eliminate as many possible. It may happen that if the program is large and has many programmers working on it, someone may redefine your variable to make it work for them, but then the next call to your function that depended on the correct value of your variable would then return erroneous results. It is also great for readability of the code. A person who reads your code will now know that this value will never change. In short, by using a constant in your code, you are communicating that

- this value will never change during the runtime of the program.
- This value is probably thread safe, that is, it will only ever be read, not written to.
- This value is likely referenced in many areas throughout the code.
- The name of the value likely tells you something important about what it means.

Most of us know that 3.14 is an approximation of Pi, but many constants are meaningless without a name types of constants. We can declare a constant using #define, a preprocessor directive: We have already discussed #define in detail in earlier lessons. You can also use the keyword const: It is similar to variable declaration except that we need to add the keyword “const” before the constant name. It is important to assign a value to the constant as soon as we declare it.

Constant definition

Constants follow the exact same syntax as variables, they require a data type, a variable name and a value. Constants can be defined in two ways, either using the const keywordor using the #DEFINE (hash define) preprocessor directive. The general syntax looks like this:

#define identifier value

This is used in the definition section of the program. The other method is to use the keyword The general syntax when using the const keyword looks like this:

```
const DataType constantName=value;
```

The same naming conventions that we mentioned when we were talking about variables apply here. It is considered good programming practice to use capital letters when defining constants. If you do this, anyone reading your code can straight away tell apart a constant from a variable. When you declare a constant, the generally accepted practice is to assign a value to it right away. Once you assign a value to it, the value will persist until the program terminates or until the program encounters an instruction to destroy the variable. A constant is like cement. Once it sets, it cannot be altered, except by being destroyed, a variable, on the other hand, is like a bottle of water. You can do all sorts of things, leave it empty, fill it to capacity, or even leave it half full.

Scope

Like we mentioned before, the position of your variable determines where it will be seen in the program and where it will be invisible. This is referred to as the scope of a variable or constant. Take for example, a PABX system at a company. Extension numbers can only be dialled from within the company. The company’s public phone numbers, however, can be dialled from anywhere in the world. You can only “dial” dome variables from within the part of the program where they reside, and you can “dial” some variables from anywhere in the program.

Local variables

Local variables, as the name suggests, are only accessible from within the program space where they are declared if you try to use the variable anywhere else, the compiler will throw an error and refuse to compile. it will be just as though you never declared the variable. going back to our example, if your boss’ internal phone number is 444, if you try to call your boss while you’re at home, your phone will tell you that the number does not exist.

Global variables

Global variables, unlike local variables, are accessible from anywhere in the program. Going back to our example, since you can't call your boss on the office line while at home, you would use your boss' cell phone number instead. It wouldn't matter if you're in the office or in a different country, you would still be able to reach your boss.

Parameters

In previous lessons, we mentioned that we can pass or assign variables to functions so that they can be used in the execution of that function. These are known as parameters or a function. It is common for people to interchange the terms parameter and argument, but if we are to be strict, since we want to get things right the first time, a Parameter refers to a variable in the declaration of function, while an Argument refers to the actual value of this variable that gets passed to function. With that out of the way, let's look at parameters. These come in 2 main forms, namely

- formal parameters, and
- actual parameters.

Before we go further, there's another name that you will see floating around is function prototype. It's actually not a new thing and is just an alternative way of referring to a function declaration.

To sum this up, here's a quick reference:

A parameter is a named variable passed into a function. Parameter variables are used to import arguments into functions.

- Function parameters are the names listed in the function's definition.
- Function arguments are the real values passed to the function.
- Parameters are initialized to the values of the arguments supplied.

A formal parameter is a variable and its type as they appear in the declaration of the function. It is an object or reference declared as part of a function declaration or definition that acquires a value on entry to the function; an identifier from the comma-separated list bounded by the brackets immediately following the macro name in a function-like macro definition. Formal parameters are what you just think of function parameters: for example

```
Int shapes(int length, int width);
```

Length and width here are formal parameters. The point is that in the function body, you're

referring to parameters "formally" without actually knowing their value. It is only when you actually evaluate a function call expression that the formal function parameters are bound to the function call arguments: An actual parameter is an expression in the comma-separated list bounded by the brackets in a function call expression, a sequence of pre-processing tokens in the comma-separated list bounded by the brackets in a function, such as a macro call or an expression. Actual parameters are what you actually (hence the name) pass to the function when you call it.

Let's look at a table that compares arguments and parameters.

ARGUMENT	PARAMETER
The values are passed during the function call	The values are defined during function definition
Are used in the function call statement to send values from the calling function to the called function.	These are used in function header of the called function to store the value from the arguments.
Each argument is always assigned to the parameter in the function definition.	Parameters are local variables which are assigned value of the arguments when the function is called.
also called Actual Parameters	also called Formal Parameters

When a function is called, the arguments in a function can be passed by value or passed by reference.

Pass by value

Pass by value means that a copy of the actual parameter's value is made in memory, that is, the calling function and called function have two independent variables with the same value. If the called function makes changes to the parameter value, the effect is not visible to the caller function.

Here's a summary of the key points

- Caller function passes an argument by value.
- Called function does not have any access to the actual variable in the calling code.
- A copy of the data in the original element is sent to the called function.

Changes made to the passed variable do not affect the actual value of the variable itself.

Pass by reference

Pass by reference, also referred to as pass by address, refers to argument passing using the reference of an argument in the calling function to the corresponding formal parameter of the called function. A copy of the address of the actual parameter is made in memory, that is, the caller function and the called function use the same variable for the parameter. If the called function modifies the parameter variable, the value is copied over to the caller function's variable.

Here's a summary of the key points

- The caller function passes an argument by reference.
- Called function gives a direct reference to the programming element in the calling code.
- The memory address of the stored data is passed.
- Changes to the value have an effect on the original data.

You may be wondering how you would know when to use pass by reference and pass by value. Well, there are quite a few guidelines.

Pass by value

If we are building multi-threaded application, then we don't have to worry of objects getting modified by other threads. In distributed applications pass by value can save the over network overhead to keep the objects in sync.

Pass by reference

In pass by reference, no new copy of the variable is made, so the program uses less time and resources since copying is skipped. This makes programs efficient especially when passing objects of large structs or classes.

To wrap up, these are the main points that we need to take out of all of this:

- A parameter is referred to as a formal parameter or formal argument and an argument is referred to as actual arguments or actual parameters.
- A parameter is optional, and C allows for a default argument to be provided in a function's declaration. This means that while calling the function, the caller can skip arguments.
- A parameter has a name, a data type, and calling method, that is, either call by reference, or call by value, and an argument is an expression. It does not have any name, but it can be a variable, a constant, or a literal.
- The scope of a parameter is the function in which the parameter has been called, that is, it serves as a local variable inside the function.
- The number of arguments in a function call should match with the number of parameters in the function definition. One exception to this rule is functions with variable-length parameter lists.
- The data type of arguments in a function call should match with the data type of parameters in the function definition.

If a parameter is passed by value, modifying it inside the function does not change the corresponding argument in the caller. However, if it is passed by reference, the values of corresponding argument can be changed in the caller.

Initialisation

Whenever you create a variable, it is good practice to initialise it. This ensures that any garbage value that is in that memory location gets overwritten. You might find out that if you use an uninitialized variable, you might get weird results which you will have no idea how the computer arrived there! Memory is typically reused in a computer system, and when memory is reassigned, it is not always cleared, and the previous values will still be in memory. when your new variable is assigned to that portion of memory, it will sit on top of whatever value that is there until you overwrite it. This won't be a problem for simple programs, but if you're writing a large program with many interdependent variables, things can quickly get ugly. It might take you a lot of time to figure out why the program is crashing, and you will go through it a gazillion times, each time seeing that the syntax and semantics are correct, but still, it won.t work properly. Long story short, just initialise your variables!

To initialise a variable, you assign a value to it. If you don't intend to use the variable right away, you could just set the value of the variable to zero. This will ensure that whatever is in the memory location

is overwritten and replaced with a value that you know and can control. This has the overall effect of making your program stable.

Conclusion

In this lesson, we looked at variables as a way of holding the data that programs will be manipulating. We also looked at the best practices when naming variables and accessing them. we had a closer look at the closest cousin to variables, constants. We discovered that while the two are not very different, there are some instances where it makes sense to use constants, as it improves stability and readability of a program. To top it all off, we wrote some code demonstrating all of this. In our next lesson, we will go even deeper as we explore how C reads input from the keyboard and gives the user output. That’s all for today and remember to practise until you’re a pro!

References

Auckland.ac.nz. (2020). Parameters and Arguments. [online] Available at: https://www.cs.auckland.ac.nz/references/unix/digital/AQTLTBTE/DOCU_056.HTM

Cf.ac.uk. (2020). Scope of Variables. [online] Available at: https://users.cs.cf.ac.uk/Dave.Marshall/PERL/node52.html#:~:text=Scope%20refers%20to%20the%20visibility,variable%20has%20a%20global%20scope.

CygnusX1 (2013). What is a formal parameter? [online] Stack Overflow. Available at: https://stackoverflow.com/questions/18870156/what-is-a-formal-parameter

DataFlair. (2019). 5 Types of Constants in C and C++ and How they're Different from Literals - DataFlair. [online] Available at: https://data-flair.training/blogs/constants-in-c-and-c-plus-plus/

dot (2020). Formal and Actual Parameters. [online] Ccsu.edu. Available at: https://chortle.ccsu.edu/java5/Notes/chap34A/ch34A_3.html

Edpresso Team (2019). Pass by value vs. pass by reference. [online] Educative: Interactive Courses for Software Developers. Available at: https://www.educative.io/edpresso/pass-by-value-vs-pass-by-reference

Learn-c.org. (2020). Function arguments by reference - Learn C - Free Interactive C Tutorial. [online] Available at: https://www.learn-c.org/en/Function_arguments_by_reference

Learn-c.org. (2020). Variables and Types - Learn C - Free Interactive C Tutorial. [online] Available at: https://www.learn-c.org/en/Variables_and_Types

MDN Web Docs. (2020). Parameter. [online] Available at: https://developer.mozilla.org/en-US/docs/Glossary/Parameter#:~:text=Note%20the%20difference%20between%20parameters,values%20of%20the%20arguments%20supplied.

Oracle.com. (2018). Code Conventions for the Java Programming Language: 9. Naming Conventions. [online] Available at: https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

OverIQ.com. (2020). Actual and Formal arguments in C. [online] Available at: https://overiq.com/c-programming-101/actual-and-formal-arguments-in-c/#:~:text=Arguments%20which%20are%20mentioned%20in%20the%20definition%20of%20the%20function.local%20variables%20inside%20the%20function.&text=Order%2C%20n

References

Readthedocs.io. (2020). Coding best practices — Research Computing University of Colorado Boulder documentation. [online] Available at: https://curc.readthedocs.io/en/latest/programming/coding-best-practices.html

Techie Delight. (2016). Difference between an argument and a parameter - Techie Delight. [online] Available at: https://www.techiedelight.com/difference-between-argument-parameter/

Tutorialspoint.com. (2020). C - Constants and Literals - Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/cprogramming/c_constants.htm#:~:text=Constants%20refer%20to%20fixed%20values,are%20enumeration%20constants%20as%20well.

Washington.edu. (2020). Function pass by value vs. pass by reference. [online] Available at: https://courses.washington.edu/css342/zander/css332/passby.html#:~:text=By%20definition%2C%20pass%20by%20value,contents%20of%20the%20actual%20parameter.&text=In%20pass%20by%20reference%20(also,the%20actual%20parameter%20is%20stored.

www.javatpoint.com. (2011). Constants in C - javatpoint. [online] Available at: https://www.javatpoint.com/constants-in-c

youth4work.com. (2020). what is lvalue and rvalue in c language? [online] Available at: https://www.youth4work.com/Talent/C-Language/Forum/172615-what-is-lvalue-and-rvalue-in-c-language