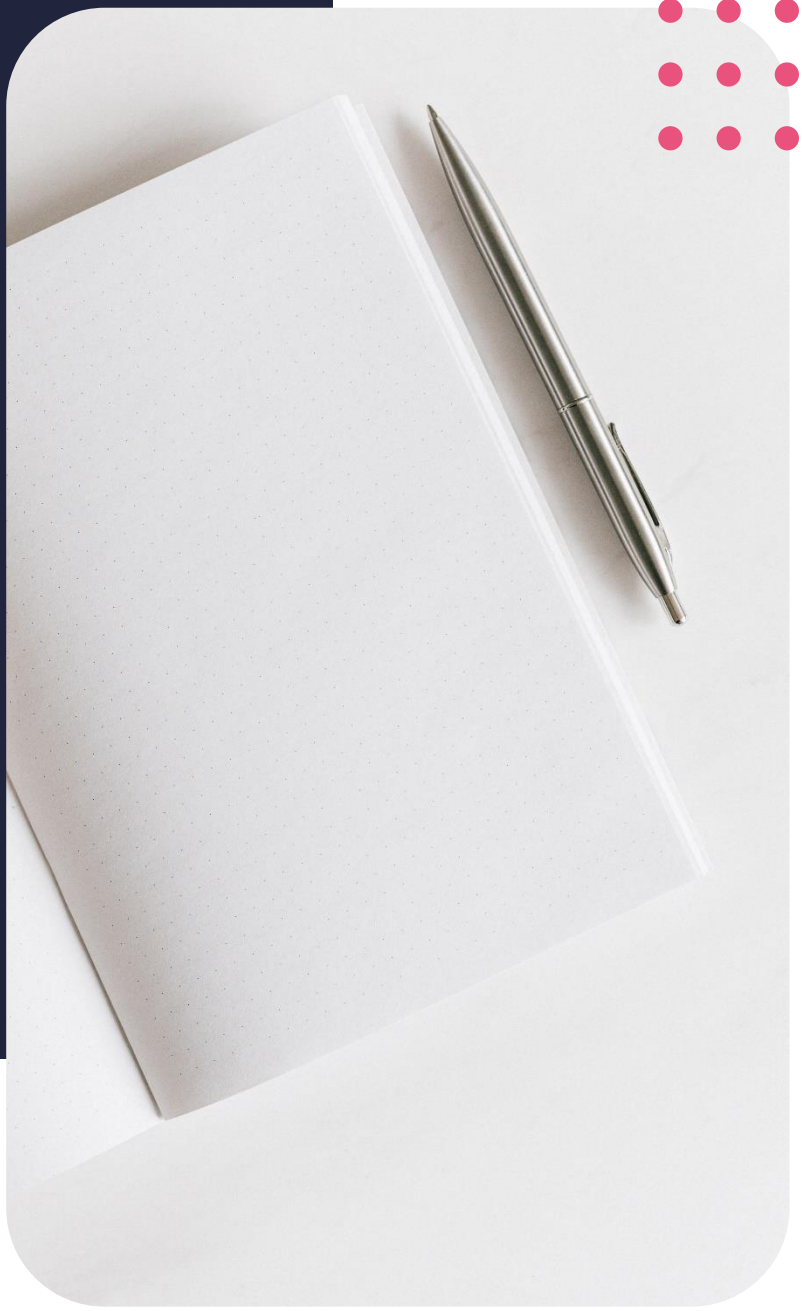
Diploma in Computer Science

C Language

Module 2 Lesson 1

Summary Notes

Contents



Lesson outcomes

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

Understand how C came into being and why it is such an important language

Appreciate the features of C and their uses in programming

Explore the applications of C in the real world

Examine some of the popular libraries included in C

Introduction to C

We will be learning how to code in C. C is a general-purpose language that is extremely popular. You have probably already used a system that was programmed using C! it is used extensively in various applications, as we shall see as we progress with our lesson.

A brief history

Rewind.

The year is 1972 - , a great computer scientist Dennis Ritchie creates a new programming language called C.

It was a mishmash (mixture) of ALGOL, BCPL and B. C contained all the features of these languages plus a bunch of its own features to make it stand out from all other languages of the day.

C was built at Bell Labs as a successor to B. its main purpose was to construct utilities that were run in UNIX. By the mid-70s, C was used in many projects within the Bell system, as it was now a powerful language with many new features. The UNIX kernel was extensively rewritten in C. around this time, Johnson's portable C compiler was born, and It served as the basis for the implementation of C on several new platforms. The publication by Brian Kernighan and Dennis Ritchie in 1978 served as an informal specification of the C language until the ANSI (American national Standards Institute) formed a committee to standardise C and incorporate many of the unofficial features. Around 1990, C was adopted by the ISO (International Standards Organisation). Since then, a lot of other versions have come, such as C99, C11 and C17.

Structure and organisation

A C program is composed of header files, the main function, variables, the body, and a return statement. A C header file contains C function declarations and macro definitions to be shared between several source files. The header file has the file extension .h. These are added to the program by the pre-processor during compilation

Main function

The main function is the core of every program. It contains all the instructions required for the computer to perform the specified task. Whenever the program is run,

The Computer will always look for the main function and get directions from there.

The body

The body of the program contains the actual instructions that the computer needs to carry out. Sometimes the program has references outside the main function, and we shall learn how this works in later lessons

Return Statement

When the program has gone through all steps, control is passed over to the return statement.

The return statement essentially has one job; it terminates the program and returns a value if directed to do so. If a function has the return type “void”, then the return statement can be used without a value. Comments are used in C to provide explanations and references to the way the code operates. C supports inline and multiline comments.

Here we just brushed lightly over these characteristics, and we shall look at them in greater detail as we progress.

Features of C Language

The C language has quite a number of features that distinguish it from other languages.

Speed and efficiency

The most popular feature is the speed and efficiency of C.

Newer programming languages require additional processing, which effectively lowers their performance. C is widely seen as a middle level programming language - providing programmers with access to direct manipulation of computer hardware.

C is a statically typed - programming language, and these are naturally faster than dynamically typed programming languages.

Procedural language

C is a procedural language. These are built with emphasis on the clarity of the code while reducing complexity. In this approach, Code is divided into subroutines. This gives C its characteristically rich control structures.

C is modular

This means that the language code is stored in the form of libraries that can be added to a program as required by the programmer. The C library has a wide range of built-in functions.

C is portable

C is a very portable language. In fact, VERY portable! Portability refers to the ability to run the same code on different platforms. Programs written in C can be compiled and run on any system with little to no changes! There are very few architectures and platforms for which a C compiler does not exist.

General-purpose language

C is a general-purpose language. It is Used in a wide variety of applications ranging from operating systems to databases and music players, among others.

C is flexible

C is generally a very flexible language. Programs written in C can easily be extended with additional features and operations, without significantly altering the existing program. This is more commonly referred to as extensibility. Besides this, c has a lot of other attributes, such as having a rich set of built in operators, which means it can be used to write complex and simple problems alike.

Related languages

By virtue of being the oldest in-use programming language, C has influenced or is related to a number of languages. The natural way and first stop in this talk would be C++. (pronounced C plus plus) This language was developed as an enhancement to C, adding object-oriented programming language features. Another programming language that was developed as an enhancement to C is Objective C it was developed to add Smalltalk like messaging to C. in fact, you could compile some c programs with a C++ compiler.

Java

Another popular programming language that has C roots is Java. Java is directly related to both C and C++; it inherits its syntax from C and the object orientation from C++. It is fairly easy to learn Java if you know how to code in C++, and the same is true for a java programmer learning C or C++. That said, java was built to solve a different set of problems from what C and C++ were built to solve.

Why learn C?

The primary design of C allows you to produce portable code while maintaining performance and minimizing system resource usage. In C, it is easier to keep a mental visualisation of what each line does. It is a stable language which has been around for longer than most of us, and as we mentioned before, it can be ported to pretty much any operating system under the sun. In C, it is easier to master the concepts of programming and apply them to many other programming languages. You will find that C is used in a lot of universities and colleges because it really cements programming language concepts and provides a solid foundation.

C runs the world!

One really good reason to learn C is it runs the world - literally. The bulk of systems that form the backbone of our computing experience today are built using C. This ranges from operating systems to servers and other programming stacks. A lot of popular programming languages are implemented using C because C is a lot closer to the machine than other languages.

Age also has its advantages, Since C has been around for donkey’s years, there’s tons and tons of source code lying around. There’s a lot to learn, a lot of questions and a lot of answers. Because C is

close to hardware, you tend to get a clearer picture of how stuff works, and this is REALLY good for computer science. When you're working as close to the machine as C is, a lot of concepts such as optimisation start to make sense. We have looked at optimisation in previous lessons and spelled out how it is such an important thing in computing. The end goal of computing is to do things efficiently, and efficiency and optimisation go hand in hand. Mastering optimisation will help you to write better code in any other programming language.

C is also great for a lot of applications: if you want to write a fast game, pick C, if you want to write an entire operating system, again, C, if you want to build a framework for your new programming language, again C would be great. These are just a few of C's many applications. With the popularity of IoT, things like Arduino are programmed in C, so is the raspberry pi. Although you can get away with programming on these platforms using other programming languages, you will ultimately get faster code if you program using C.

Properties of C

Paradigm

If you recall from module 1, we categorised programming languages into paradigms, according to what they can and cannot do. The C language is generally considered to be the procedural imperative programming language. This is true because it allows us to use a procedural programming paradigm and it does not have built-in support for other programming paradigms. Because C describes computation in terms of statements that change a program state, it is an imperative programming language. C describes, step by step, the exact procedure that should, according to the programmer, be followed to solve a specific problem.

Also, from module 1, we learnt that procedures are modular and are bound by scope. You will see this from as early as lesson 3. Because of these characteristics, C is generally easier to read and more maintainable. It is also more flexible and facilitates the practice of good program design.

Libraries

C comes with several standard libraries that are defined by the ANSI standard.

The C library POSIX specification was defined at around the same time as a superset of the ANSI standard.

The C standard library provides macros, type definitions and functions for tasks such as string handling, mathematical computations, input/output processing, memory management, and several other operating system services. The library is presented in the form of header files. The more formal and common way to refer to this would be API (application programming interface). You may have heard this abbreviation in other programming circles. It refers to a computing interface that defines the interactions between multiple software intermediaries, that is what kind of calls to make, how to make them, data formats and conventions to follow and so on.

Header files

Provide function declarations and macro definitions that can be shared between several source files. The header file is requested in a program by using a pre-processor directive.

There are two types of header files:

System header files declare the interfaces to parts of the operating system. You include them in your program to supply the definitions and declarations you need to invoke system calls and libraries.

Your own header files contain declarations for interfaces between the source files of your program. Each time you use a pre-processor directive, the header file is copied into the source file that needs it.

The API of the C standard library is defined by the set of header files shown in this table:

<assert.h>	Conditionally compiled macro that compares its argument to zero
<complex.h> (C99)	Complex number arithmetic
<ctype.h>	Functions to determine the type contained in character data
<errno.h>	Macros reporting error conditions
<fenv.h> (C99)	Floating-point environment
<float.h>	Limits of float types
<inttypes.h> (C99)	Format conversion of integer types
<iso646.h> (C95)	Alternative operator spellings
<limits.h>	Sizes of basic types
<locale.h>	Localization utilities
<math.h>	Common mathematics functions
<setjmp.h>	Nonlocal jumps
<signal.h>	Signal handling
<stdalign.h> (C11)	alignas and alignof convenience macros
<stdarg.h>	Variable arguments
<stdatomic.h> (C11)	Atomic types
<stdbool.h> (C99)	Macros for Boolean type
<stddef.h>	Common macro definitions
<stdint.h> (C99)	Fixed-width integer types
<stdio.h>	Input/output
<stdlib.h>	General utilities: memory management, program utilities, string conversions, random numbers
<stdnoreturn.h> (C11)	no return convenience macros
<string.h>	String handling
<tgmath.h> (C99)	Type-generic math (macros wrapping math.h and complex.h)

<threads.h> (C11)	Thread library
<time.h>	Time/date utilities
<uchar.h> (C11)	UTF-16 and UTF-32-character utilities
<wchar.h> (C95)	Extended multibyte and wide character utilities
<wctype.h> (C95)	Functions to determine the type contained in wider character data

The exact standard to which the header file conforms is in brackets if the header file is peculiar to that standard.

Header files

Each header file provides particular functionality and services to the program in which it has been included, For Example, the header file <stdio.h> (pronounced standard I Oh) that we are going to be using in our next lesson, provides the functionality necessary for you to read input from the keyboard and display output on the screen.

Memory management

C provides several functions for memory allocation and management. These functions are found in the <stdlib.h> (pronounced standard library header file) This header file includes functions for allocating initialised and initialised arrays, releasing blocks of memory, and extending allocated memory. The former functions are used when you are aware of the size of your data before you run the program, while the latter are used when the data size will only be known at runtime. In later lessons, we are going to write code that tackles each of these scenarios. It generally works better to only reserve memory when you are using it. Memory is not infinite, so it is good practice to release it so that it can be used for other processes. This is especially true for programs that run on constrained resources, such as embedded system. Filling up such memory is just asking for trouble.

Typically, embedded systems have built in safety and would just reset if that sort of thing happens, but in some instances, all hell will break loose as stacks overflow into each other, corrupting variables and producing unexpected results, and you wouldn't want this sort of thing happening on, say, a nuclear reactor's embedded controller, ANYTHING can happen when a computer runs out of memory! this is why programming languages have built in memory management. Because C does not have automatic memory management (commonly called garbage collection), you need to learn to program in ways that avoid a common programming error called a memory leak.

This occurs when dynamically allocated memory becomes unreachable. We will learn more about this when we look at dynamic data types.

Language tools

Lint and alint provide a way for the developer to check the program for portability problems, syntax errors, wasteful style, and other forms of bugs.

This is a form of a debugger, that is attached to the program during compilation and at runtime, the latter which we looked at in module 1 when we discussed the compilation process.

Applications of C

C is a very old language; this naturally comes with widespread use and deep roots. Anyone who has used a computer has extremely likely encountered a system that was built using, or implemented using C. in fact, chances are, the system you are using right now has several components built using C.

Hardware programming

C is the optimum choice when programming hardware. It is closely related to hardware, is mad fast and has a small system footprint. This is - good news for embedded systems, that have severe power constraints and need to run really small, not very fancy computer that's only there for a few tasks. Take for example, the Arduino Uno has 2KB of RAM and 32KB flash memory, so the footprint of whatever language used here has to be really small. There really isn't any capacity for fancy features here, and the code has to do its job really efficiently. The Arduino, however, does not run C, but a programming language that is similar to C and was just mentioned as an example.

Operating systems

You probably know by now that most operating systems are written in C. this was kind of the way it was going to go anyway, since C was built for use in an operating system. The Linux kernel, for example, is completely written in C. you might remember from module 1 that windows and other operating systems are written in C. this is almost a default choice since C is a fast programming language and is very close to hardware. In module 1, we also learnt that operating systems are responsible for managing hardware resources, so it only makes sense to use a programming language that allows you to access hardware with minimal abstraction.

Embedded systems

Embedded systems are a type of computer systems that are typically not thought of as computers, but as specialised systems that typically rum on time and memory constraints. A common characteristic feature of embedded systems is that none or at least not all functions of embedded software are initiated/controlled via a human interface, but through machine-interfaces instead.

These systems can be found in cars, washing machines, CCTV, routers, set top boxes and monitors to mention a few. A very good example of embedded software is present in the device that you're using to watch this lesson right now using, in BIOS.

Embedded systems are typically required to use hardware resources as efficiently as possible and run as fast as possible. You wouldn't want an airplane to crash because the controller didn't respond in time!

Other notable applications

C has been used in the development of the chromium project, the birthplace of one of the most popular browsers, Google Chrome. It doesn't end there, the google file system and a large number of google projects were developed using C. other browsers such as Mozilla Firefox were also written in C.

One of the most popular uses of C is the development of compilers. The close association of C with low level languages while maintaining the readability of high-level languages makes it a good candidate for development of compilers.

Another application of C is the development of games and graphics. A number of early games were written exclusively in C. a popular game engine,

Examples: Unity, is written in C. again, the speed and close association with hardware makes C the perfect candidate here. Other gaming engines such as Unreal engine, Blender, Buildbox, CryEngine and Cube are also built using C.

Why C?

You may be wondering why we picked C out of all the languages in the world. Surely, after all these years, there are a lot of other languages that are far easier to learn than C. in fact, if you look at the courses offered by Shaw Academy, there are a lot of other programming languages that you might find easier to learn than C.

If you look at the courses offered by most universities in the world, you will see that they still use C to teach programming in Computer Science. There are quite a number of very good reasons why this is so.

C helps you to understand the internal architecture of a computer, that is, how the computer stores and retrieves information, which is precisely what we are trying to achieve in computer science!

After learning C, it will be much easier to learn other programming languages like Java, Python, and so on. Like we mentioned before, C is a middle level language, which makes it sort of bare bones in its approach to solving problems. This helps to really cement concepts and helps you to write better code

Learning C grants you the opportunity to work on open-source projects. Some of the largest open-source projects such as Linux kernel, Python interpreter, SQLite database, and so on are written in C programming. If you are big on Linux, you can even edit your own kernel, and this will help you understand operating system concepts and other operating system theories that are used in operating system development. We can even audaciously say understanding OS-level concepts can only be achieved in C and C alone.

Although computer science is mostly about concepts, which could be learnt in any programming language, there are fields which if explained or learnt using C, they will be better understood than if you learnt them using any other random language of your choice.

Algorithms and Data Structures

In many other languages, these are normally embedded in libraries and other resource files, and there is almost no incentive for you to learn how they work or produce code that is highly optimised for the target system. In C, it’s a completely different story, as most of this functionality has to be built by the programmer, and this forces you to learn the concepts behind the code you are writing. When you

eventually move to other languages, you get a massive head start as you will have mastered these concepts and can confidently fine tune code to suit your needs and expected level of performance.

In many other languages, the code almost certainly feels light, but in reality, it is not. It produces clunky machine code instructions, which are made up for by the speed of today's computers. If you look at a piece of C code, the algorithmic cost of the program is pretty much apparent. Everything in C compiles to a few highly optimised machine instructions. High-level languages. If you start yourself off on very high level, English-like languages, you will tend to be incapable of dealing with performance problems and will often write slow, finicky programs with high algorithmic cost and poor performance. You will also most likely not know how to solve those problems.

You cannot do C without dealing with pointers. Pointers force you to think on two-levels, and that stretches your abstraction skills.

C is as close as you can get to the Von Neumann architecture while staying reasonably expressive. If we really want to be pedantic, we could get away with saying pascal could server the same purpose. Heck, I even learnt pascal in high school. The problem is, other languages take you higher and higher up the hierarchy, which takes you further away from the architecture. Those that don't are really just too old or not powerful enough for the task. That kind of defeats the purpose, as in computer science, I will say it again, our aim is to understand how a computer works and be able to give it the best possible instructions. Knowing how the machine works will make you understand why something that looks fast in terms of algorithmic complexity isn't actually performing as expected. possible instructions. Knowing how the machine works will make you understand why something that looks fast in terms of algorithmic complexity isn't actually performing as expected.

References

What happens when microcontrollers run out of RAM (2015). What happens when microcontrollers run out of RAM? [online] Electrical Engineering Stack Exchange. Available at: https://electronics.stackexchange.com/questions/146298/what-happens-when-microcontrollers-run-out-of-ram

Bailey, T. (n.d.). An Introduction to the C Programming Language and Software Design. [online] Available at: http://www-personal.acfr.usyd.edu.au/tbailey/ctext/ctext.pdf.

DataFlair. (2019). Applications of C Programming That Will Make You Fall In Love With C - DataFlair. [online] Available at: https://data-flair.training/blogs/applications-of-c/

Psu.edu. (2020). Download Limit Exceeded. [online] Available at: http://citeseerx.ist.psu.edu/viewdoc/download

Rochester.edu. (2011). Programming Tools Tutorial Notes. [online] Available at: http://cng.seas.rochester.edu/CNG/docs/ProgTools.html.

Rungta, K. (2020). What is C Programming Language? Basics, Introduction and History. [online] Guru99.com. Available at: https://www.guru99.com/c-programming-language.html

Semantic Designs, Inc (2018). Semantic Designs: C Programming Language Tools. [online] Semdesigns.com. Available at: http://www.semdesigns.com/Products/LanguageTools/CTools.html

Sitesbay.com (2014). Applications of C | Uses of C - C Tutorial. [online] Sitesbay.com. Available at: https://www.sitesbay.com/cprogramming/c-applications

Toptal Engineering Blog. (2015). After All These Years, the World is Still Powered by C Programming. [online] Available at: https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming

Utexas.edu. (2013). Chapter 5: Introduction to C Programming. [online] Available at: http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C5_IntroductionToC.htm

What language are compilers written with (2010). What language are compilers written with? [online] Super User. Available at: https://superuser.com/questions/136136/what-language-are-compilers-written-with