In this course, items printed in Courier new font would appear in a C program exactly as shown; items in *italics* represent text to be supplied by the programmer.

Basic features of C

Introducing C

What is C? The short answer – a widely used programming language developed by Dennis Ritchie, Ken Thompson and others in the early 1970s at Bell Laboratories. The C language is designed to create small, fast programs. It's lower-level than most other languages*; that means it creates code that's a lot closer to what machines really understand.*

Before we get lost in the details of the language, let`s take a look at where C came from, what it was designed for and we`ll also discuss C`s strengths and weaknesses and how to get the most out of the language. This is a practical course of C, but it`s important to know a little bit of history about the language, so don’t skip the next part and try not to fall asleep during this brief history.

**The History of the C Programming Language**

The history of the C programming language is closely tied to the history of the development of the UNIX Operating System.

If we look back to understand what led to the development of the operating system that changed the world of computing, we'll see the steps that led to the development of C.

Simply put, C was derived from the need to initially find and eventually create a language to apply on the UNIX Operating system.

Like other operating systems of that time, UNIX was written in assembly language. Programs written in assembly language are usually painful to debug and hard to enhance; UNIX was no exception. Thompson decided that a higher-level language was needed for further development of UNIX, so he designed a small language named B(remember that was in the mid-1960`s).

Dennis Ritchie soon joined the UNIX project and began programming in B. As they started to write portions of UNIX in B, they realized that it was not well suited for the project in 1971 Ritchie began to develop an extended version of B, and he called the language NB(New B) at first, and then, as it began to diverge more from B, he changed the name to C. In 1972 C was born, and the first C compiler was written and implemented for the first time on the DEC PDP-11 machine.

A picture containing text, person

Description automatically generated

[The famous picture of Thompson and Ritchie working on a PDP-11, Image source Wikipedia](https://en.wikipedia.org/wiki/Ken_Thompson#/media/File:Ken_Thompson_(sitting)_and_Dennis_Ritchie_at_PDP-11_(2876612463).jpg)

By 1973 the language was stable enough that UNIX could be rewritten in C. The switch to C provided an important benefit: **portability**. By writing C compilers for other computers at Bell Labs, the team could get UNIX running on those machines as well.

**Standardization**

C continued to evolve during the 1970s, especially between 1977 and 1979. It was during this period that the first book on C appeared.

The C programming language, written by Brian Kernighan and Dennis Ritchie and published in 1978, quickly became the bible of C programmers. In the absence of an official standard for C, this book – known as K&R or the “White Book” to aficionados – served as a de facto standard.

During the 1970s, there were relatively few programmers, and most of them were UNIX users. By the 1980s, however, C had expanded beyond the narrow confines of the UNIX world. C compilers became available on a variety of machines running under different operating systems. In particular, C began to establish itself on the fast-growing IBM PC platform.

With C`s increasing popularity came problems. Programmers who wrote new C compilers relied on K&R as a reference. Unfortunately, K&R was fuzzy about some language features, so compilers often treated this features differently. Also, K&R failed to make a clear distinction between which features belonged to C and which were part of UNIX. To make matters worse, C continued to change after K&R was published, with new features being added and a few older features removed. The need for a thorough, precise, and up-to-date description of the language soon became apparent. Without such a standard, numerous dialects would have arisen, threatening the portability of C programs, one of the language`s major strengths.

The world needed a common version of C, a standard for the language.

The development of a U.S. standard for C began in 1983 under the auspices of the American National Standards Institute (ANSI). After many revisions, the standard was completed in 1988 and formally approved in December 1989 as ANSI standard X3.159 -1989. In 1990, it was approved by the International Organization for Standardization (ISO) as international standard ISO/IEC 9899:1990. This version of the language is usually referred to as C89 or C90, to distinguish it from the original version of C, often called K&R C.

The language underwent a few changes in 1995(described in a document known as Amendment 1). More significant changes occurred with the publication of a new standard, ISO/IEC 9899:1999. The language described in this standard is commonly known as C99.

The next version after C99, C11, was published in 2011. The most recent version is C17, and is called ISO/IEC 9899:2018.

**C – Based Languages**

C as had a huge influence on modern-day programming languages, many of which borrow heavily from it. Of the many C- based languages, several are especially prominent:

* C++ includes all the features of C, but adds classes and other features to support object – oriented programming.
* Java is based on C++ and therefore inherits many C features.
* C# is a more recent language derived from C++ and Java.
* Perl was originally a fairly simple scripting language; over time it has grown and adopted many features of C.

Considering the popularity of these newer languages, it`s logical to ask whether it`s worth the trouble to learn C. I think it is, for several reasons. First, learning C can give you greater insight into the features of C++, Java, C#, and other programming languages. Programmers who learn one of these languages first often fail to master basic features that were inherited from C. Second, there are a lot of older C programs around; you may find yourself needing to read and maintain this code. Third, C is still widely used for developing new software, especially in situations where memory or processing power is limited or where the simplicity of C is desired.

If you haven`t already used one of the newer C – based languages, you`ll find that this course is excellent preparation for learning these languages. It emphasizes data abstraction, information hiding, and other principles that play a large role in object – oriented programming. C++ includes all the features of C, so you`ll be able to use everything you learn from this book if you later tackle C++. Many of the features of C can be found in the other C – based languages as well.

**Advantages and disadvantages of C**

**Advantages**

* **It is easy to understand**

One of the main reasons why people choose C over other programming languages is its simplicity. C is a highly portable language as programs coded in it are far more fast and efficient. This makes learning C easier than any other programming language. You can easily grasp the concepts behind C because there aren't many keywords or symbols involved. In addition, you don't need to be an expert in computer science to get started with C programming. All you have to do is read through some tutorials online and start writing your own codes. Also, there are system-generated functions and user-defined functions in C Language.

* **Presence of many Libraries**

C Language provides lots of built-in functions which consist of system-generated functions and user-defined functions. Many general functions can be used to develop a program, while the programmer can also create a function as per their requirements, which is called a user-generated/defined function, in C Compiler.

* **Easy to write**

Another reason why C is so popular as an efficient language among programmers is that it allows them to create their own software without having to worry about syntax errors. If you're not familiar with coding, then using structured language C will help you develop better skills. With C, you'll find yourself creating more efficient and effective solutions compared to those created by other programming languages.

* **Low cost**

If you want to build something from scratch, then C is definitely worth considering. Because of its simple structure, you won't spend too much time trying to figure out whether you've made a mistake or not when developing your program. And if you decide to hire someone else to complete the task, they would only charge you less money.

* **Fast execution speed**

If you want to execute your application quickly, then C is probably the right choice for you. Since C uses fewer instructions, it executes faster than other programming languages such as Java, Ruby, PHP, etc.

* **Portable**

Since C is based on ASCII characters, it works well across different platforms including Windows, Linux, Mac OS X, Android, iOS, etc. Therefore, you can run your C programs anywhere regardless of where you live.

* **Easy debugging**

Since C doesn't require complex statements like loops, conditionals, variables, functions, arrays, pointers, etc., you can debug your code easily. For example, if you encounter problems while executing your program, just press CTRL+D to stop the process immediately. Then, you can simply step back one line and continue working until you reach the problematic statement.

* **Procedure Oriented Language**

Users create procedures or functions to execute their tasks in C Language. It's very easy to learn a procedure-oriented language because of the way it works (algorithm to execute the statements you write). If you want to develop a program using procedure-oriented language, you need to frame an algorithm and start converting it into a function.

* **Speed of Compilation**

The C compiler produces machine code very fast. Close to a thousand lines of code can be put together in a couple of seconds. The C Compiler makes the code more efficient for faster execution.

* **Execution of algorithms and data structures**

The utilization of algorithms and data structures in C has made program calculations extremely quick and smooth. Subsequently, the C language can be employed in complex estimations and tasks like MATLAB.

* **Dynamic memory allocation**

In C Language you can allocate memory dynamically or statically. In dynamic allocation, we don't know how much space will be required for our data structure at run time. But if we use static allocation then we need to reserve a fixed amount of memory before starting the execution of the application. So this feature makes us more flexible than other languages like Java where we must declare all variables as final.

In dynamic memory allocation, you are allowed to distribute memory at run time. For instance, considering that you don't have the foggiest idea of how much memory is needed by objects in your program, then you can proceed to run a program in C and appoint the memory simultaneously.

**Disadvantages**

* **Lack of Object Orientation**

C is a very vast and powerful language and simply follows the procedural programming approach. It doesn't extend its support to the concept of OOPs (Inheritance, Polymorphism, Encapsulation, Abstraction, Data Hiding). Here, you can't create a class with multiple inheritances like Java, Python, or C++. In OOP languages like Java, we can inherit methods from the parent class. But there is nothing similar in the C language. We cannot create subclasses for our classes. So it makes it difficult to reuse existing codes.

* **Inefficient Memory Management**

In C Language you don't need any memory management techniques because it automatically manages all allocated resources for you. But if you want to use dynamic allocation then you should allocate dynamically by malloc function. If you do this manually then you will get a segmentation fault error. So we must always remember about Memory Management Techniques.

* **No Garbage Collection**

Garbage collection is a feature that automatically reclaims memory from objects no longer needed by an application or library. It can be used for both automatic and manual garbage collection. Automatic garbage collection occurs when there is insufficient free space on the heap to allocate new objects; this may occur because all available physical RAM was allocated to other processes running on the computer system. Manual garbage collection involves explicitly freeing unused blocks of memory with calls to functions such as malloc.

But in C/C++ languages, there's no such feature of garbage collection as the culture of these languages is to leave storage management to the developer or programmer. Hence, it would be technically tedious and harsh on pockets to implement a precise garbage collector for C / C++.

* **Run-time checking**

In the C programming language, the errors are not detected after each line of code. The compiler shows all the errors in the code only during the run-time of the whole which makes the checking of code (debugging) exceptionally complex in enormous projects.

Also, the compiler doesn't check whether a variable was declared before its use. The programmer must remember this rule while writing programs.

* **Concept of namespace is not present in C**

C doesn't carry out the idea of namespaces. A namespace is organized as a chain of commands to permit the reuse of names in various settings. Without namespaces, we can't pronounce two factors of a similar name.

However, C programming is devoid of this feature, and consequently, you can't characterize a variable with the same name in C.

* **Absence of Exception Handling**

Special case Handling is perhaps the main element of programming dialects. While compiling the code, different errors and bugs can happen. Exception Handling permits you to identify the bugs and rectify them. Be that as it may, C doesn't show this significant feature.

* **Lacks Constructor and Destructor**

C doesn't have any object-oriented functionalities, and hence, it doesn't have Constructor and Destructor features. So in C Language, you need to carry out the manual construction and/or destruction of the variable, either by utilizing a function or by different means.

**Why C?**

* **C is the preferred language for building a strong programming foundation**

Programming has many concepts that you need to be clear about like classes, objects, polymorphism, and inheritance among others. Other advanced languages build upon these concepts. C is the language where you can find the basis for these concepts. Hence, it is the best language for those who are new to programming.

* **It is simple to understand**

C programming language uses blocks to separate pieces of code performing different tasks. This helps make programming easier and keeps the code clean. Thus, the code is easy to understand even for those who are starting out.

* **It is used in Operating Systems**

The C programming language was created with the intention of writing UNIX operating systems. Furthermore, the execution time of programs written in C is comparable to that of assembly language, making C the most important component in the development of multiple operating systems. It was used to write the Unix kernel, Microsoft Windows utilities and operating system apps, and a large portion of the Android operating system.

* **It is used in embedded programming**

C is used in embedded programming, which is used to control micro-controllers. These micro-controllers are used in robotics, computers, and automation. Learning C can help you here as well.

* **It forms the basis for C++ and Java**

Java and C++ are amongst the most widely used programming languages today. They derive their syntax, and concepts from the C programming language. Being acquainted with the C programming basics will thus make learning C++ and Java easier.

* **It is fast**

The programs that you write in C compile and execute much faster than those written in other languages. This is because it does not have garbage collection and other such additional processing overheads. Hence, the language is faster as compared to most other programming languages. This is one of the reasons why C is used even today for programming.

* **It is used to build PC games**

C finds use in many PC games. It has been used to code various small games and some tasks in bigger games as well. Thus, learning C can be beneficial if you are aspiring to make a career in game development.

* **It is used in 3D movies**

Applications written in C and C++ are commonly used to make 3D videos, because they handle a large quantity of data and do many computations per second, these apps must be extremely efficient and quick. The less time it takes for designers and animators to create movie shots, the more money the corporation saves.

* **It gives you the best of both worlds**

There are three levels of programming languages: Low-level, Mid-level and High-level. Low-level languages process programs faster, while High-Level languages are more user and developer-friendly. The C is a mid-level language that combines the best of both. It is simple to understand, easy to use, and processes programs faster.

**The way C works**

Computers really only understand one language - machine code, a binary stream of 1s and 0'. You convert your C code into machine code with the aid of a compiler.

Graphical user interface, application

Description automatically generated

1. **Source code** – the code you write.
2. **Compiler** – checks for errors and once it`s happy, it compiles the source code.
3. **Machine** **code** – a stream o 1`s and 0`s than the computer understands.

**Requirements**

For this course you`ll need a toolchain in order to compile our programs.

Everything you need to install we`ll be free, so don’t worry.

In order to be efficient, I recommend you to create a folder with subfolders for this course, so you have everything at hand.

First, you will install Visual Studio Code, from the official website(click [here](https://code.visualstudio.com/)).

Double click on the downloaded file. Follow the steps from the images below:

Step 1:

![Graphical user interface, text, application, email

Description automatically generated]()

Step 2:

![Graphical user interface, text, application, email

Description automatically generated]()

Step 3:

![Graphical user interface, text, application

Description automatically generated]()

Step 4:

![Text

Description automatically generated with medium confidence]()

Step 5:

![Graphical user interface, text, application

Description automatically generated]()

After that, set up your folder. File -> Open Folder. In that folder you will need to have everything, that will be your “environment”.

The next thing you need to install is **docker** (click [here](https://docs.docker.com/desktop/install/windows-install/)).

Double click on the downloaded file. Follow the steps from the images below:

Step 1:

![Graphical user interface, application

Description automatically generated]()

Step 1.1(the installation may take a while)

![Graphical user interface, text, application

Description automatically generated]()

Step 2:

![Graphical user interface, application, Teams

Description automatically generated]()

Step 3: (click on the docker icon)

![Graphical user interface, text

Description automatically generated]()

Step 4: (watch the mini tutorial suggested)

![Graphical user interface, application

Description automatically generated]()

This is how you first page should look like:

![Graphical user interface, application

Description automatically generated]()

You will need an account, so press sign up and follow the steps from their website.

From our github [account](https://github.com/AROBS-Arad/Training-C), press Training C, and download the WORKSPACE folder in the folder you created before.

Now, you need to build your image.

1. Open Command Prompt
2. Change the path, to the folder where you have the Dockerfile.

cd + *path*

1. Build your image, from the dockerfile.

docker build *image\_name*\* .(this usually takes about 10 minutes).

\*our image will be named toolchain\_image

* 1. To check if your image was successfully built use the command:

docker images

You can create a container from that image:

1. Create your container using the command:
2. docker run -d *image\_name*
3. Enter the container using the command:

docker exec -it *container\_name* /bin/bash

1. In order to exit the container, without stopping it, use CTRL+p and CTRL+q.

**Binary numbers**

**The Number System**

The number system is basically used to represent the count of things. For example: How many rides are there in a circus? So more formally, Number System is a

writing system for representing numbers using digits or symbols. In the number system, there are many ways to represent a number.

There are four main types of number systems:

* Binary System (base 2)
* Octal system (base 8)
* Decimal System (base 10)
* Hexadecimal system (base 16)

The base of a number system simply means a number of different digits a particular number system uses to represent numbers.

For example:

* base 2 means two digits (0,1) are used to represent numbers.
* base 10 means 10 digits (0,1,2,3,4,5,6,7,8,9) are used to represent numbers.

We can convert a number from one number system to any other number system.

**Binary System**

The Binary Numbering System is one of the four types of number systems and the most fundamental numbering system in all digital and computer-based systems.

Why binary you ask? *Well, why decimal?*

We’ve been using decimal forever and have mostly taken for granted the reason we settled on the **base-10** number system for our everyday number needs. Maybe it’s because we have 10 fingers, or maybe it’s just because the Romans forced it upon their ancient subjugates. Regardless of what lead to it, tricks we’ve learned along the way have solidified base-10’s place in our heart; everyone can count by 10’s. We even round large numbers to the nearest multiple of 10. We’re obsessed with 10!

Computers and electronics are rather limited in the finger-and-toe department. At the lowest level, they really only have two ways to represent the state of anything: ON or OFF, high or low, 1 or 0. And so, almost all electronics rely on a **base-2** number system to store, manipulate, and math numbers.

The heavy reliance electronics place on binary numbers means it’s important to know how the base-2 number system works. You’ll commonly encounter binary, or its cousins, like hexadecimal, all over computer programs. Analysis of Digital logic circuits and other very low-level electronics also requires heavy use of binary.

So, now that we`ve seen the importance of binary numbers, let`s see how we can convert a decimal system into a binary one.

To convert numbers from decimal to binary, the given decimal number is divided repeatedly by 2 and the remainders are noted down till we get 0 as the final [quotient](https://www.cuemath.com/numbers/quotient/). The following steps are considered as the decimal to binary formula that shows the procedure of conversion.

* **Step 1:** Divide the given decimal number by 2 and note down the [remainder](https://www.cuemath.com/numbers/remainder/).
* **Step 2**: Now, divide the obtained quotient by 2, and note the remainder again.
* **Step 3:** Repeat the above steps until you get 0 as the quotient.
* **Step 4:** Now, write the remainders in such a way that the last remainder is written first, followed by the rest in the reverse order.
* **Step 5:** This can also be understood in another way which states that the Least Significant Bit (LSB) of the binary number is at the top and the Most Significant Bit (MSB) is at the bottom. This number is the binary value of the given decimal number.

Let us understand this with an example.

Diagram

Description automatically generated

Table

Description automatically generated

After noting the remainders, we will write them in such a way that the Most Significant Bit (MSB) of the binary number is written first, followed by the rest. Therefore, the binary equivalent for the given decimal number . This means that

**Hexadecimal system**

The Hexadecimal System uses the **Base of 16** system and are a popular choice for representing long binary values because their format is quite compact and much easier to understand compared to the long binary strings of 1’s and 0’s.

However, there is a potential problem with using this method of digit notation caused by the fact that the decimal numerals of 10, 11, 12, 13, 14 and 15 are normally written using two adjacent symbols. For example, if we write 10 in hexadecimal, do we mean the decimal number ten, or the binary number of two (1 + 0).

To get around this tricky problem hexadecimal numbers that identify the values of ten, eleven, . . ., fifteen are replaced with capital letters of **A**, **B**, **C**, **D**, **E** and **F** respectively.

Then in the Hexadecimal Numbering System we use the numbers from 0 to 9 and the capital letters A to F to represent its Binary or Decimal number equivalent, starting with the least significant digit at the right-hand side.

As we have just said, binary strings can be quite long and difficult to read, but we can make life easier by splitting these large binary numbers up into even groups to make them much easier to write down and understand.

For example, the following group of binary digits 1101 0101 1100 1111 are generally much easier to read and understand than 1101010111001111 when all the binary number are bunched up together.

In the everyday use of the decimal numbering system, we use groups of three digits or 000’s from the right-hand side to make a very large number such as a million or trillion, easier for us to understand and the same is also true in digital systems.

Hexadecimal Numbers is a more complex system than using just binary or decimal and is mainly used when dealing with **computers** and **memory address locations**. By dividing a binary number up into groups of 4 bits, each group or set of 4 digits can now have a possible value of between “0000” (0) and “1111” ( 8+4+2+1 = 15 ) giving a total of 16 different number combinations from 0 to 15. Don’t forget that “0” is also a valid digit.

Also, since 16 in the decimal system is the fourth power of 2 (or ), there is a direct relationship between the numbers 2 and 16 so one hex digit has a value equal to four binary digits so now q is equal to “16”.

Because of this relationship, four digits in a binary number can be represented with a single hexadecimal digit. This makes conversion between binary and hexadecimal numbers very easy, and hexadecimal can be used to write large binary numbers with much fewer digits.

The numbers 0 to 9 are still used as in the original decimal system, but the numbers from 10 to 15 are now represented by capital letters of the alphabet from A to F inclusive and the relationship between decimal, binary and hexadecimal is given below.

|  |  |  |
| --- | --- | --- |
| Decimal Number | 4-bit Binary Number | Hexadecimal Number |
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |
| 16 | 0001 0000 | 10 (1+0) |
| 17 | 0001 0001 | 11 (1+1) |
| Continuing upwards in groups of four | | |

Using the original binary number from above 1101 0101 1100 11112 this can now be converted into an equivalent hexadecimal number of D5CF which is much easier to read and understand than a long row of 1’s and 0’s that we had before.

So, by dividing a long length of binary digits into groups of 4, starting from right to left, we can convert them into hexadecimal notation and present the same digital number using fewer digits and with a much less likelihood of an error occurring. Similarly, converting hexadecimal based numbers back into binary is simply the reverse operation.

Then the main characteristic of a Hexadecimal Numbering System is that there are 16 distinct counting digits from 0 to F with each digit having a weight or value of 16 starting from the least significant bit (LSB).

As the base of a hexadecimal system is 16, which also represents the number of individual symbols used in the system, the subscript 16 is used to identify a number expressed in hexadecimal. For example, the previous hexadecimal number is expressed as:  D5CF16.

This adding of additional hexadecimal digits to convert both decimal and binary numbers into a **Hexadecimal Number** is very easy if there are 4, 8, 12 or 16 binary digits to convert. But we can also add zeros to the left of the most significant bit, the MSB if the number of binary bits is not a multiple of four.

For example, 110010110110012 is a fourteen-bit binary number that is too large for just three hexadecimal digits only, yet too small for a four-hexadecimal number. The answer is to ADD additional zeros to the left most bit until we have a complete set of four-bit binary number or multiples thereof.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Binary Number | 0011 | 0010 | 1101 | 1001 |
| Hexadecimal Number | 3 | 2 | D | 9 |

This “adding” of zero’s applies to any binary number length in order to find the equivalent hexadecimal number. So, for example, if you had a 9-bit binary number and required a 4-digit hexadecimal number (16-bits), then 7 zeros would be added to the left side of the 9-bit binary number. Giving: 00000001111111112 = 01FF16 and so on.

The main advantage of a **Hexadecimal Number** is that it is very compact and by using a base of 16 means that the number of digits used to represent a given number is usually less than in binary or decimal. Also, it is quick and easy to convert between hexadecimal numbers and binary.

Let`s try a couple of conversions:

* From binary to hexadecimal

|  |  |  |  |
| --- | --- | --- | --- |
| 11101010 | | | |
| Group the bits into four’s starting from the right-hand side | | | |
| = | 1110 | 1010 |  |
| Find the Decimal equivalent of each individual group (intermediate step, could be skipped) | | | |
| = | 14 | 10 | (In decimal) |
| Convert to Hexadecimal using the table above | | | |
| = | E | A | (In Hex) |
| Then, the hexadecimal equivalent of the binary number  1110 10102  is  #EA16 | | | |

* From hexadecimal into binary(and decimal)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | #3FA716 | | = 0011 1111 1010 01112 | | = (8192 + 4096 + 2048 + 1024 + 512 + 256 + 128 + 32 + 4 + 2 + 1) | | = 16,29510 | |

Then, the Decimal number of **16,295** can be represented as: -

#**3FA716** in Hexadecimal

or

**0011 1111 1010 01112** in Binary.

## **Hexadecimal Numbers Summary**

Then to summarize, the **Hexadecimal**, or **Hex**, numbering system is commonly used in computer and digital systems to reduce large strings of binary numbers into a set of four digits for us to easily understand. The word “Hexadecimal” means sixteen because this type of digital numbering system uses 16 different digits from 0-to-9, and A-to-F.

Hexadecimal Numbers group binary numbers into sets of four digits. To convert a binary sequence into an equivalent hexadecimal number, we must first group the binary digits into a set of 4-bits. These binary sets can have any value from 010 (0000) to 1510 (1111) representing the hexadecimal equivalent of 0 through to F.

Then we have seen here that we can convert long binary numbers into a smaller hexadecimal number using just a few digits making it easier to read, write and understand.