

The Best C Tutorial for Beginners – Is It Worth Learning C for 2019?

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 4, 2019

C is the ‘Mother of all programming languages’ and is the most prominent programming languages. If you want to step your foot forward to conquer the realm of programming, you have come to the right place. For a novice at programming, C is the best language to start off with. In this C tutorial, you will be learning:

- What is C
- Why C is important
- C’s features, pros, and cons
- What is the need to explore this language?

C is one of the most primitive languages as it is closely associated with low-level languages. It is a well-established fact that C is a high-level programming language that lies at the low-level end spectrum of a high-level language. Hence, a strong foundation in C is a must if you want to develop a career in programming.

It is the base to learn problem-solving and programming as it involves developing a logistical approach to solving the most basic problems you have encountered in your mathematics textbook in elementary school or day to day real-life situations which require a specific algorithm to be solved.

Excited to explore the C Tutorial for Beginners? Let’s begin.

1. What is C Programming Language?

C is a procedural programming language as well as a general-purpose programming language that was developed by Dennis Ritchie at AT&T’s Bell laboratories in 1972. It is an amazing and simple language that helps you develop complex software applications with ease. It is considered as the mother of all languages. C is a high-level programming language that provides support to a low-level programming language as well.

C consists of a series of concepts ranging from variables, functions, operators, scope and much more. We would be exploring all of these topics in detail in our next C tutorial.

First of all, let us start by understanding some of the basic terms related to C.

Let’s check the [best practice for C programming](#) to become an expert

2. Understanding C Language

From the above definition, we understood what is C programming but,

- What is a procedural programming language?

A procedural language follows well-organized architecture by specifying all the steps that the computer must take to obtain the desired output. Programming languages like C, Fortran, BASIC, Pascal, and C++ follow procedural programming. It is a programming paradigm consisting of a set of a computational hierarchy of steps to execute a program.

- What is meant by a high-level and a low-level programming language?

A **high-level language** is similar to the human language and is easy to understand and write. High-level language focuses more on arithmetic operations, program efficiency, and simplicity in coding. In other words, a high-level programming language is in contrast to machine-level languages. It is in close association with the language we converse in, that is, the human language. We use it in developing the program in an easy and understandable way.

A **low-level language** is a very machine friendly language. Therefore, writing programs in a low-level language is very complex for humans.

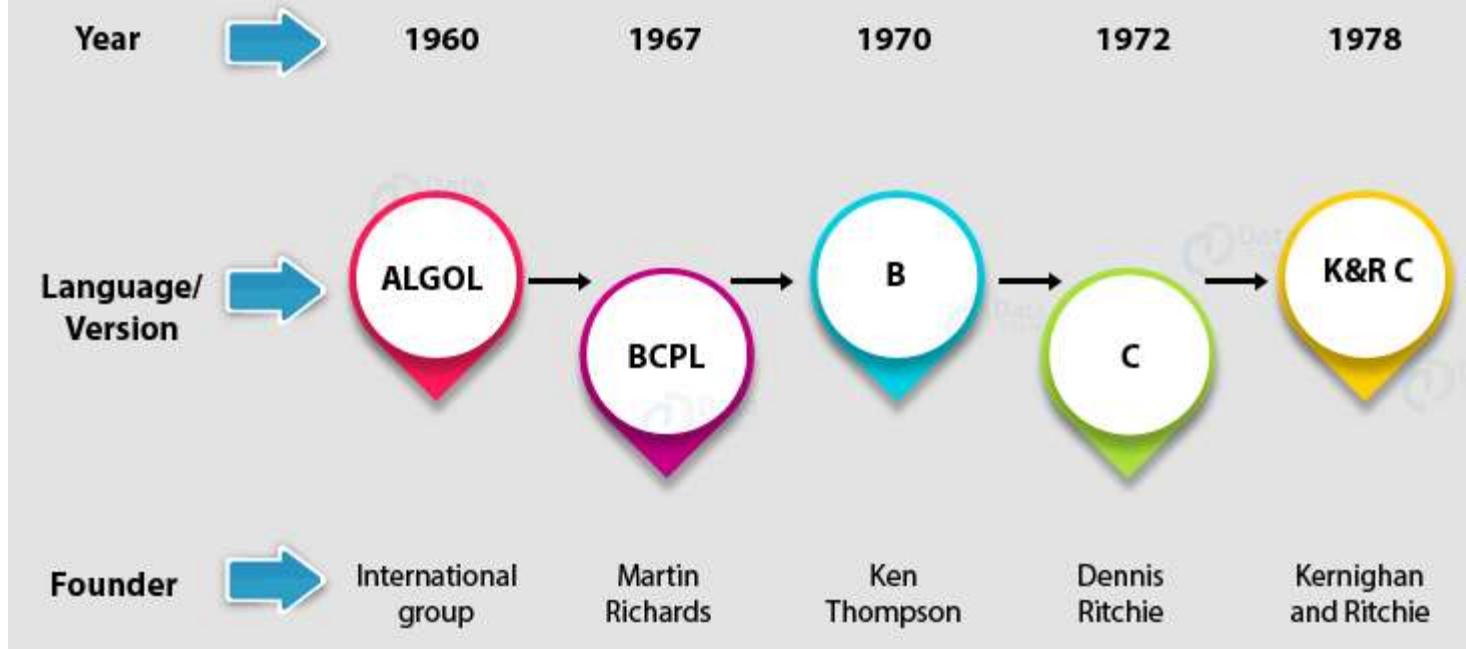
3. History of C

After the introduction of the C language, previously developed programming languages like ALGOL, COBOL, B soon lost its significance and drastically shook their well-established niche in the domain of programming.

C was developed by Dennis Ritchie in 1972. Ritchie aimed at improving B language as it was considered slow and lacked features such as byte addressability. The development of C language was followed by the origin of Unix, the first operating system implemented in a high-level language.

In 1978, Ritchie and Kernighan published the first edition of the book “The C programming language” referred to as the K&R C version of the C language. The second and improved version of the book was published by ANSI (American National Standard Institute) C standard in 1989. C grew into new and improved versions over the years. The latest version of C language is C18, extending no new features but simply certain technical corrections to anomalies in C11.

Given below is an image of the detailed history of how C evolved.



4. Why C?

The C tutorial for beginners is incomplete without the knowledge of why C programming is used. C language was developed due to various reasons that made it a very specific and convenient programming language.

The **C compiler** supports both assembly language features and high-level language and hence, it is best suitable for writing both system applications and most of the business packages. It is a portable language and hence, once the code is written, it can run on any computer system. C is basically used for developing Operating Systems. The first Operating System developed using C was Unix. Although, assembly language provides relatively higher speed and maximum control over the program it lacks portability.

From the above discussion, we inferred that the main strength of C is its capability of being functional on any computer architecture along with great flexibility and reliability. Moreover, the vast library functions pre-defined in C makes it an optimal choice for programming. Although assembly language has extremely powerful programs, it proves it be of great inconvenience when it comes to creating large applications.

Wait, check the [latest C career opportunities](#), which deny the fact of an outdated language

5. Why should beginners learn the C language?

C is the most basic language and almost all programming languages are derived from C. Other programming languages inherited their features from C and hence C is called the mother of all programming languages.

If you learn C, it becomes easy for you to learn programming languages like Ruby, [Python](#), PHP, C++, JAVA, Lua and more.

6. What is a Compiler in C?

Till now, in this C tutorial, we heard the term compilers many times, but what are compilers in C? Why are they used? Let's get the answers to these questions:

A compiler is a computer program that converts our program code to machine understandable code(binary code). The C compiler is a software application for converting the code to make it machine-friendly.

We have already discussed that a high-level programming language is machine independent and easily understandable by humans. Therefore, there is a need to convert a low-level language to a high-level language. The process of compilation is referred to as the conversion of a high-level language to low-level language.

7. How to Write a Program in C?

Now, taking a break from the theoretical portion, we are now moving towards the practical approach. First of all, we will start with the simplest program to print a message.

```
1. #include <stdio.h>
2. int main()
3. {
4. // printf() displays the string inside quotation
5. printf("Welcome to DataFlair! C tutorial");
6. return 0;
7. }
```

Output – Welcome to DataFlair! C tutorial

Get a complete guide for – [Structure of C Program](#)

Understanding every line of code

- **#include <stdio.h>** – It includes a preprocessor command. By this command, we add all the files and content of stdio.h in the program.
- **int main()** – Execution of any C program begins with this step.
- **printf()** – It sends the formatted output to the screen.
- **return 0** – This statement shows the end status, that is, the function returns whatever argument passed in return. Here, 0 is passed as the return value which means that the function returns NULL value.

8. Features of C Programming

There are various features or we can say reasons to learn C programming that make it popular in the technical and well as management industries. The [salient features of the C language](#) include:

1. **Simple and efficient** – The syntax style is easy to comprehend. We can use C to design applications that were previously designed by assembly language.
2. **Memory Management** – It allows you to allocate memory at the runtime, that is, it supports the concept of dynamic memory allocation.
3. **Dynamic Memory Allocation**- When you are not sure about the memory requirements in your program and want to specify it at the run time, that is, when you run your program, you can do it manually.
4. **Pointers** – C language provides a pointer that stores the memory address as its value. Pointers are useful in storing and accessing data from memory. We will study this in detail in our upcoming tutorials.
5. **Case Sensitive** – It is pretty clear that lowercase and uppercase characters are treated differently in C. It means that if you write “program” and “Program”, both of them would connote different meanings in C. The ‘p’ in “program” is in lowercase format whereas, the ‘P’ in Program is in uppercase format.
6. **Compiler Based** – C is a compiler based language, that is, to execute a code we first need to compile it.
7. **Structure Oriented/Modular** – C is a structured programming language. This means you can divide your code and task within a function to make it interactive. These functions also help in code reusability.

9. Applications of C Language

There is no meaning of mastering the C language, until and unless you know its real-time uses. Following are some of the [applications of C](#):

1. It is used in the development of [Operating Systems](#) and Embedded Softwares. For example, the Unix Kernel was born out of C as discussed earlier.
2. It comes in handy when designing a compiler for other programming languages.
3. Data structures and algorithms are implemented in C
4. It acts as a base language to develop new languages. For instance, C++ was developed from C.
5. Computer applications can be developed using C.
6. Firmware is designed for electrical, industrial and communication appliances using C.

10. Advantages of C Programming Language

In this section, we are going to list all the benefits of C for the user:

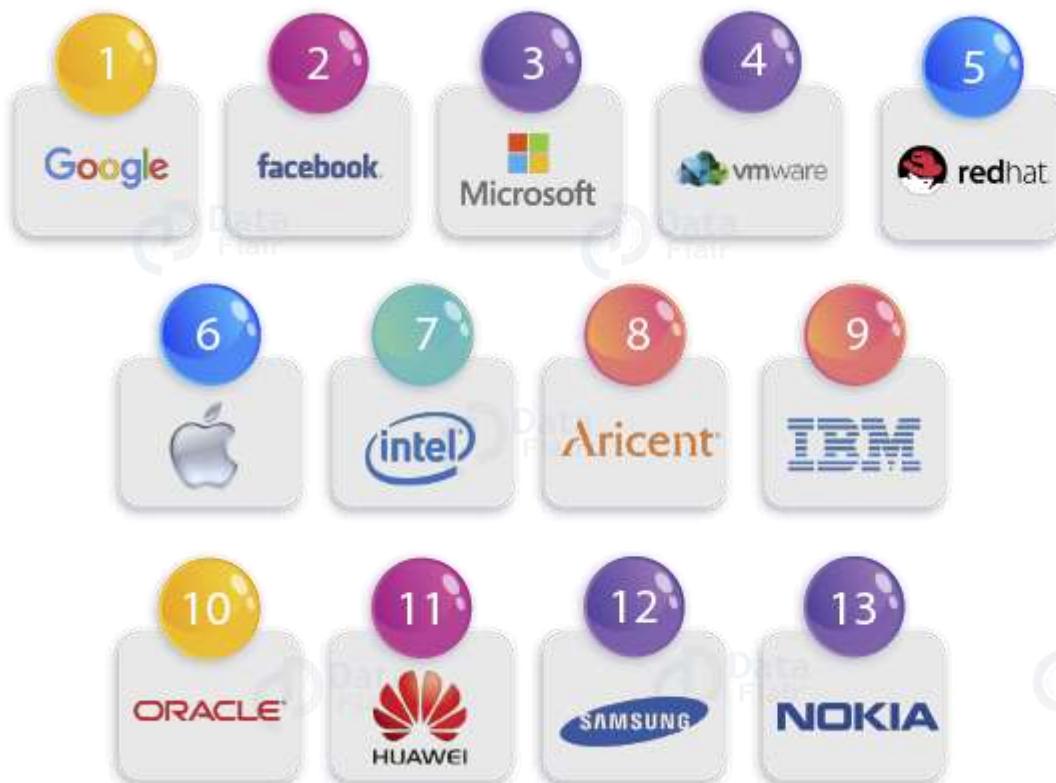
- Portable** – It is easy to install and operate and the result file is a .exe file that is easy to execute on any computer without any framework.
- Compiles faster** – C has a faster compiler that can compile 1000 lines of code in seconds and optimize the code to give speedy execution.
- User-defined functions** – C has many header files that define a lot of functions, making it easier for you to code. You can also create your functions; these are called user-defined functions (UDFs).
- C has a lower level of abstraction** – C is a very clear and descriptive language. You can, in a way, directly see into the machine without any conceptual hiding and so learning C first makes the concepts very clear for you to proceed.

Note: Abstraction means Data Hiding

Explore more [Advantages and Disadvantages of C](#)

11. Companies Using C

Almost all the companies that work on firmware, gaming, networking, graphics use C. Some of the companies that use C to write algorithms are :



12. Career Aspects in C Programming Language

Up till now, in this C tutorial, we learned all the basic concepts of C programming. But, what about its career opportunities?

C language is always in demand and you can grab a list of opportunities for your career growth. Many people talk about moving to advanced languages like Java, and Python. But, the core of every machine remains the same – C.

There are **dozens of jobs available** if you are clear with your programming concepts.

- Companies that work on embedded programming can be an excellent option.
- If you are interested in **Robotics** and other security devices or electronic devices, you should learn c programming to develop basic algorithms for various microcontrollers.
- You can become a Software Engineer or a Team Leader if you are good at Data Structures.

You don't need to search for career possibilities, just be confident about what you learn and implement, using logic and proper application of all the protocols of a programming language.

Hence, you can build your desired career if you excel in it what you study.

Salary Prospects

“The more you learn, the more you earn.”

This comes to be the first concern as of now. Yes, money is important but at the starting of your career whether you are a student or someone looking to learn new technology, focus on the ways that can help you grow.

13. Summary

C is considered as the mother of all programming languages as it has built a strong base for all programming languages like Java, Python, C++, etc. Let us summarise what we have learned in this C tutorial. We understood the history of C language, and also how it became so popular. We also discussed some of the basic features, advantages, and limitations that the C language possesses. At last, we discussed the career and salary prospects in C language.

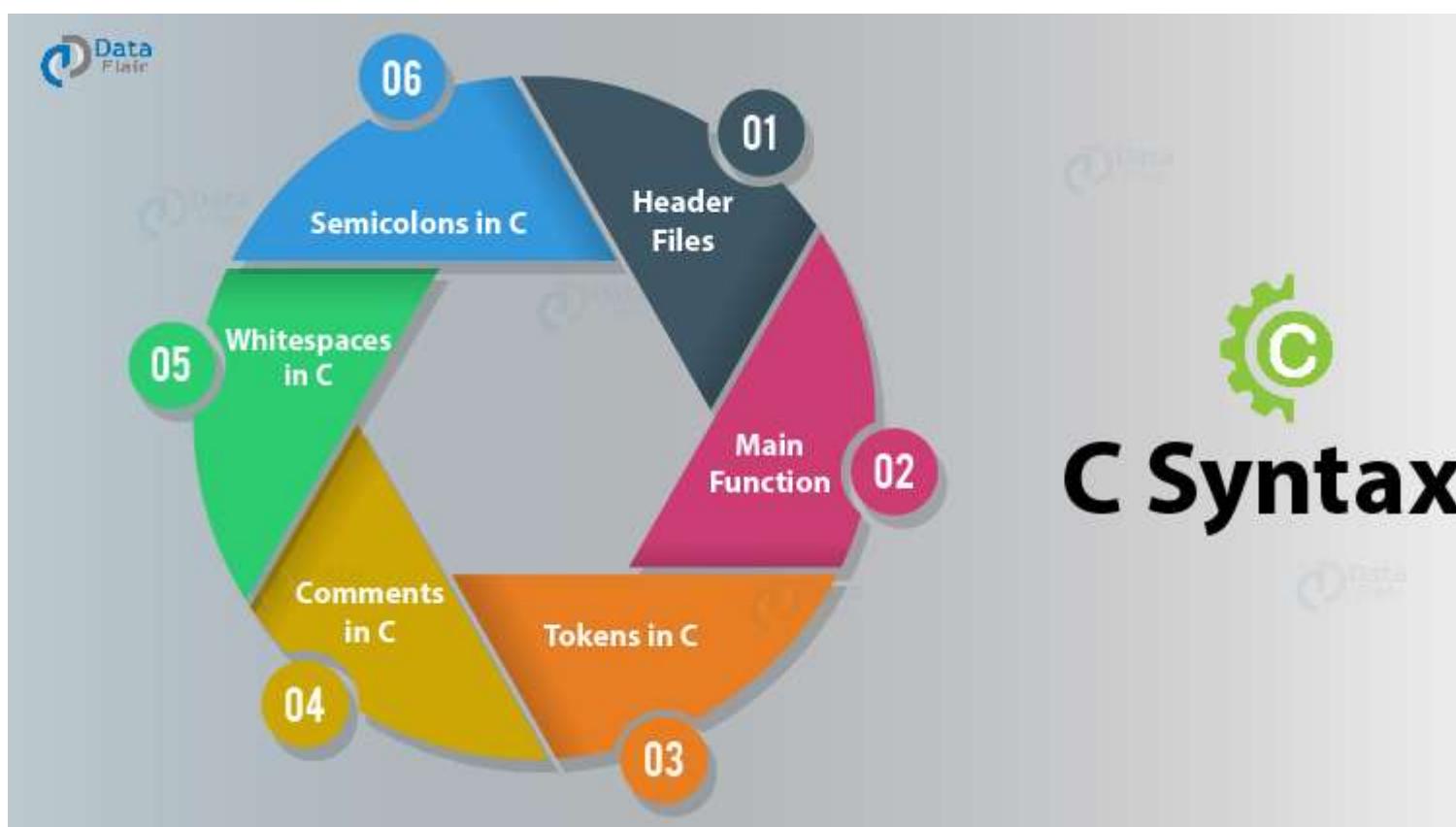
Hopefully, C tutorial for beginners helped you to develop a basic understanding of the C programming language. Please share your feedback in the comment section below.

C Syntax Rules – Learn the ABCs of Programming in C Language

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 3, 2019

In this tutorial, we will be discussing the C syntax in detail. If you have gone through our previous tutorials you might have understood the importance of [C language](#). We will now focus on the syntax and the program structure in C. We will not be discussing any complex program code at this stage. Hence, this tutorial will help you start with a basic C program in a precise manner. At the end of this tutorial, you will be able to explain each line of code.

So, let's start today's journey with C basic Syntax.



C Basic Syntax

Syntax basically refers to the protocols to be followed while writing a program. It is very necessary to follow proper syntax while coding to get the desired set of output. The C basic syntax consists of header files, main function, and program code. This is the most fundamental structure in the C program. A C program necessarily consists of the main function because the execution of the program starts from this line. Without the main function, the program execution does not start.

1. `#include<stdio.h>`
2. `int main() // main function with integer return type`
3. `{`

```
4. printf("Welcome to DataFlair tutorials!\n"); // print statement to display output on the screen
5. return 0; // Indicates that the main function returns null value
6. }
```

Do you know what is the reason behind the popularity of C?

Code on Screen-

The screenshot shows a terminal window titled "dataflair@asus-System-Product-Name". The window has a dark background and a light-colored text area. At the top, there's a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". Below the menu is a status bar showing "GNU nano 2.9.3" on the left and "syntax.c" on the right. The main text area contains the following C code:

```
#include<stdio.h>
int main() // main function with integer return type
{
    printf("Welcome to DataFlair tutorials!\n"); // print statement to display output on the screen
    return 0; // Indicates that the main function returns null value
}
```

Output-

dataflair@asus-System-Product-Name

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ gcc syntax.c -o syntax
dataflair@asus-System-Product-Name:~/Desktop$ ./syntax
Welcome to DataFlair tutorials!
dataflair@asus-System-Product-Name:~/Desktop$ █
```

From the above C syntax, we understood the placing of these elements. Now, let's discuss each element of the above code in depth.

1. Header Files

In the above code, we have used two header files.

```
#include <stdio.h>
#include<conio.h>
```

C has a series of predefined functions embedded in the header files in the C library. We use these functions to perform a specific task with the help of a header file.

What are header files?

Header files contain a bunch of files that help the user to include the predefined functions according to his requirements. You can add header files using the preprocessor directive `#include`.

You can [create your own header file in C](#) within seconds

2. Main Function

When your operating system runs a program in C, it gives the control of the computer to the C program. If a program consists of only one line of code in the main function, then the program can run successfully.

As the program execution starts from this line, it becomes compulsory for every C program to have the main function.

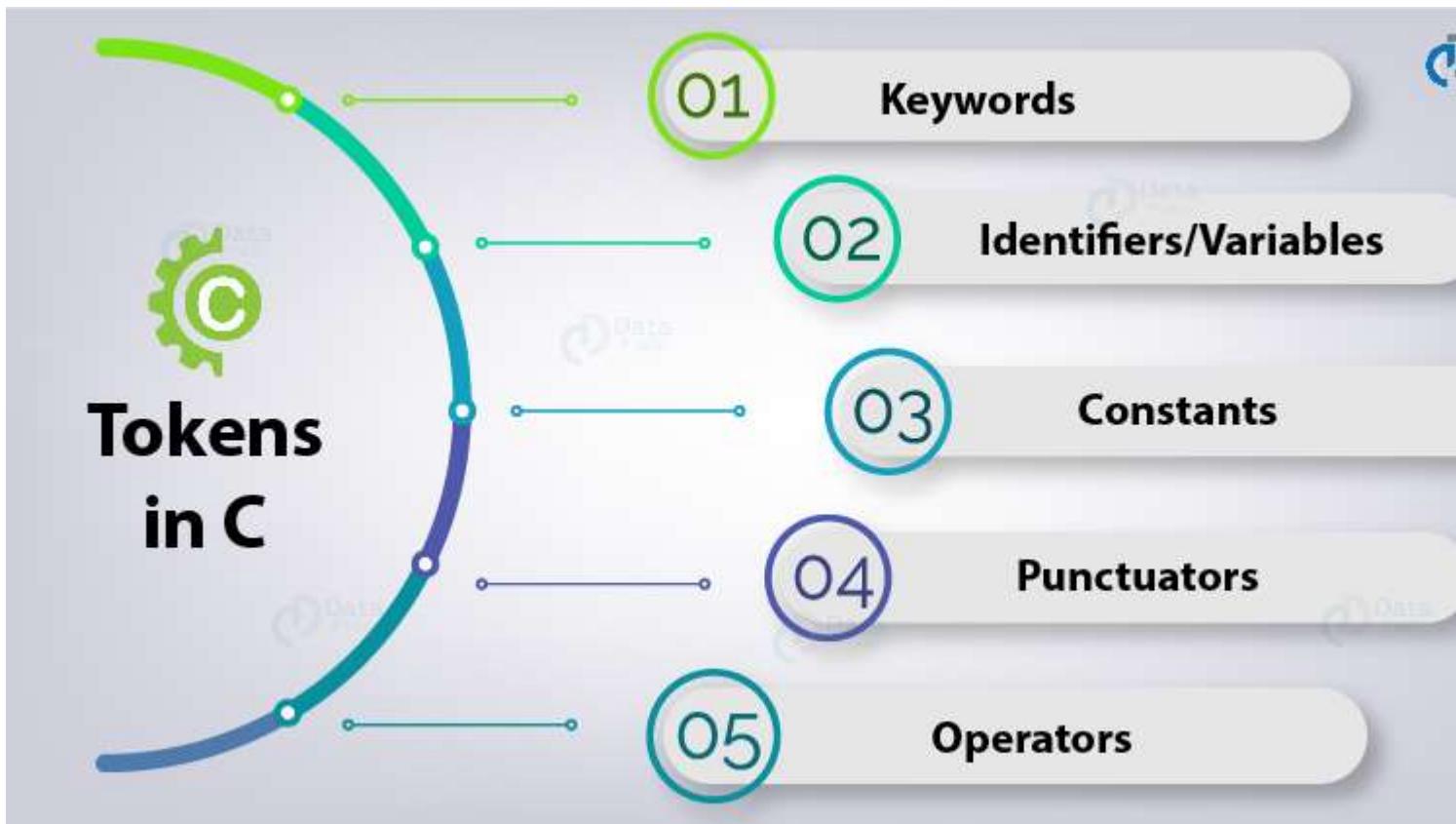
- `clrscr()` – This function is predefined in the header file `<conio.h>`. `clrscr()` helps to clear the screen.
- `printf()` – This is another function that helps you to display the message on the output screen. This function is defined in `<stdio.h>`. In the above program, it is pretty clear that the `printf()` function helps you display the output message: “Welcome to DataFlair”.
- `getch()` – This function is defined in `conio.h` and is used to hold the output screen until we hit the next key after we run the program.

This was a line to line description of the simplest code that can be built in C. With this, you might be clear with the flow of a C [syntax](#). We will further discuss more complex programs in the simplest form of our upcoming C tutorials.

3. Tokens in C

A C syntax/program consists of a series of tokens. They can be identifiers, variables, symbols. We can say that they are *the smallest units of a program that convey a specific meaning to the compiler*. There are 5 [types of tokens in the C language](#). They are :

1. Keywords
2. Identifiers/Variables
3. Constants
4. Punctuators
5. Operators



3.1 Keywords in C

Keywords are a set of words used for a special purpose in the program. They are reserved words. Some examples are for, while, switch, break, goto, etc.

```
1. #include<stdio.h>
2. int main()
3. {
4.     for(int i=0;i<3;i++)
5.     {
6.         printf("Welcome to DataFlair tutorials!\n");
7.     }
8.     return 0;
9. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

keywords.c

```
#include<stdio.h>
int main()
{
for(int i=0;i<3;i++)
{
printf("Welcome to DataFlair tutorials!\n");
}
return 0;
};
```

Output-

In this example, we have used a keyword to print a message using the `printf()` [function in C](#).

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ touch keywords.c
dataflair@asus-System-Product-Name:~/Desktop$ gcc keywords.c -o keywords
dataflair@asus-System-Product-Name:~/Desktop$ ./keywords
Welcome to DataFlair tutorials!
Welcome to DataFlair tutorials!
Welcome to DataFlair tutorials!
dataflair@asus-System-Product-Name:~/Desktop$ █
```

3.2 Variables in C

It is a *data item in the program whose value can be changed* and thus, it is called variable. They are a combination of letters and digits.

Rules for naming identifiers or variables

- No special character can be used to name a variable except underscore(_).
- The variable name cannot start with a digit, it can be a letter or an underscore.
- No keyword can be used as an identifier.

Don't forget to check, [how to initialize, declare and access variables in C](#)

3.3 Constants in C

Constants are those values that cannot be changed during program execution.

3.4 Punctuators in C

Punctuators are used for formatting the C program statements and expressions.

Examples for punctuators are : [] { } () , ; : * =

Let us talk about one of the punctuators i.e., semicolons

A semicolon is used to show the ending of a statement. It acts as a separator between two statements. If you miss this statement at the end of any line, the compiler would join the statement with the next line of code and will give a **syntax error**.

3.5 Operators in C

There are various [operators in C](#) that we use in order to *perform different mathematical operations*. They are:

- Arithmetic Operators : +, -, *, /
- Increment/Decrement Operators : ++, -
- Relational Operators: ==, <, >, <=, >=, !=
- Logical operators : &&, ||, !
- Conditional Operators : ?:

4. Comments in C

A comment is a simple text that makes your code more descriptive and readable. You can add comments in your code just to explain how the code is functioning. Comments aren't a necessary part of a code, you can add them according to your convenience.

There are two types of comments:

1. Single line comments
2. Multi-line comment

For single line comment just put '//' before the comment line

For multi-line comment /*---*/ , the statements of comment should be enclosed within /* and */.

```
1. #include<stdio.h>
2. int main()
3. {
4.     printf("Welcome to DataFlair tutorials!\n"); // This is a single line comment
5.     /* This is a multi-line comment */
6. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

comments.c

```
#include<stdio.h>
int main()
{
printf("Welcome to DataFlair tutorials!\n"); // This is a single line comment
/* This is a multi-line comment */
};
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ touch comments.c  
dataflair@asus-System-Product-Name:~/Desktop$ gcc comments.c -o comments  
dataflair@asus-System-Product-Name:~/Desktop$ ./comments  
Welcome to DataFlair tutorials!  
dataflair@asus-System-Product-Name:~/Desktop$ □
```

5. Whitespaces in C

In a C program, whitespace is generally a blank line. The C programming language uses whitespaces to describe blanks, newline characters, and comments.

Whitespace separates one statement from another statement for the compiler to identify. In this statement,

```
int data ;
```

If you don't provide proper whitespace within the statements where it is needed, then the compilation will not be correct and you will not be able to get the desired output.

For example: If you have not provided whitespace between int and data, then it would create a compilation error.

Master the [structure of C program](#) in just 7 mins

6. Semicolons in C

Semicolons are *provided to terminate the statement*. A Semicolon indicates the start and end of statement execution.

This tells the compiler that the statement is completed. If a semicolon is not placed at the end of a statement, then the program will not compile generating a compilation error message.

In the above code, all the statements within the function terminate with a semicolon. If the semicolon is not provided, it will generate “Compilation Error”.

Rules for a Program in C



The screenshot shows a C program code editor with several annotations:

- A red arrow points to the first two lines of code: `#include<stdio.h>` and `#include<conio.h>`, with the text "Including Header Files" above them.
- A red arrow points to the `main()` keyword in the `void main()` declaration, with the text "main() Function Must Be There" below it.
- A red arrow points to the `// helps to print the message "Welcome to DataFlair"` comment, with the text "Single Line Comment" to its left.
- A red arrow points to the `getch();` call, with the text "Semicolon After Each Statement" below it.
- A red arrow points to the closing brace `};`, with the text "Program Enclosed Within Curly Braces" below it.

1. C is a case sensitive language. Most of the program statements are in lower case.
2. All statements must necessarily terminate with a semicolon.
3. Whitespaces should be provided only between variables and keywords.

Summary

We hope you understood the basic syntax of the C programming language with the program structure. Now, you can easily apply tokens, keywords, comments and other essential elements of C language in your program. These elements of the C syntax play a very important role in programming in C and makes you proficient in the C programming language.

8 Crucial Features of C Language – What is the Reason Behind its Popularity?

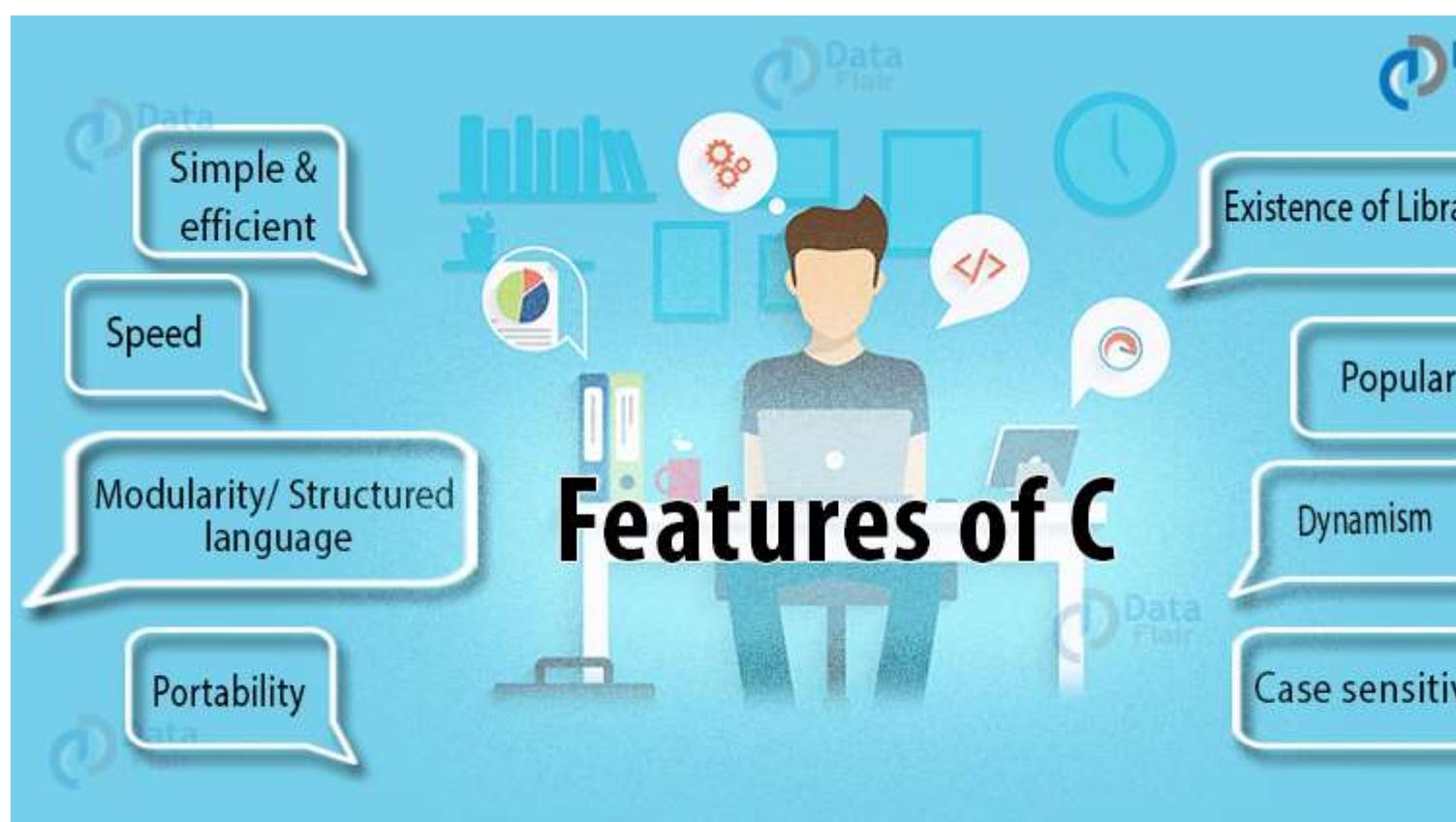
BY [DATAFLAIR TEAM](#) · UPDATED · JULY 4, 2019

Have you ever wondered, why C is still so popular? Today, the market is lead by two strong programming languages – Java and Python. But the demand for C programming language is not affected by these two. What is so special about C? Why are industries hiring C professionals? All of these queries can be resolved by discussing the features of the C language.

Features of C programming language will brief you with its unique qualities. So, don't waste your time and explore the reasons to learn C.

Best Features of C Programming Language

We can differentiate every programming language by its features as it makes the language special and unique. These are some major features of C programming that makes it demanding in IT sector.



1. Portability

It refers to the usability of the same fragment of code in different environments. C programs are capable of being written on one platform and being run on another with or without any modification. Let's understand with an example-

Suppose we wrote a program to find the area of a right-angled triangle on CodeBlocks using C language, the same code can be written or modified by generalizing any triangle using Heron's formula on Turbo C3; it will work nonetheless, provided the modification made is error-free.

Before we move you should know the concept of [Functions in C/C++](#)

2. Modularity/Structured Language

This feature of C language allows the program to be splintered (broken) into smaller units and run individually with the help of functions. In simple words, modular programming refers to the software design technique, which increases the number of fragments of the same code. For instance, you want to find the area of a square, a rectangle and, a triangle. Instead of writing the code as a whole, we can divide it into separate functions, one for finding the area of a square, a rectangle, and triangle respectively. It guarantees fewer chances of errors and makes it visually appealing and more organized.

3. Simple and Efficient

The [syntax style of C programming](#) is easy to comprehend and can be used to design applications that were previously designed by assembly language. In high schools or colleges, C is generally taught as an introductory programming language as it is a well-established fact that it is easier to learn any other programming language in the long run if you are well acquainted with C.

4. Speed

Since it is a compiler-based language, it is comparatively faster than other programming languages like [Java or Python](#), which are interpreter based. A compiler considers the entire program as input and thereby generates an output file with the object code whereas an interpreter takes instruction by instruction as input and then generates an output but does not generate a file.

5. Popular

It is one of the most extensively used languages in the development of operating and embedded systems. Needless to mention how popular it is.

6. Existence of Libraries

C language comprises of its library which has a wide range of built-in functions. Even the user-defined functions can be added to the C library. It gives the user a wide latitude of scope to develop his own functions for implementing problems for later use and implementation.

7. Dynamism

It supports the *feature of DMA* (Dynamic Memory Allocation), which helps in the utilization and management of memory. Among all the features of C, dynamism is unique. Using DMA, the size of a data structure can be changed during runtime using some predefined functions in the C library such as [*malloc\(\), calloc\(\), free\(\) and realloc\(\)*](#).

8. Case Sensitive

It treats lowercase and uppercase characters differently. For instance, if we declare a variable ‘x’ of integer type, it would connote a different meaning altogether if we type ‘X’ rather than ‘x’.

Features of C language have not ended yet. Now, we are going to uncover the secret behind its popularity.

Why C is Popular?

After the birth of Unix because of the advent of the C language, it revolutionized the kingdom of science and technology. The various features of C make it popular. In today’s world, every programmer or software developer who has achieved milestones in his life is very much familiar with the concept of [C](#) programming. Not only is it a fundamental language but it is permissive in nature as well. It allows the user to manage program memory as it offers the feature of dynamic memory allocation which makes it much faster than any other language. Today, every computer literate person is aware of the term “C Programming”.

Summary

We shed light on the salient features of C programming by motivating the learners to study this language for the features it offers, making it unique and ubiquitous. We successfully discovered the secret behind the popularity of C. Currently, C is used in many popular industries, and its popularity and shine are not going to fade at least till the next decade.

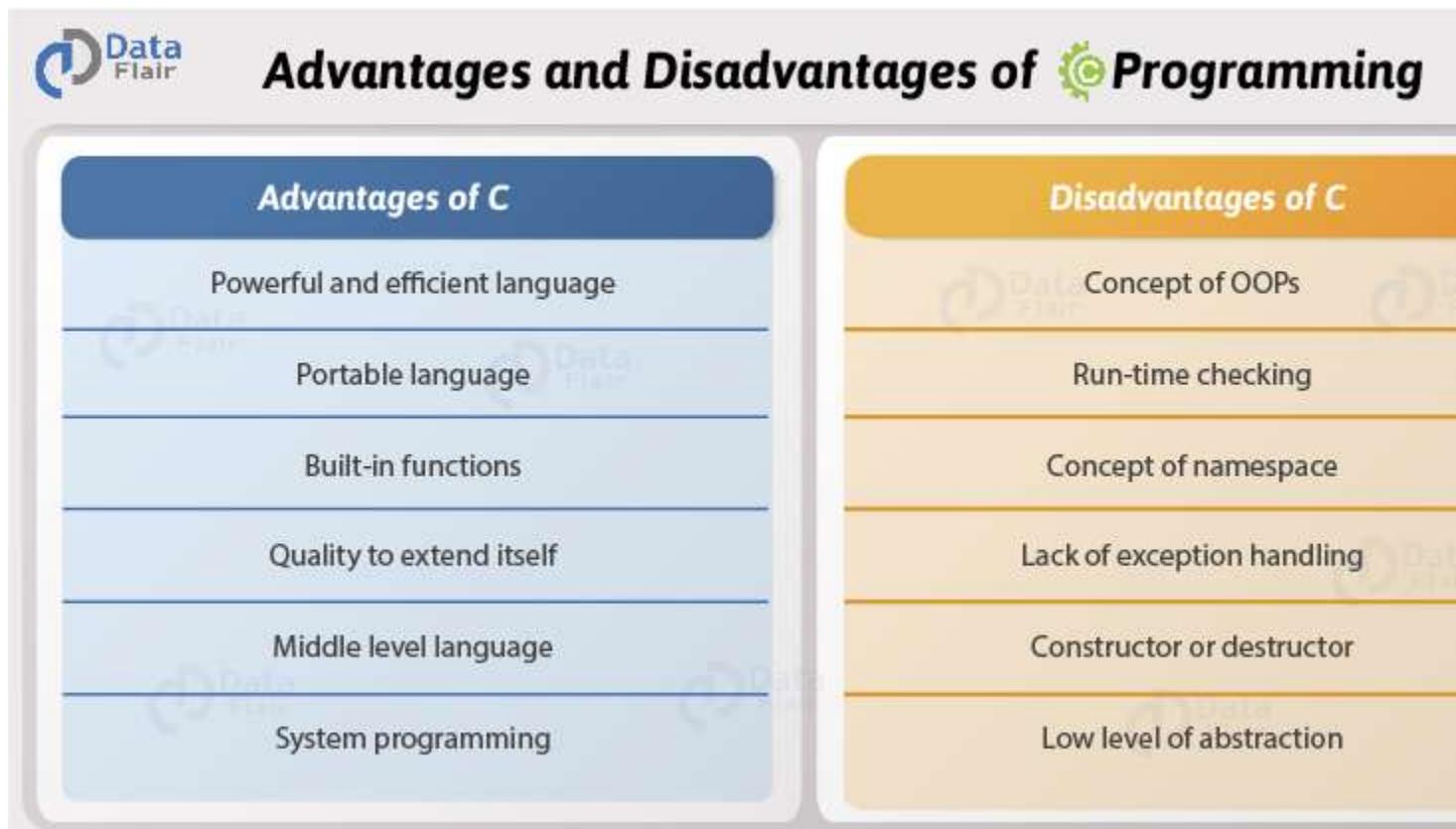
Advantages and Disadvantages of C Programming – Discover the Secrets of C

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 4, 2019

C is one of the oldest languages that cannot be replaced by any other language, as *it is the core of every machine*. In the 21st century, where languages like Java, [Python are reigning the market](#), you should not worry about how relevant the C programming language is and if you are still not sure, proceed with this tutorial “Advantages and Disadvantages of C” to know how powerful the C language is. Well, everything has its strengths and limitations that make it unique.

“Don’t judge a book by its cover.”

So, let’s begin our discussion on the advantages and disadvantages of C.



Advantages and Disadvantages of C language

First of all, let us discuss *what makes* C language the mother of all languages. There are various benefits of C programming that depends on these positive points which can surely define the functionality of C in a better manner.

1. Advantages of C Programming Language

1.1. Building block for many other programming languages

C is considered to be the most fundamental language that needs to be studied if you are beginning with any programming language. Many programming languages such as Python, C++, Java, etc are built with the base of the C language.

1.2. Powerful and efficient language

C is a robust language as it contains many data types and operators to give you a vast platform to perform all kinds of operations.

Take a break & Learn [Different Data Types in C](#)

1.3. Portable language

C is very flexible, or we can say machine independent that helps you to run your code on any machine without making any change or just a few changes in the code.

1.4. Built-in functions

There are only 32 keywords in ANSI C, having many built-in functions. These functions are helpful when building a program in C.

1.5. Quality to extend itself

Another crucial ability of C is to extend itself. We have already studied that the C language has its own set of [functions in the C](#) library. So, it becomes easy to use these functions. We can add our own functions to the C Standard Library and make code simpler.

1.6. Structured programming language

C is structure-based. It means that the issues or complex problems are divided into smaller blocks or functions. This modular structure helps in easier and simpler testing and maintenance.

1.7. Middle-level language

C is a middle-level programming language that means it supports high-level programming as well as low-level programming. It supports the use of kernels and drivers in low-level programming and also supports system software applications in the high-level programming language.

1.8. Implementation of algorithms and data structures

The use of algorithms and data structures in C has made program computations very fast and smooth. Thus, the C language can be used in complex calculations and operations such as MATLAB.

1.9. Procedural programming language

C follows a proper procedure for its functions and subroutines. As it uses procedural programming, it becomes easier for C to identify code structure and to solve any problem in a specific series of code. In procedural programming *C variables* and functions are declared before use.

1.10. Dynamic memory allocation

C provides dynamic memory allocation that means you are free to allocate memory at run time. For example, if you don't know how much memory is required by objects in your program, you can still run a program in C and assign the memory at the same time.

1.11. System-programming

C follows a system based programming system. It means the programming is done for the hardware devices.

So, with this, we are aware of *why C considered a very powerful language and why is it important to know the advantages of C?*

When we study anything new, it becomes important to know the benefits that we gain from that technology. This allows us to grow our interest and implement our knowledge in a practical scenario. Now, let us move on to the “Advantages and Disadvantages of the C Programming Language”.

2. Disadvantages of C Programming language

We have already discussed the advantages of C.

You might be thinking about why we are not approaching the language practically and studying the theoretical part in every tutorial. It is because if you will understand the basic functionalities of the language and the methods or operation of the programming language, it becomes easy for you to know whether this language is *suitable for your career or not*.

Also, with the basic knowledge of the C language, you can understand the flow of any program.

So, now let us see what the limitations of C programming language are-

1. Concept of OOPs

C is a very vast language, but it does not support the concept of OOPs (Inheritance, [Polymorphism](#), Encapsulation, Abstraction, Data Hiding). C simply follows the procedural programming approach.

2. Run-time checking

In the C programming language, the errors or the bugs aren't detected after each line of code. Instead, the compiler shows all the errors after writing the program. It makes the checking of code very complex in large programs.

3. Concept of namespace

C does not implement the concept of namespaces. A namespace is structured as a chain of commands to allow the reuse of names in different contexts. Without namespaces, we cannot declare two variables of the same name.

But, C programming lacks in this feature, and hence you cannot define a variable with the same name in C.

4. Lack of Exception Handling

Exception Handling is one of the most important features of programming languages. While compiling the code, various anomalies and bugs can occur. Exception Handling allows you to catch the error and take appropriate responses. However, C does not exhibit this important feature.

5. Constructor or destructor

C does not have any constructor or destructor.

Constructors & Destructors support basic functionality of Object Oriented Programming. Both are member functions that are created as soon as an object of the class is created. You will be studying constructor and destructor in detail later on.

6. Low level of abstraction

C is a small and core machine language that has minimum data hiding and exclusive visibility that affects the security of this language.

Summary

Here, we end our tutorial on ‘Advantages and Disadvantages of C Programming’. We hope you found this tutorial beneficial in developing a simple understanding of the pros and cons of C. You might have understood how powerful C is in its process of implementation and execution and at the same time how it lags behind other languages in certain aspects.

Applications of C Programming That Will Make You Fall In Love With C

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 4, 2019

A popular myth about *C* is that it has become obsolete and no one is using it anymore.

Generally, people think that applications of C programming are decreasing day by day. But, it is an inevitable fact that *C* is regarded as one of the oldest and fundamental languages widely used all over the world. The knowledge of programming is incomplete without the incorporation of the C language. It continues to dominate the realm of programming.

If you want to conquer any technological domain, then you must develop strong roots from the beginning.

Check the latest [Career Opportunities in C](#)

Various Real-World Applications of C Programming

The applications of C are not only limited to the development of operating systems, like Windows or Linux, but also in the development of GUIs (Graphical User Interfaces) and, IDEs (Integrated Development Environments). Here are some *striking applications offered by the C programming language*:



1. Operating Systems

The first operating system to be developed using a high-level programming language was UNIX, which was designed in the C programming language. Later on, Microsoft Windows and various Android applications were scripted in C.

2. Embedded Systems

The C programming language is considered an optimum choice when it comes to scripting applications and drivers of embedded systems, as it is closely related to machine hardware.

3. GUI

GUI stands for Graphical User Interface. Adobe Photoshop, one of the most popularly used photo editors since olden times, was created with the help of C. Later on, Adobe Premiere and Illustrator were also created using C.

4. New Programming Platforms

Not only has C given birth to C++, a programming language including all the **features of C** in addition to the concept of object-oriented programming but, various other programming languages that are extensively used in today's world like MATLAB and Mathematica. It facilitates the faster computation of programs.

5. Google

Google file system and Google chromium browser were developed using C/C++. Not only this, the Google Open Source community has a large number of projects being handled using C/C++.

6. Mozilla Firefox and Thunderbird

Since Mozilla Firefox and Thunderbird were open-source email client projects, they were written in C/C++.

7. MySQL

MySQL, again being an open-source project, used in Database Management Systems was written in C/C++.

8. Compiler Design

One of the most popular uses of the C language was the creation of compilers. Compilers for several other programming languages were designed keeping

in mind the association of C with low-level languages, making it easier to be comprehensible by the machine.

Several popular compilers were designed using C such as Bloodshed Dev-C, Clang C, MINGW, and Apple C.

7 [Basic C Programs](#) that will help you to rise from Noob to Pro

9. Gaming and Animation

Since the [C](#) programming language is relatively faster than Java or Python, as it is compiler-based, it finds several applications in the gaming sector. Some of the most simple games are coded in C such as Tic-Tac-Toe, The Dino game, The Snake game and many more. Increasing advanced versions of graphics and functions, Doom3 a first-person horror shooter game was designed by id Software for Microsoft Windows using C in 2004.



Summary

Many of the world's leading companies are using C programming for their professional use, which clears the fact that C is not an outdated language. It is still the most preferred language for programmers and back-end developers. Here, we got an insight into the applications of C programming in the real world.

We inferred that C is used in all spheres of hardware and software development, making it useful for upcoming software developers and software professionals, of course, who have great command over C to design complex interfaces.

How to Install C – Learn to Install GCC Compiler for Ubuntu

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 3, 2019

How to Install C Programming Language?

Still stuck with the same problem of how to install C? You cannot install a programming language. Since C is a programming language, you directly start writing programs in it.

Then what do you install? The answer is- you need to install a Compiler, i.e. GCC Compiler, to compile and run your programs using it. This compiler recognizes the language by its syntax.

Steps to Install C Language

In this installation tutorial, we will help you to write and compile your own programs in the C programming language using terminal for Linux Operating System.

1. Environment for C programming

The terminal command prompt is pre-installed in your LINUX operating system. Our task now is to install the GCC compiler using terminal to execute the C programs.

Check the [8 Crucial Features of C Language](#) & find the Reason behind its Popularity

2. How to install GCC compiler using terminal in Linux?

You can find the terminal icon on your Desktop screen. If not, you can find it through the search menu. A shortcut to open terminal is: Press Ctrl + Alt + T.

Open terminal so that we can proceed with the further steps to install the GCC Compiler.

The main command for installing the GCC compiler using terminal on Ubuntu is:

sudo apt install GCC

Here, GCC is the C compiler.

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~$ sudo apt install gcc  
[sudo] password for dataflair: []
```

GCC is the C compiler

If you do not have permission to install the [GCC](#) compiler, this message would be displayed where you are required to enter the admin password to get the permission granted:

If you have permission, then the installation process will proceed as follows:

admin2@admin2: ~

File Edit View Search Terminal Help

admin2@admin2:~\$ sudo apt-get install gcc

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following additional packages will be installed:

 cpp cpp-7 gcc-7 gcc-7-base gcc-8-base libasan4 libatomic1 libc-dev-bin libc6-dev
 libitm1 liblsan0 libmpx2 libquadmath0 libstdc++6 libtsan0 libubsan0 linux-libc-de

Suggested packages:

 cpp-doc gcc-7-locales gcc-multilib make autoconf automake libtool flex bison gcc-
 libitm1-dbg libatomic1-dbg libasan4-dbg liblsan0-dbg libtsan0-dbg libubsan0-dbg l

The following NEW packages will be installed:

 gcc gcc-7 libasan4 libatomic1 libc-dev-bin libc6-dev libcilkrt5 libgcc-7-dev lib
 linux-libc-dev manpages-dev

The following packages will be upgraded:

 cpp cpp-7 gcc-7-base gcc-8-base libcc1-0 libgcc1 libgomp1 libstdc++6

8 upgraded, 16 newly installed, 0 to remove and 232 not upgraded.

Need to get 24.2 MB of archives.

After this operation, 73.4 MB of additional disk space will be used.

Do you want to continue? [Y/n] y

GCC Compiler Installation Process

Type ‘y’ when the command prompt asks “Do you want to continue?” and then press Enter.

Thereafter, your installation will begin

admin2@admin2: ~

```
File Edit View Search Terminal Help
Unpacking gcc (4:7.4.0-1ubuntu2.2) ...
Selecting previously unselected package libc-dev-bin.
Preparing to unpack .../15-libc-dev-bin_2.27-3ubuntu1_amd64.deb ...
Unpacking libc-dev-bin (2.27-3ubuntu1) ...
Selecting previously unselected package linux-libc-dev:amd64.
Preparing to unpack .../16-linux-libc-dev_4.15.0-51.55_amd64.deb ...
Unpacking linux-libc-dev:amd64 (4.15.0-51.55) ...
Selecting previously unselected package libc6-dev:amd64.
Preparing to unpack .../17-libc6-dev_2.27-3ubuntu1_amd64.deb ...
Unpacking libc6-dev:amd64 (2.27-3ubuntu1) ...
Selecting previously unselected package manpages-dev.
Preparing to unpack .../18-manpages-dev_4.15-1_all.deb ...
Unpacking manpages-dev (4.15-1) ...
Setting up libquadmath0:amd64 (8.3.0-6ubuntu1~18.04) ...
Setting up libgomp1:amd64 (8.3.0-6ubuntu1~18.04) ...
Setting up libatomic1:amd64 (8.3.0-6ubuntu1~18.04) ...
Setting up libcc1-0:amd64 (8.3.0-6ubuntu1~18.04) ...
Setting up libtsan0:amd64 (8.3.0-6ubuntu1~18.04) ...
Setting up linux-libc-dev:amd64 (4.15.0-51.55) ...
Setting up liblsan0:amd64 (8.3.0-6ubuntu1~18.04) ...
Setting up gcc-7-base:amd64 (7.4.0-1ubuntu1~18.04) ...
Setting up libmpx2:amd64 (8.3.0-6ubuntu1~18.04) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up libc-dev-bin (2.27-3ubuntu1) ...
Setting up manpages-dev (4.15-1) ...
Setting up libc6-dev:amd64 (2.27-3ubuntu1) ...
Setting up libitm1:amd64 (8.3.0-6ubuntu1~18.04) ...
Setting up libasan4:amd64 (7.4.0-1ubuntu1~18.04) ...
Setting up libcilkrt5:amd64 (7.4.0-1ubuntu1~18.04) ...
Setting up libubsan0:amd64 (7.4.0-1ubuntu1~18.04) ...
Setting up libgcc-7-dev:amd64 (7.4.0-1ubuntu1~18.04) ...
Setting up cpp-7 (7.4.0-1ubuntu1~18.04) ...
Setting up cpp (4:7.4.0-1ubuntu2.2) ...
Setting up gcc-7 (7.4.0-1ubuntu1~18.04) ...
Setting up gcc (4:7.4.0-1ubuntu2.2) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
admin2@admin2:~$
```

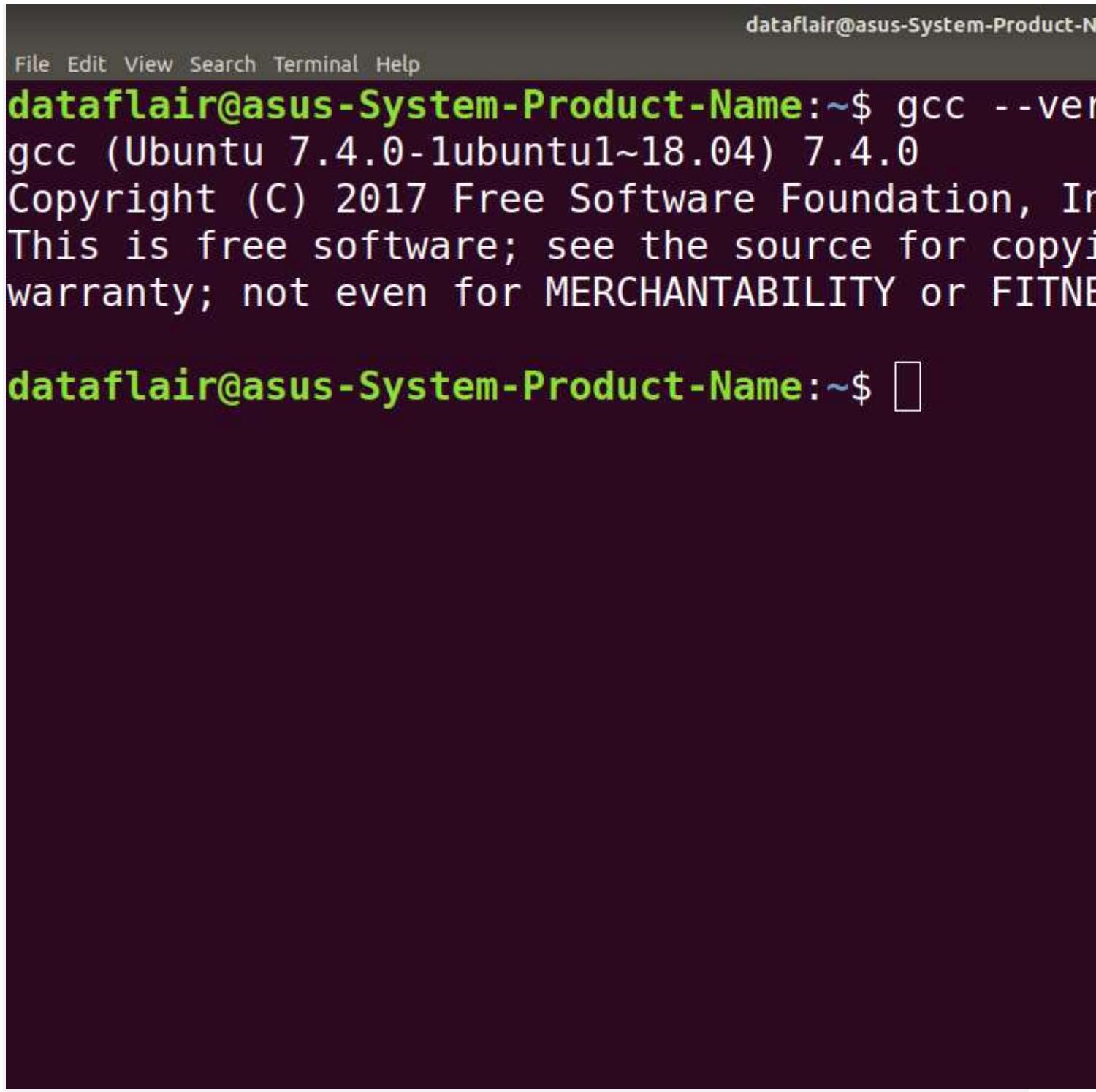
gcc installation begins

3. How to check the installed compiler version?

- You can find the terminal icon on your Desktop screen or on the search menu. A shortcut to open terminal is: Press Ctrl + Alt + T simultaneously.
- Now the terminal will open.
- The ‘gcc’ command will help you throughout your coding journey. It is a very important command that comes pre-installed on your latest Ubuntu version.
- Write the following command to know the version of the GCC compiler that you installed

GCC — version

After pressing enter, the version of the terminal software would be displayed.



The screenshot shows a terminal window with a dark background and light-colored text. At the top, it says "dataflair@asus-System-Product-Name". Below that is a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal shows the command "dataflair@asus-System-Product-Name:~\$ gcc --version" followed by its output: "gcc (Ubuntu 7.4.0-1ubuntu1~18.04) 7.4.0 Copyright (C) 2017 Free Software Foundation, Inc. This is free software; see the source for copying conditions. There is NO WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE." A small square icon is visible on the right side of the terminal window.

Displaying the version of the terminal software

This version is 7.4.0

If you don't see such a message on your display screen, then it probably means that you haven't successfully installed the GCC compiler.

4. How to create a C program?

In order to create a C program, use the ‘touch’ command and give the name of the file with .c extension

But before that, you need to choose the directory you are working on.

Most of the programmers generally prefer to work on the Desktop folder as it proves to be quite convenient to write, modify and run your programs instantly.

In order to locate your files on the Desktop folder and to change the reference of the current directory to Desktop, use this command:

`cd Desktop`

`dataflair@asus-System-Product-Name:`

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~$ cd Desktop  
dataflair@asus-System-Product-Name:~/Desktop$ █
```

Key takeaway: Commands are case sensitive

The command for creating a program in C is:

touch program.c

Now, a file has been created in our Desktop folder called program.c



The file is created on the desktop

Open this file and write a basic code – “Hello World!”

A screenshot of a code editor window. The title bar shows 'program.c' and the path '~Desktop'. The menu bar includes 'File', 'Edit', 'View', 'Tools', 'Help', and a 'Recent' section. The toolbar has icons for 'Open', 'Save', 'New', 'Run', and 'Stop'. The main area contains the following C code:

```
#include<stdio.h>
int main()
{
printf("Hello World!\n");
return 0;
}
```

Hello World in C Programming

5. How to compile and run a C program?

Now that we have created a C program called `program.c`, it is now time to compile it and run it in terminal.

To compile the code, we use the GCC command:

GCC program.c -o program

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

dataflair@asus-System-Product-Name:~\$ cd Desktop

dataflair@asus-System-Product-Name:~/Desktop\$ touch program.c

dataflair@asus-System-Product-Name:~/Desktop\$ gcc program.c -o program

Compile code using GCC command

Here, the ‘gcc’ command is followed by the file name with .c extension and ‘-o’ and the name of the executable file, which is, ‘program’ in this case.

Key takeaway: The executable file name can be different from the source file name
In order to run the program, use the command

./program

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~$ cd Desktop
dataflair@asus-System-Product-Name:~/Desktop$ touch program.c
dataflair@asus-System-Product-Name:~/Desktop$ gcc program.c -o program
dataflair@asus-System-Product-Name:~/Desktop$ ./program
```

Run the code using ./program command

Here, the period symbol ‘.’ followed by a backslash is followed by the name of the executable file, that is, ‘program’ in this case

The program output would be:

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~$ cd Desktop
dataflair@asus-System-Product-Name:~/Desktop$ touch program.c
dataflair@asus-System-Product-Name:~/Desktop$ gcc program.c -o program
dataflair@asus-System-Product-Name:~/Desktop$ ./program
Hello World!
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Output of Program

Summary

In this way, you can install the GCC compiler using terminal when working on the LINUX OS. We inferred that it is pretty easy to install the GCC compiler in LINUX as it requires only one command. Thereafter, we saw how to check the version of the installed compiler. Then, we saw how to create a program in C with the help of certain commands. Finally, we learned how to compile and run C programs using terminal.

We hope you found this tutorial helpful for installing the GCC compiler using terminal for LINUX!

If you have any queries regarding the installation process or if you're finding any difficulties to install C in your system feel free to leave a comment below.

Why C is Popular? Explore 4 Unknown Facts of C Programming

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 15, 2019

Everyone who has heard about C knows that it is one of the oldest programming languages, but doesn't know the secret of its popularity. Do you know Why C is popular?

Here, we will get each and every detail, which makes it popular and a demanding language in the IT industry. Not only the secret is revealed but also, we will uncover some unknown truths.

Before we explore the secret, you should know the [latest career opportunities in C](#)

1. Why C is Called an Evergreen Language?

Despite several decades of existence, programmers do not regard C as an obsolete language. It continues to dominate the realm of programming. The historical relevance of C is what gives it its appeal. The first operating system ([UNIX](#)) to be developed with the help of a programming language was done using the C programming language. Here are some of the reasons why C is known as an evergreen language and why it didn't fade away with time which will eventually provide one of the reasons for you to learn C.

1. The C programming language is simple to understand. Technically, C is a mid-level language which means that it is in close association with both low-level and high-level languages. The high-level language characteristic makes it easily relatable to the human language with easy English words and phrases.
Fun fact: Sanskrit is regarded as the most suitable language for programming as each and every word in Sanskrit connotes a logical meaning to the compiler.
2. The [**most important feature of C**](#) is the implementation of various data types, loops, arrays, functions, structures, unions, macros, user-defined operations, linked lists, stacks and queues, binary trees, hash tables, and pointers.
3. C serves as the base language or you can say, a prerequisite for learning other programming languages. Most of the popularly used programming languages came after the birth of C.
4. C is a structured (modular) programming language that allows the programmer to break his codes into smaller fragments to improve the readability of the code and hence make the program less redundant and simple.

5. The portability feature of C also plays an important role in making it an evergreen language. Who wants to study a language where you cannot use the same piece of code on different platforms? C comes with this unique feature that makes our work much easier. Suppose you are a Windows user but for some apparent reasons you want to switch to Linux. Now, the programs that you have written and executed on let's say, CodeBlocks will work and give the same output when copied and pasted in a terminal while using LINUX.
6. The [C standard library](#) provides you a remarkable range of inbuilt functions that eases out things for the programmer.
7. The C programming language helps you to manage memory efficiently as it supports the feature of dynamic memory allocation and bit fields.

2. Why C is known as the Mother Language?

C is one of the oldest and most popular programming languages that is very important to learn in order to step foot in the programming world. The knowledge of programming would be incomplete if you know various other languages except for C.

Only after the advent of C, other programming languages came into the big picture.

We convey the core concepts of programming with the help of looping, arrays, conditional statements, functions, pointers, file handling, and [structures in C](#). Learning to implement these concepts helps you develop a base for programming to solve complex problems in the long run.

3. Why it is Still Used in Industries?

Use of the [C programming language widely finds applications](#) in industries because of the following reasons:

1. Software engineers still use C in embedded systems and compiler design. C is in close association with the machine language that is easily understood by the compiler. Hence, the C language serves as a communication channel between the compiler and the operating system.
2. Microsoft Windows and different types of Android applications are scripted in C.
3. Programmers use it for designing the GUI (Graphical User Interface). The Adobe Photoshop editor came into existence with the help of C. Similarly, other photo editors can be developed with the help of the C programming language.
4. C offers the benefit to the software developer to create new programming languages.
5. The tremendous speed is one of the salient features of C that allows us to work faster and is extensively used in the gaming and animation sector. Other programming languages like Python and Java are relatively slower than C as they are interpreter based.

4. Learning C is Still Worthwhile

Like we already discussed that the *C programming language is considered a prerequisite for learning other programming languages as it builds a foundation for beginners to implement advanced concepts.*

It would hardly take a month for you to get acquainted with the concepts and [syntax of C](#).

After that, it is a lifelong journey for you to explore as much as C has to offer that comes through hard work and practice. This gives the basic answer to your question of reasons to learn C. The working hard part becomes easy to adapt as programming is an interesting concept to learn and implement.

5. Summary

Now, we got the answers to the queries we had in mind before starting off with the C programming language. We got to know why the world still considers C as an evergreen language although it has been almost five decades since it came into existence. Thereafter, we discussed the reason behind its use in industries. Finally, we concluded our discussion by stating why C is the mother of all languages.

Learn the Importance of Preprocessors in C | Is it worth or not?

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 3, 2019

Preprocessing in C is not a part of the compilation process but done just before compilation. Let's take an example. Suppose you want to go somewhere on your bike. So, going to a random place is part of your processing, whereas picking up the keys of the bike, turning on the ignition, and sitting on the bike is a part of **preprocessing**. So, this is just an example of preprocessing. Here, we are going to discuss many more interesting concepts of the preprocessors in C-

- What are preprocessors in C?
- What are the types of preprocessors?
- How do preprocessors work in C?
- List of preprocessors

1. What are Preprocessors in C?

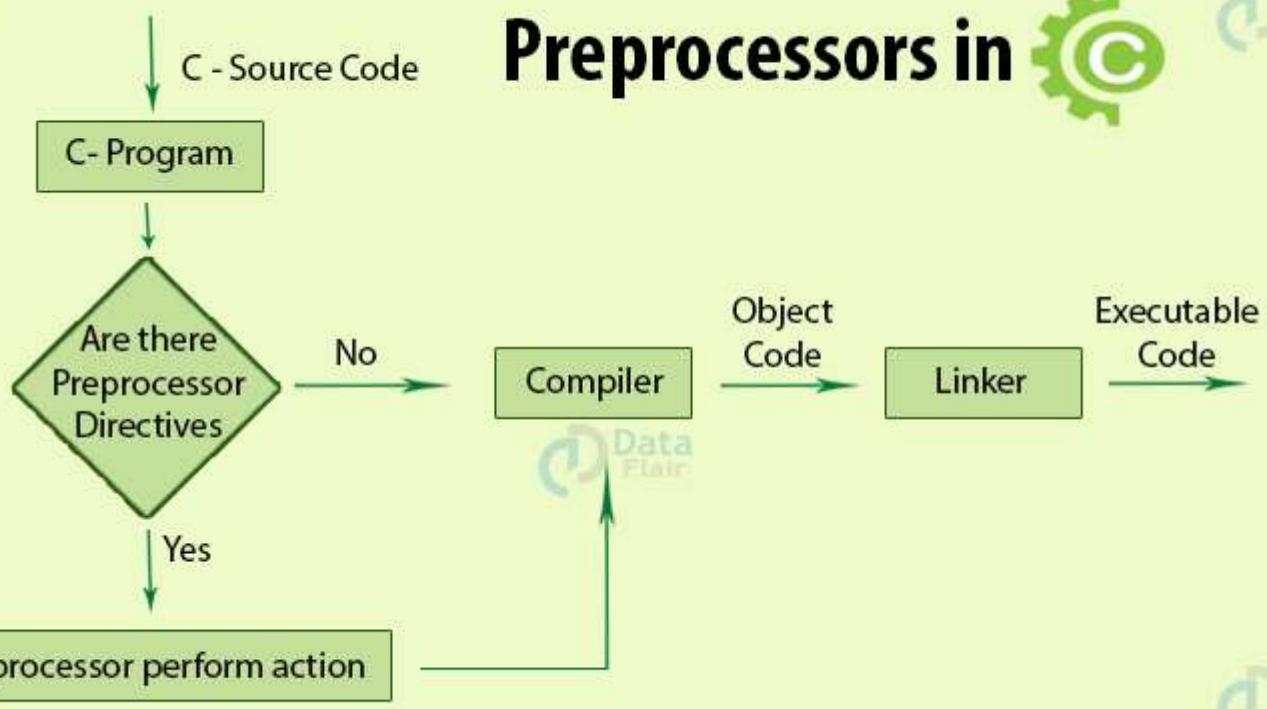
The term “preprocessor” is self-explanatory. The word “pre” means “before” and the word “processor” refers to “making something”. Before the source code is compiled, it gets automatically processed due to the presence of preprocessor directives.

In programming terminology, *a preprocessor is nothing but a System Software that performs the processing of a high-level language before compilation by translating the source code written in a high-level language to object code written in the machine-level language, that is easily understood by the compiler.*

Do you still think [Is It worth learning C for 2019?](#)

The source code is written in the form of **test.c** with .c extension where “test” is the name of the file. This file is then processed with the help of preprocessors in C expanding the source code file. After the source code file has been expanded, the next step would be to compile the code that would produce an object code file with the extension .obj and therefore would be named as **test.obj**. This object code file is then linked to the [Standard Library Functions](#) to finally generate the file with extension .exe that would be named as test.exe that can be executed.

Preprocessors in C



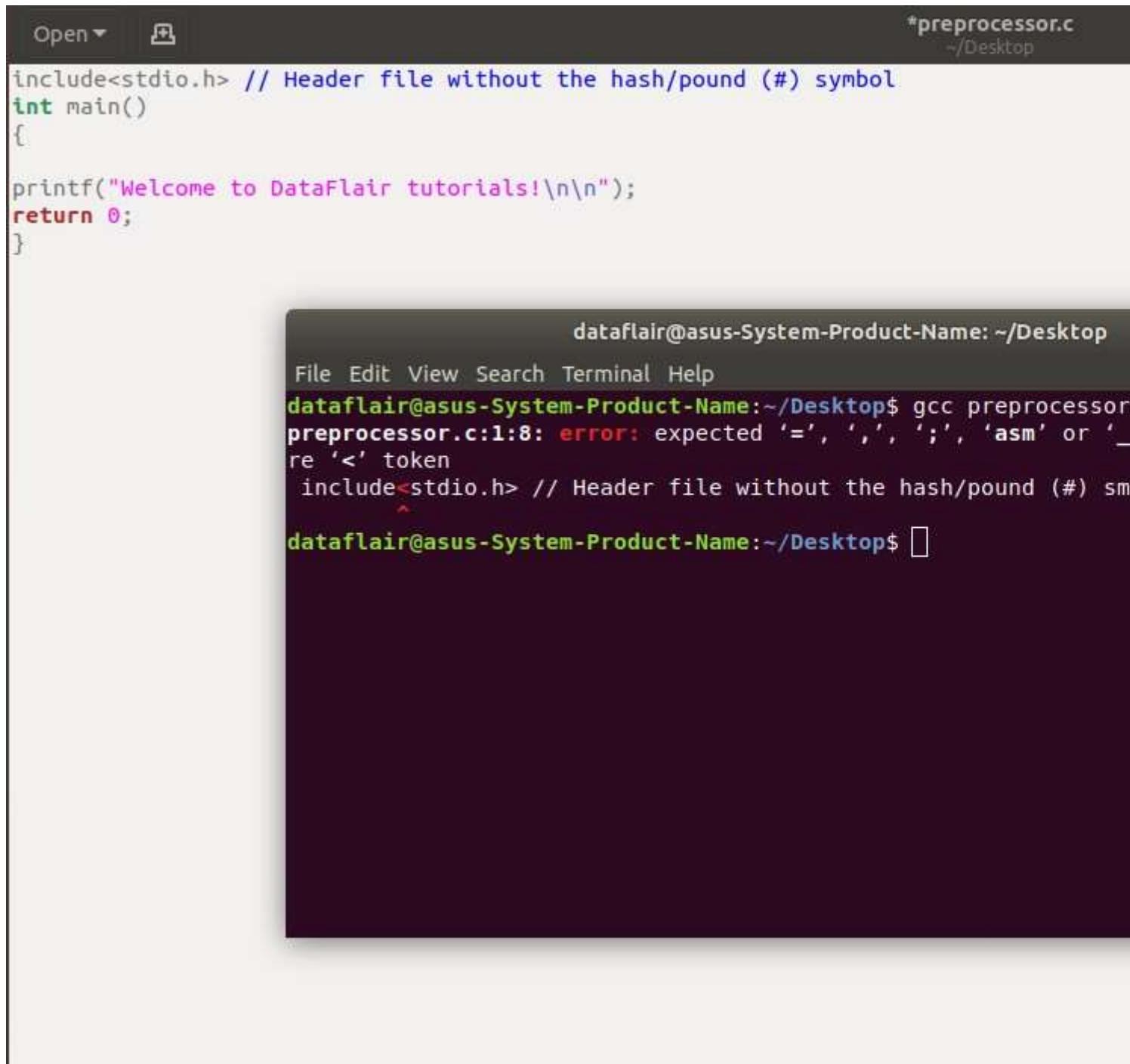
Example of Preprocessors in C

Let us acknowledge the existence of preprocessors by considering a simple example. Suppose you forgot to use the **hash/pound (#)** symbol while working with the header file `#include<stdio.h>`, will your program work?

In the C programming language, you would find an error if you skip the **# sign** while defining a header file.

1. `include<stdio.h> // Header file without the hash/pound (#) symbol`
2. `int main()`
3. `{`
4.
5. `printf("Welcome to DataFlair tutorials!\n\n");`
6. `return 0;`
7. `}`

Code on Screen-



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there's a header bar with 'Open' and a file icon on the left, and the path '*preprocessor.c ~/Desktop' on the right. Below the header, the terminal shows the following code:

```
include<stdio.h> // Header file without the hash/pound (#) symbol
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    return 0;
}
```

Then, the terminal outputs the command and its error:

```
dataflair@asus-System-Product-Name: ~/Desktop
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ gcc preprocessor
preprocessor.c:1:8: error: expected '=', ',', ';', 'asm' or 're '<' token
     include<stdio.h> // Header file without the hash/pound (#) sm
^
dataflair@asus-System-Product-Name:~/Desktop$
```

But, what is the reason behind this error? Why is the **hash/pound (#) symbol** so important? In order to answer these questions, take a quick tour of this tutorial made super easy for beginners!

2. What are Preprocessor Directives in C?

Preprocessor Directives are nothing but the commands we use in the preprocessor.

Key takeaway: In the C language, all of the preprocessor directives begin with a hash/pound (#) symbol.

We know that in C we can use preprocessor directives anywhere in our program. But, it is preferable to use them at the beginning of the program. This enhances the readability of the code.

3. Types of Preprocessors in C

There are basically 4 types of [preprocessors](#) in C.

Let's take a step forward towards increasing our understanding of preprocessors by discussing each of its types in detail:

3.1 Macros

In layman language, macros are the substitutes for strings that are used while defining a constant value.

In programming terminology, a macro is a segment of code that has the ability to provide the inclusion of header files and specifies how to map a replacement output sequence in accordance to a well-defined series of steps a particular input sequence. For example,

```
#define MAX 100
```

Here, the string MAX has the assigned constant value of 100.

Also, MAX is called macro template and 100 is repressed to as macro expansion.

Key takeaway: #define macro_template macro_expression does not terminate with a semicolon.

Read more about [Macros in C Language](#)

3.2 File Inclusion

File inclusion is responsible for instructing the compiler to include a specific file in the source code program.

Depending on the type of file included, file inclusion is categorized into two types, namely:

1. **Standard header files** – These files refer to the pre-existing files, which convey a specific meaning to the compiler before the actual compilation has taken place.
2. **User-defined files** – The C language gives the programmer the provision to define their own header files in order to divide a complex code into small fragments.

3.3 Conditional Compilation

Just like we use if-else statements for the flow of control over specific segments of code, in the same way, we use the concept of conditional compilation. Conditional compilation is a type of preprocessor that gives the programmer the power to control the compilation of particular segments of codes. It is done with the help of the 2 popular preprocessing commands, namely:

- ifdef

- endif

3.4 Other Directives

There are 2 more preprocessor directives in C apart from the ones already discussed. They are, namely:

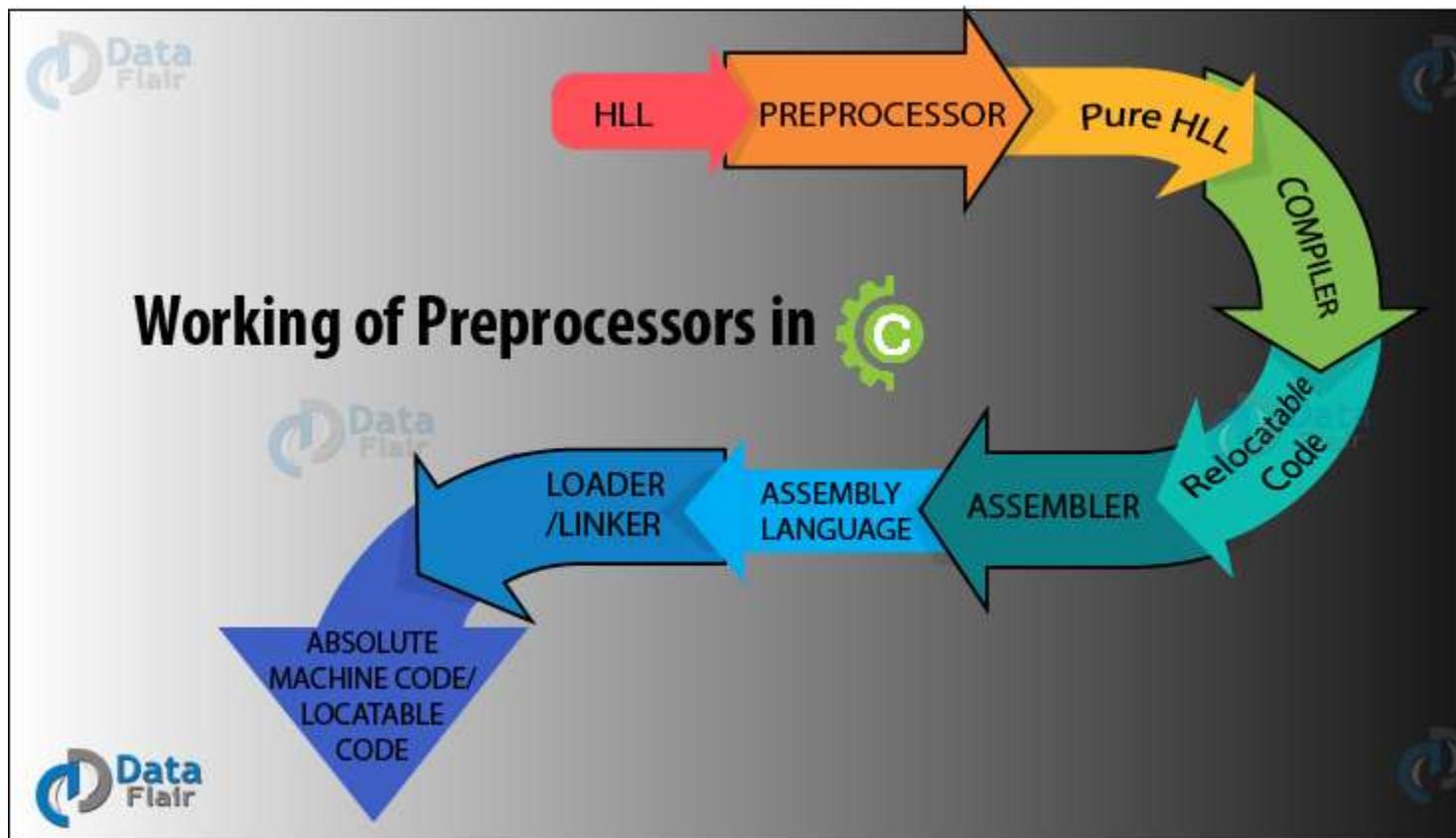
- **#undef**: As eccentric as it sounds, we use #undef directive to undefine the pre-existing, standard header or a user-defined header file.
- **#pragma**: We use this type of preprocessor directive to enable or disable certain features. It is important to note that #pragma varies from compiler to compiler.

Build your basic concepts with [Tokens in C Programming Language](#)

4. How Preprocessor Works in C?

Since you are now well-acquainted with what preprocessors are and what they do, it's time to discuss how they work.

In order to summarize the above discussion, here is a diagrammatic representation of how preprocessors work in C programming language:



The steps involved while preprocessing are as follows:

1. Filtering Out Comments

Since comments do not contribute to any logical statements in the program and are used only for the convenience of the user, they are disregarded during processing.

2. File Inclusion

It instructs the C compiler to include certain header files so that certain functions can be performed associated with them. It can be done in two ways:

- **#include<filename.h>**: The file name is enclosed between angular brackets.
- **#include“filename.h”**: The file name is enclosed within double-quotes.

3. Macro Expansion

There are certain situations where we want to use the same fragment of code in a recursive fashion. This is generally why we use macros instead of simple declarations or initializations.

There are 2 types of macros:

- **Object-like macros**: These macros are not capable of taking parameters.
- **Function-like macros**: These macros are capable of taking parameters.

Here is a table that summarizes the utility of preprocessors in C:

Preprocessor	Elucidation
#include	Used to insert a specific header from a file.
#define	Used as a replacement of a preprocessor macro.
#ifdef	Used when dealing with conditions. If the macro is defined, it returns true.
#endif	Used to close the preprocessor directive in accordance with a given condition.
#undef	Used to undefine a standard or user-defined header.
#pragma	Used to enable and disable certain features.
#ifndef	If the macro is not defined, it returns true.
#if	Used to check if the condition is true in compile time.
#else	Used to check the next condition if #if proves to be false in compile time.
#elif	Used as a combination of #else and #if.
#error	Used to print error on stderr.

Summary

Preprocessors are an important part of the C program, without it, we can't run any program. Beginners should learn this concept to master C Programming. Here, we covered details like, what are preprocessor directives, how do preprocessors work and what are its different types. We tried to understand preprocessors in C through a real-time example.

Header Files in C/C++ | Create Header Files Within Seconds

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 3, 2019

Are you aware of the various Header Files in C/C++? If not, there's no need to worry. We are going to discuss each and everything about header files in C/C++. Each program requires at least one header file to work.

Before we begin our discussion, it is important to understand, what are preprocessor directives? These are the basic building blocks of header files in C and C++. The term “**preprocessor**” is *self-explanatory*. The word “pre” means “before” and the word “processor” means “to make something”. Before the source code is compiled, it gets automatically processed due to the presence of preprocessor directives.

So, let's start and explore the depth of header files of C/C++, which will be helping you out in numerous ways.

The infographic features a central title 'Header Files in C/C+' with 'in' in black and 'C/C+' in green hexagonal icons. Below the title are three blue-tinted boxes with white text:

- How Header file works?**
- How to create your own header file?**
- Different type of Header Files**

Decorative Data Flair logos are placed around the text blocks.

1. What is a Header File in C and C++?

The [C/C++ Standard Library](#) offers its users a variety of functions, one of which is header files.

In C++, all the header files may or may not end with the .h extension but in C, all the header files must necessarily begin with the.h extension.

A header file in C/C++ contains:

- Function definitions
- Data type definitions
- Macros

Header files offer these features by importing them into your program with the help of a preprocessor directive called **#include**. These preprocessor directives are responsible for instructing the C/C++ compiler that these files need to be processed before compilation.

Every C program should necessarily contain the header file `<stdio.h>` which stands for standard input and output used to take input with the help of **scanf() function** and display the output using **printf() function**.

C++ program should necessarily contain the header file `<iostream>` which stands for input and output stream used to take input with the help of “`cin>>`” **function** and display the output using “`cout<<`” **function**.

From this example, it is clear that each header file of C and C++ has its own specific function associated with it.

Basically, header files are of 2 types:

1. **Standard library header files:** These are the pre-existing header files already available in the C/C++ compiler.
2. **User-defined header files:** Header files starting `#define` can be designed by the user.

Don't forget to check- [Basic structure of C Programming](#)

2. Syntax of Header File in C/C++

We can define the syntax of the header file in 2 ways:

- **#include<filename.h>**

The name of the header file is enclosed within angular brackets. It is the most common way of defining a header file. As discussed earlier, in C, all header files would compulsorily begin with the .h extension otherwise, you would get a compilation error but it is not the case in C++.

For example,

```
#include<string.h> // Supported both in C and C++
```

And,

```
#include<vector> // An exclusive feature of C++
```

- **#include“filename.h” or #include “filename”**

The name of the header file is enclosed within double-quotes. It is usually preferred when defining user-defined header files.

For example,

```
#include "stdlib.h" // Available in both C and C++
```

And,

```
#include "iostream" // Exclusive to C++
```

Key takeaway: We cannot include the same header file in the same program twice.

Are you aware of the [Rules of Syntax in C Programming](#)

3. How Header Files Work?

The source file contains #include which is responsible for directing the C/C++ compiler that this file needs to be processed before compilation and includes all the necessary data type and function definitions.

4. How to Create your own Header File in C/C++?

Instead of writing a large and complex code, you can create your own header files and include it in the C/C++ library to use it whenever you wish as frequently as you like. It enhances code functionality and readability.

Let us understand how to create your own header file in C++ with the help of an example.

Consider a problem where you want to compute the factorial of a number. Since it not pre-defined in the standard C++ library, you can create it by yourself!

The Steps involved are-

Step – 1

Write your own code in C++ and save the file with a .h extension instead of a .cpp, because you are creating a header file, not a C++ program. The name of the file you save with **.h extension** would be the name of your header file. Suppose you named it **factorial.h**.

```
1. int factorial(int number)
2. {
3.     int iteration, factorial=1;
4.     for(iteration=1; iteration<=number; iteration++)
5.     {
6.         factorial=factorial*iteration;
7.     }
8.     return factorial;
9. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

factorial.h

```
int factorial(int number)
{
    int iteration, factorial=1;
    for(iteration=1; iteration<=number; iteration++)
    {
        factorial=factorial*iteration;
    }
    return factorial;
}
```

Step – 2

Open a fresh window and include your header file. In this case, you can write in two ways:

#include“factorial.h” – Enclosing the header file name within double quotes signifies that the header file of C and C++ is located in the present folder you are working with. It is a preferred practice to include user-defined header files in this manner.

Must Check the [Reasons Behind the Popularity of C](#)

#include<factorial.h> – Enclosing the header file name within angular brackets signifies that the header file is located in the standard folder of all other header files of C/C++.

Step – 3

After the code is written using your file with the .h extension, compile and run your program. This is a C++ program to find the factorial of a number using a self-created header file:

```
1. #include <iostream>
2. #include "factorial.h"
3. using namespace std;
4.
5. int main()
6. {
7.
8. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
9.
10.int positive_integer;
11.cout<<"Enter a positive integer: "<<endl;
12.cin>>positive_integer;
13.cout<<"The factorial of " << positive_integer << " is: " << factorial(positive_integer) << endl;
14.
15.return 0;
16. }
```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

factorial.cp

```
#include <iostream>
#include "factorial.h"
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int positive_integer;
    cout<<"Enter a positive integer: "<<endl;
    cin>>positive_integer;
    cout<<"The factorial of " << positive_integer << " is: "

    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ factorial.cpp
dataflair@asus-System-Product-Name:~/Desktop$ ./factorial
Welcome to DataFlair tutorials!
```

```
Enter a positive integer:
```

```
5
```

```
The factorial of 5 is: 120
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Key takeaway: The header file and your C/C++ program should be in the same folder.

Different Types of C/C++ Header File

Do you ever think, how many header files are there in C/C++ Programming Language?

There are many header files present in C and C++. Even we can create them according to our requirement. In order to access the [Standard Library](#) functions, certain header files in C/C++ need to be included before writing the body of the program.

C/C++ Header File

Let's have a look at these Header files in C and C++:

1. #include<stdio.h> (Standard input-output header)

Used to perform input and output operations in C like scanf() and printf().

2. #include<string.h> (String header)

Perform string manipulation operations like strlen and strcpy.

3. #include<conio.h> (Console input-output header)

Perform console input and console output operations like clrscr() to clear the screen and getch() to get the character from the keyboard.

4. #include<stdlib.h> (Standard library header)

Perform standard utility functions like dynamic memory allocation, using functions such as malloc() and calloc().

5. #include<math.h> (Math header)

Perform mathematical operations like sqrt() and pow(). To obtain the square root and the power of a number respectively.

6. #include<ctype.h>(Character type header)

Perform character type functions like isalpha() and isdigit(). To find whether the given character is an alphabet or a digit respectively.

7. #include<time.h>(Time header)

Perform functions related to date and time like setdate() and getdate(). To modify the system date and get the CPU time respectively.

8. #include<assert.h> (Assertion header)

It is used in program assertion functions like assert(). To get an integer *data type in C/C++* as a parameter which prints stderr only if the parameter passed is 0.

9. #include<locale.h> (Localization header)

Perform localization functions like setlocale() and localeconv(). To set locale and get locale conventions respectively.

10. #include<signal.h> (Signal header)

Perform signal handling functions like signal() and raise(). To install signal handler and to raise the signal in the program respectively

11. #include<setjmp.h> (Jump header)

Perform jump functions.

12. #include<stdarg.h> (Standard argument header)

Perform standard argument functions like va_start and va_arg(). To indicate start of the variable-length argument list and to fetch the arguments from the variable-length argument list in the program respectively.

13. #include<errno.h> (Error handling header)

Used to perform error handling operations like errno(). To indicate errors in the program by initially assigning the value of this function to 0 and then later changing it to indicate errors.

Learn the [6 Types of Operators in C/C++](#) to enhance your fundamental skills

List of C++ Header File

Following are some C++ header files which are not supported in C-

1. **#include<iostream> (Input Output Stream)** – Used as a stream of Input and Output.
2. **#include<iomanip.h> (Input-Output Manipulation)** – Used to access set() and setprecision().
3. **#include<fstream.h> (File stream)** – Used to control the data to read from a file as an input and data to write into the file as an output.

Summary

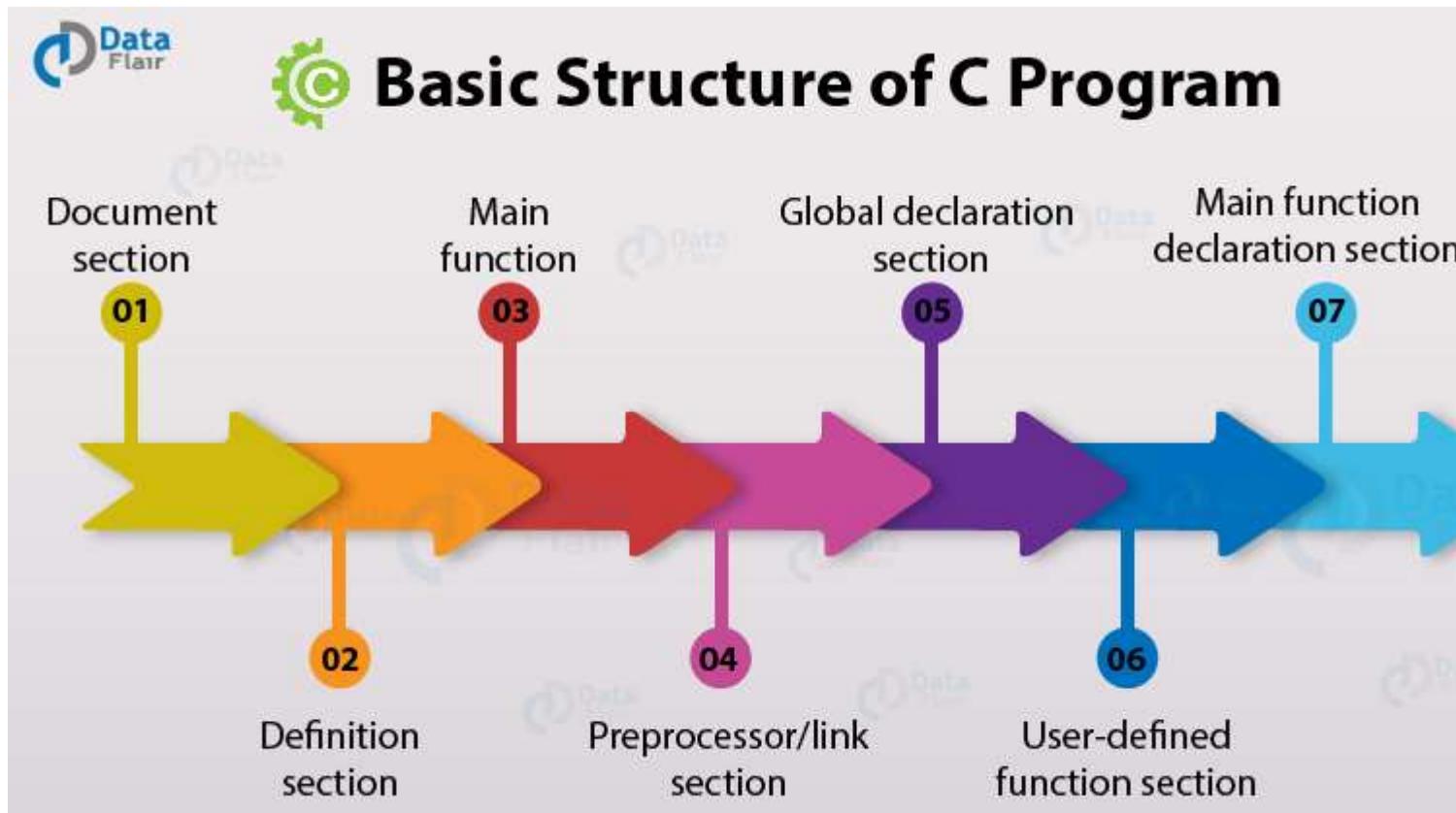
A C/C++ programmer should be well acquainted with the use of header files. In this tutorial, we got to know about the various header files available in C/C++, how they are defined, how they work and how to create a user-defined header file. We ended our discussion by summarizing the function of various kinds of header files

Learn the Basic Structure of C Program in 7 Mins

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 3, 2019

Programming in C is a difficult task for someone who is completely oblivious to the basic structure of a C program. After completing this tutorial, you would learn how the Structure of C Program looks like and soon you would be comfortable writing your own programs with ease!

So, what are you waiting for? Let's start the Basic Structure of the C program.



Anatomy of a C Program

C language has a bundle of protocols that define its programming activities.

The C programming language came into existence when its developers were working on the development of the Unix operating system using the B language, out of which C evolved. The B language lacked certain features that led to the [introduction of C](#). These features constituted the part of the C program upon which it was built.

Parts of C program-

1. `# include <stdio.h>` – This command is a [preprocessor directive in C](#) that includes all standard input-output files before compiling any C program so as to make use of all those functions in our C program.

2. **int main()** – This is the line from where the *execution of the program starts*. The main() function starts the execution of any C program.
3. **{ (Opening bracket)** – This indicates the *beginning of any function* in the program (Here it indicates the beginning of the main function).
4. **/* some comments */** – Whatever is inside /*---*/ are *not compiled and executed*; they are only written for user understanding or for making the program interactive by inserting a comment line. These are known as multiline comments. Single line comments are represented with the help of 2 forward slashes “//---”.
5. **printf(“Hello World”)** –The printf() command is included in the C stdio.h library, which helps to *display the message on the output screen*.
6. **getch()** – This command helps to *hold the screen*.
7. **return 0** –This command terminates the C program and returns a null value, that is, 0.
8. **}** (Closing brackets)- This indicates the *end of the function*. (Here it indicates the end of the main function)

Before we move towards an example, you should revise the [Syntax of C](#).

Example of C Program Structure

The “Hello World!” example is the most popular and basic program that will help you get started with programming. This program helps you display the output “Hello World” on the output screen.

With the help of this example, we can easily understand the basic structure of a C program.

```

1. #include <stdio.h>
2. int main()
3. {
4. // Our first basic program in C
5. printf("Hello World!\n\n");
6. return 0;
7. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

hello.c

```
#include <stdio.h>
int main()
{
// Our first basic program in C
printf("Hello World!\n\n");
return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ gcc hello.c -o hello  
dataflair@asus-System-Product-Name:~/Desktop$ ./hello  
Hello World!  
dataflair@asus-System-Product-Name:~/Desktop$
```

You can practice the code explained above and compile it on your machine to get an essence of C programming.

Key takeaway: \n is used to move the cursor to the new line.

Steps involved to get the desired output

Well, when you run a C program and get the output on your screen, there is a series of steps that come in the way. To get the desired output, you need to follow all the steps. These are the steps involved while writing a program in C.

1. Create
2. Compile
3. Execute or run

4. Desired output

First of all, try to code the program in the most precise manner following the protocols of C programming like,

- C is a case-sensitive programming language.
- Each line of code in C ends with a semicolon(;), except the function definition.

You can compile the code on your system once you [install the compiler](#).

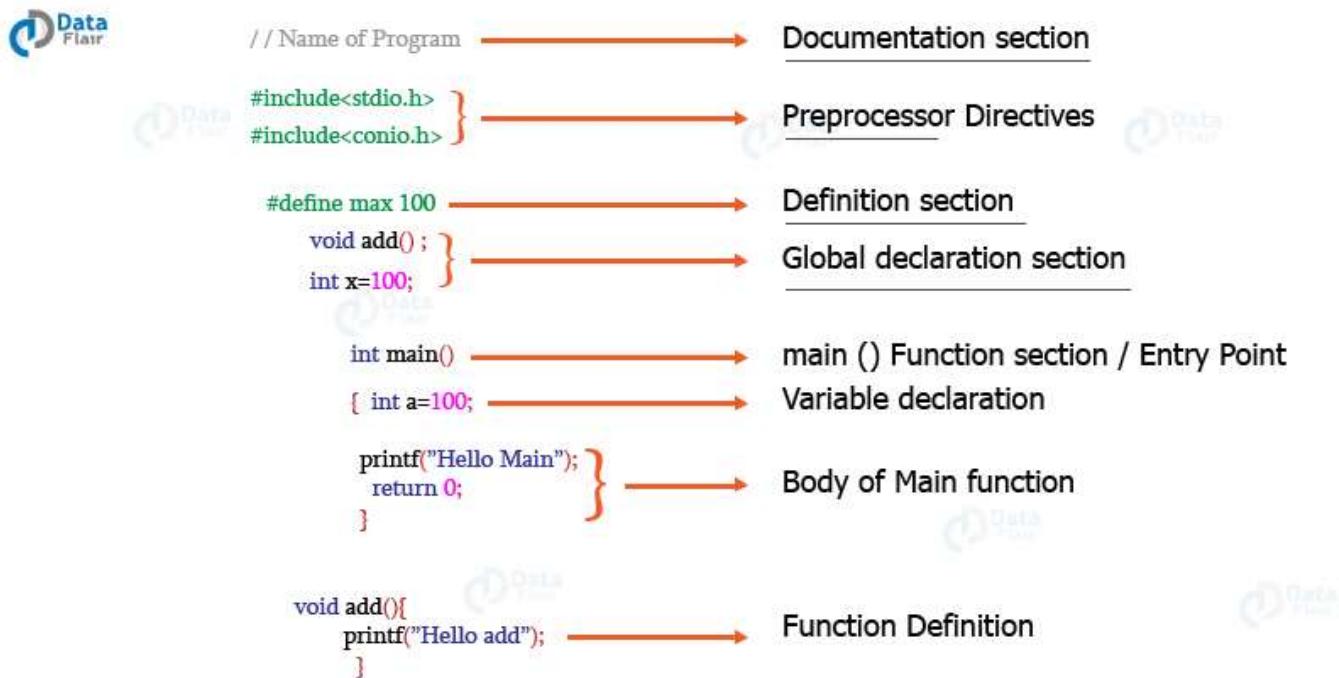
Also, if you don't want to install the compilers on your system, you can also use the online compilers available to run your code and get the output.

Once, your code gets executed you will surely get the desired output on your output screen.

Basic Structure of C Program

The components of the basic structure of a C program consists of 7 parts

1. Document section
2. Preprocessor/link Section
3. Definition section
4. Global declaration section
5. Function declaration section
6. Main function
7. User-defined function section



Still, not getting? Let's discuss every basic component of the C program with the help of an example.

```
2. Documentation section
3. C programming structure
4. Author: DataFlair
5. */
6. #include <stdio.h> /* Link section */
7. int subtract = 0; /* Global declaration, definition section */
8. int all (int, int); /* Function declaration section */
9. int main () /* Main function */
10. {
11.
12. printf("Welcome to DataFlair tutorials!\n\n");
13.
14. printf ("This is a C program \n");
15. subtract= all (25,10);
16. printf ("Subtraction of the two numbers : %d \n", subtract);
17. return 0;
18. }
19. int all (int x, int y) /* User defined function */
20. {
21. return x-y; /* definition section */
22. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

structure.c

```
/*
Documentation section
C programming structure
Author: DataFlair
*/
#include <stdio.h> /* Link section */
int subtract = 0; /* Global declaration, definition section */
int all (int, int); /* Function declaration section */
int main () /* Main function */
{
    printf("Welcome to DataFlair tutorials!\n\n");
    printf ("This is a C program \n");
    subtract= all (25,10);
    printf ("Subtraction of the two numbers : %d \n", subtract);
    return 0;
}
int all (int x, int y) /* User defined function */
{
    return x-y; /* definition section */
}
```

Output

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ touch structure.c
```

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc structure.c -o structure
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./structure
```

```
Welcome to DataFlair tutorials!
```

```
This is a C program
```

```
Subtraction of the two numbers : 15
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

1. Documentation Section

It is the section in which you can give comments to make the program more interactive. The compiler won't compile this and hence this portion would not be displayed on the output screen.

2. Preprocessor directives Section

This section involves the use of header files that are to be included necessarily in the program.

3. Definition section

This section involves the *variable definition and declaration in C*.

4. Global declaration Section

This section is used to define the global variables to be used in the programs, that means you can use these variables throughout the program.

5. Function prototype declaration section

This section gives the information about a function that includes, the data type or the return type, the parameters passed or the arguments.

6. Main function

It is the major section from where the execution of the program begins. The main section involves the declaration and executable section.

7. User-defined function section

When you want to define your function that fulfills a particular requirement, you can define them in this section.

Summary

We learned about the basic structure of a C program and also the commands necessary to build a C program. Every C program involves many sections that complete the program; we have gone through all these sections with the help of some examples.

15 Types of Escape Sequence in C that Make your Coding Better

BY [DATAFLAIR TEAM](#) · UPDATED · JULY 3, 2019

An *Escape Sequence in C* is a sequence of characters that doesn't represent itself when used inside a character or a string literal but has its own specific function.

So far we have learned the character set along with [Tokens in C](#) in detail but we haven't come across the term "Escape sequence".

In order to unveil this topic, let us begin step by step its detailed discussion. The concept of escape sequences was originally developed in the C language, and then, later on, carried to other languages like C# and Java.

What is Escape Sequence in C?

An *escape sequence* is a sequence of characters used in formatting the output and are not displayed while printing text on to the screen, each having its own specific function.

All the escape sequences in C are represented by 2 or more characters, one compulsorily being backslash (\) and the other any character present in the C character set.



Escape Sequence in C

Escape Sequence	Meaning	Elucidation
\n	New line	Used to shift the cursor control to the new line.
\t	Horizontal tab	Used to shift the cursor to a couple of spaces to the right in the same line.
\a	Audible bell	A beep is generated indicating the execution of the program to alert the user.
\r	Carriage Return	Used to position the cursor to the beginning of the current line.
\\\	Backslash	Used to display the backslash character.

Significance of Escape Sequence in C

In order to acknowledge the significance of escape sequences, let us consider a simple problem at hand where we want to display some text in a new line in order to enhance

code readability. A novice at C programming would say that this task can be ambiguously achieved by placing multiple white spaces in the **printf()** function. This is not an appropriate method to solve this problem, as it proves to be quite inconvenient and requires multiple test cases to get the output according to the user's wish. We can solve this problem by using **\n escape sequence**. Not only this, the C language offers about 15 escape sequences that allow the user to format the output on the screen.

Before we move ahead let's revise the concept of [Functions in C](#)

Here is a simple code in C which illustrates the use of **\n escape sequence**:

```
1. #include<stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome\n");
6.     printf("to\n");
7.     printf("DataFlair\n");
8.     printf("tutorials!\n");
9.     return 0;
10. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

n1.c

```
#include<stdio.h>
int main()
{
/* To illustrate the use of \n escape sequence */

printf("Welcome\n");
printf("to\n");
printf("DataFlair\n");
printf("tutorials!\n");
return 0;
}
```

Output–

```
File Edit View Search Terminal Help  
dataflair@admin4-H110M-H:~$ cd Desktop  
dataflair@admin4-H110M-H:~/Desktop$ touch n.c  
dataflair@admin4-H110M-H:~/Desktop$ gcc n.c -o n  
dataflair@admin4-H110M-H:~/Desktop$ ./n  
Welcome  
to  
DataFlair  
tutorials!  
dataflair@admin4-H110M-H:~/Desktop$ □
```

Types of Escape Sequence in C

There are 15 types of [escape sequence](#) in C to achieve various purposes.

Here is a table which illustrates the use of escape sequences in C:

1. \n (New line) – We use it to shift the cursor control to the new line
2. \t (Horizontal tab) – We use it to shift the cursor to a couple of spaces to the right in the same line.
3. \a (Audible bell) – A beep is generated indicating the execution of the program to alert the user.
4. \r (Carriage Return) – We use it to position the cursor to the beginning of the current line.
5. \\ (Backslash) – We use it to display the backslash character.

6. \' (Apostrophe or single quotation mark) – We use it to display the single-quotation mark.
7. \" (Double quotation mark)- We use it to display the double-quotation mark.
8. \0 (Null character) – We use it to represent the termination of the string.
9. \? (Question mark) – We use it to display the question mark. (?)
- 10.\nnn (Octal number)- We use it to represent an octal number.
- 11.\xhh (Hexadecimal number) – We use it to represent a hexadecimal number.
- 12.\v (Vertical tab)
- 13.\b (Backspace)
- 14.\e (Escape character)
- 15.\f (Form Feed page break)

Get the Samurai Technique to [Learn Arrays in C](#)

Here is a code in C that illustrates the commonly used escape sequences in C:

```

1. #include<stdio.h>
2. int main()
3. {
4. /* To illustrate the use of \n escape sequence */
5.
6. printf("Welcome\v");
7. printf("to\n");
8. printf("DataFlair\v");
9. printf("tutorials!\n");
10.
11. /* To illustrate the use of \n escape sequence */
12.
13. printf("Welcome\tto\tDataFlair\ttutorials!");
14.
15. /* To illustrate the use of \v escape sequence */
16.
17. printf("Welcome\vto\vDataFlair\v tutorials!");
18. return 0;
19. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

n.c

```
#include<stdio.h>
int main()
{
/* To illustrate the use of \n escape sequence */

printf("Welcome\n");
printf("to\n");
printf("DataFlair\n");
printf("tutorials!\n");

/* To illustrate the use of \n escape sequence */

printf("Welcome\tto\tDataFlair\ttutorials!");

/* To illustrate the use of \v escape sequence */

printf("Welcome\vto\vDataFlair\vutorials!");

return 0;
}
```

Output-

```
File Edit View Search Terminal Help  
dataflair@admin4-H110M-H:~$ cd Desktop  
dataflair@admin4-H110M-H:~/Desktop$ gcc n.c -o n  
dataflair@admin4-H110M-H:~/Desktop$ ./n  
Welcome  
      to  
DataFlair  
      tutorials!  
Welcome to      DataFlair      tutorials!Welcome  
                           to  
                               DataFlair  
dataflair@admin4-H110M-H:~/Desktop$ □
```

Summary

In this tutorial, we discussed the important escape sequences in C along with their significance in the C programming language. Further, we discussed the different types of escape sequences in detail and an illustrative program that helped us understand its use.

Tokens in C – An Awesome Concept you can't Afford to Miss Out

BY [DATAFLAIR TEAM](#) · UPDATED · MAY 29, 2019

As we can't construct a sentence without the use of words, similarly, we can't construct a program without using building blocks. The smallest individual element in a program is a token. Without tokens, programming can't be done in C. Generally, tokens in C is the basic component of creating source code.

Today, you would be well acquainted with the –

1. Keywords in C
2. Identifiers in C
3. Strings in C
4. Operators in C
5. Constant in C
6. Special Characters in C

What are Tokens in C?

Tokens in C language are the smallest possible unit of a program, that conveys a specific meaning to the compiler. It is the building blocks of a programming language.

Different Types of Tokens in C

There are 6 types of Tokens in C programming languages-

It is important to understand that these names can't be used interchangeably, hence, we will discuss each type of tokens in C in detail.

1. Keywords in C

Keywords in C language are the pre-defined & reserved words, each having its own significance and hence has a specific function associated with it. We can't simply use keywords for assigning variable names, as it would connote a totally different meaning altogether and would be erroneous. There are a total of 32 keywords offered in C.

auto	break	case	char
continue	do	default	const
double	else	enum	extern
for	if	goto	float
int	long	register	return
signed	static	sizeof	short
struct	switch	typedef	union
void	while	volatile	unsigned

Key takeaway: All the keywords in C are lowercase in nature.

Do you know how to [create your own header files in C?](#)

Apart from the above-listed keywords, there are certain supplementary words that cannot be used as variable names. They are listed in alphabetical order as follows:

1. asm

- 2. bool
- 3. catch
- 4. class
- 5. const_cast
- 6. delete
- 7. dynamic_cast
- 8. explicit
- 9. export
- 10. false
- 11. friend
- 12. inline
- 13. mutable
- 14. namespace
- 15. new
- 16. operator
- 17. private
- 18. protected
- 19. public
- 20. reinterpret_cast
- 21. static_cast
- 22. template
- 23. this
- 24. throw
- 25. true
- 26. try
- 27. typeid
- 28. typename
- 29. using
- 30. virtual
- 31. wchar_t

2. Identifiers in C

The C programmer has the provision to give names of his own choice to variables, arrays, and functions. These are called identifiers in C. The user may use the combination of different character sets available in the C language to name an identifier but, there are certain rules to be abided by the user on his part when naming identifiers, otherwise the situation would prove to be dicey.

Rules for Identifiers in C –

- First character:** The first character of the identifier should necessarily begin with either an alphabet or an underscore. It cannot begin with a digit.
- No special characters:** C does not support the use of special characters while naming an identifier. For instance, special characters like comma or punctuation marks can't be used.
- No keywords:** The use of keywords as identifiers is strictly prohibited, as they are reserved words which we have already discussed.
- No white space:** White spaces include blank spaces, newline, carriage return, and horizontal tab, which can't be used.
- Word limit:** The identifier name can have an arbitrarily long sequence that should not exceed 31 characters, otherwise, it would be insignificant.
- Case sensitive:** Uppercase and lowercase characters are treated differently.

Here is a table which illustrates the valid use of Identifiers in C:

Identifiers Name	Valid or Invalid	Correction or Alternative, If Invalid	Elucidation, If Invalid
20th_name	Invalid	name_20	It violates Rule 1 as it begins with a digit
_age	Valid	—	—
market.bill	Invalid	market_bill	It violates Rule 2 as it contains a special character ‘.’
delete[5]	Invalid	delet[5]	It violates Rule 3 as it contains a keyword
employee[10]	Valid	—	—
customer name	Invalid	customername	It violates Rule 4 as it contains a blank space
area()	Valid	—	—

3. Constants in C

Often referred to as literals, constants, as the name itself suggests, are fixed values i.e. they cannot change their value during program run once they are defined.

Syntax of Constant in C Programming-

const data_type variable_name = value;

Take a tour to the [Basic Syntax Rules of C](#)

Various Types of Constants in C Language-

- Integer constants:** These are of the integer data type. For example, *const int value = 400;*
- Floating constants:** These are of the float data type. For example, *const float pi = 3.14;*
- Character constants:** These are of the character data type. For example, *const char gender = 'f';*

4. **String constants:** These are also of the character data type, but differ in the declaration. For example, `const char name[] = "DataFlair";`
5. **Octal constants:** The number system which consists only 8 digits, from 0 to 7 is called the octal number system. The constant octal values can be declared as, `const int oct = 040;` (It is the octal equivalent of the digit “32” in the decimal number system.)
6. **Hexadecimal constants:** The number system which consists of 16 digits, from 0 to 9 and alphabets ‘a’ to ‘f’ is called hexadecimal number system. The constant hexadecimal values can be declared as, `const int hex = 0x40;` (It is the hexadecimal equivalent of the digit 64 in the decimal number system.)

Don't struggle with [Data Types in C Programming](#)

4. Strings in C

Just like characters, strings are used to store letters and digits. *Strings in C are referred to as an array of characters. It is enclosed within double quotes, unlike characters which are stored within single quotes.* The termination of a string is represented by the null character that is ‘\0’. The size of a string is the number of individual characters it has. In C, a string can be declared in the following ways:

1. `char name[30] = "DataFlair"; // The compiler reserves 30 bytes of memory`
1. `char name[] = "DataFlair"; // The compiler reserves the required amount of memory`
1. `char name[30] = { 'D', 'a', 't', 'a', 'F', 'l', 'a', 'i', 'r', '\0' }; // How a string is represented as a set of characters.`

5. Special Symbols of C

Apart from letters and digits, there are some special characters in C, which will help you to manipulate or perform data operations. Each special symbol has a specific meaning to the C compiler.

1. [] – **Square brackets** – The opening and closing brackets of an array indicate single and multidimensional subscripts.
2. () – **Simple brackets** – Used to represent function declaration and calls, used in print statements.
3. {} – **Curly braces** – Denote the start and end of a particular fragment of code which may be functions or loops or conditional statements.
4. , – **Comma** – Separate more than one statements, like in the [declaration of different variable names in C](#).
5. # – **Hash / Pound / Preprocessor** – A preprocessor directive, utilized for denoting the use of a header file.
6. * – **Asterisk** – To declare pointers, used as an operand for multiplication.
7. ~ – **Tilde** – As a destructor to free memory.
8. . – **Period/dot** – To access a member of a structure.

6. Operators in C

Operators in C are tools or symbols, which are used to perform a specific operation on data. Operations are performed on operands. [Operators](#) can be classified into three broad categories, according to the number of operands used. Which are as follows:

1. Unary: It involves the use of one a single operand. For instance, '!' is a unary operator which operates on a single variable, say 'c' as !c which denotes its negation or complement.

2. Binary: It involves the use of 2 operands. They are further classified as:

- Arithmetic
- Relational
- Logical
- Assignment
- Bitwise
- Conditional

3. Ternary: It involves the use of 3 operands. For instance, ?: Is used in place of if-else conditions.

Get a complete overview of [operators in C with examples](#)

Summary

Tokens in C are the basic building blocks of a program. A person who has mastered these concepts is a valuable entity in the market. Different types of tokens in C allow us to have numerous functionalities to various fields. In short, we can say that every C aspirant should know the concept of tokens in the C programming language.

Structures in C – Makes Coder Life Easy

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 17, 2019

So far, we discussed the various user-defined [data types available in C](#) such as union, enumeration, and typedef. But we are yet to discuss one of the most important user-defined data type called structures in C.

C Structures are the optimal way to represent data as a whole. We use structures to overcome the drawback of arrays.

We already know that arrays in C are bound to store variables that are of similar data types. Creating a structure gives the programmer the provision to declare multiple variables of different data types treated as a single entity.

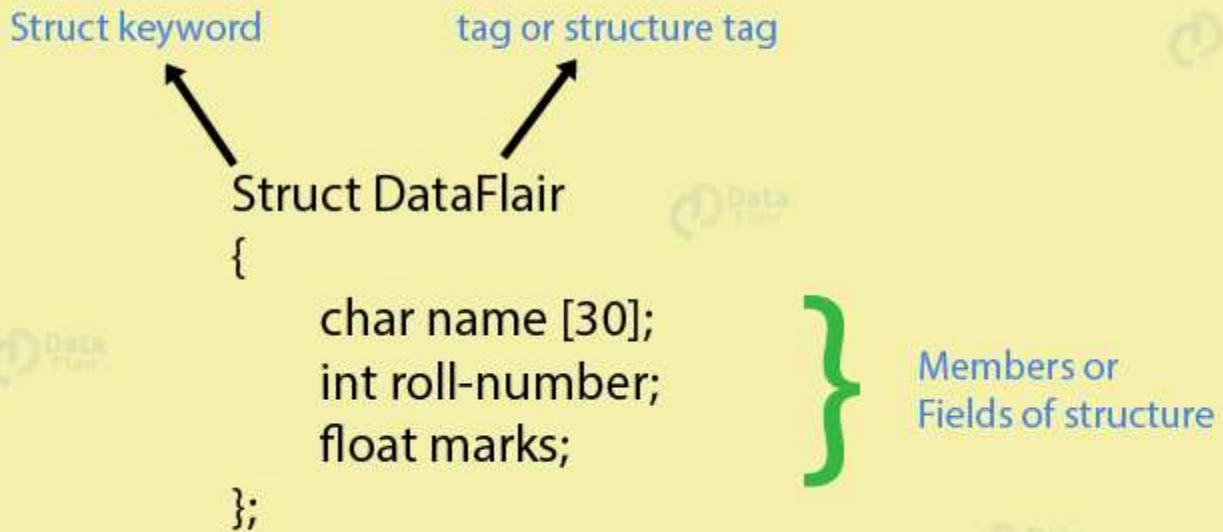
In this tutorial, we will discuss:

- Significance of Structures in C
- Create a Structure in C
- Meaning of Array of Structures
- Simplify the syntax of structure using typedef
- Nested Structures in C
- Pass structures to a function
- Access Structures using pointers
- Difference between Structure in C and C++

1. What are Structures in C?

In layman language, *a structure is nothing but a cluster of variables that may be of different data types under the name name.*

In programming terminology, *a structure is a composite data type (derived from primitive data types such as int and float) that we use in order to define a collection of similar or different data types under one same name in a particular block of computer memory.*



2. Significance of Structures in C

Now that we understood the primary purpose of structures, let us acknowledge its significance by considering a very simple problem at hand.

Suppose you want to take the input of a particular date and display it. It would probably be a lot easier if we treat the records of the structure (day, month and year) as a single unit by logically relating them with each other.

Here is a program in C that illustrates how to solve the above problem:

```
1. #include <stdio.h>
2. struct date
3. {
4.     unsigned int day, month, year;
5. };
6. int main()
7. {
8.
9.     struct date d = {12, 11, 2020}; // Here d is an object of the structure time
10.
11.    printf("Welcome to DataFlair tutorials!\n\n");
12.
13.    printf("The Date is %d / %d / %d\n", d.day, d.month, d.year);
14.
15. }
```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

date.c

```
#include <stdio.h>
struct date
{
    unsigned int day, month, year;
};
int main()
{
    struct date d = {12, 11, 2020}; // Here d is an object of the structure time
    printf("Welcome to DataFlair tutorials!\n\n");
    printf("The Date is %d / %d / %d\n", d.day, d.month, d.year);
    return 0;
}
```

Output-

dataflair@asus-System-Product-Name

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ gcc date.c -o date
dataflair@asus-System-Product-Name:~/Desktop$ ./date
Welcome to DataFlair tutorials!

The Date is 12 / 11 / 2020
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Now, let us consider another scenario where you want to store an employee's records such as his name, age, and salary under a single unit. In this case, all the 3 records are of different types but can easily be clubbed into one with the help of structures.

[Unveil the Difference between Structures and Unions in C](#)

Here is a program in C that illustrates how to solve the above problem:

```
1. #include <stdio.h>
2. struct employee
3. {
4.     char name[30];
5.     int age;
6.     float salary;
7. };
8. int main()
9. {
10.
11.     struct employee e = {"Rachel", 29, 60000}; // Here e is an object of the structure employee
```

```
12.  
13. printf("Welcome to DataFlair tutorials!\n\n");  
14.  
15. printf("The name is: %s\n",e.name);  
16. printf("The age is: %d\n",e.age);  
17. printf("The salary is: %0.2f\n",e.salary);  
18. return 0;  
19. }
```

Code-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: emp.c

```
#include <stdio.h>
struct employee
{
    char name[30];
    int age;
    float salary;
};
int main()
{
    struct employee e = {"Rachel", 29, 60000}; // Here e is an object of the structure
    printf("Welcome to DataFlair tutorials!\n\n");
    printf("The name is: %s\n",e.name);
    printf("The age is: %d\n",e.age);
    printf("The salary is: %0.2f\n",e.salary);
    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc emp.c -o emp
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./emp
```

```
Welcome to DataFlair tutorials!
```

```
The name is: Rachel
```

```
The age is: 29
```

```
The salary is: 60000.00
```

```
dataflair@asus-System-Product-Name:~/Desktop$ █
```

3. How to Create Structures in C?

A structure is created outside the main function, preferably before the main function. In order to access the data members of the structure, variable(s) are created either inside or outside the main function depending upon the choice of the programmer.

The struct keyword is used to indicate the definition of a structure.

Syntax of the body of the Structure-

```
struct structure_name
```

```
{
```

```
data_type data_member1;
```

```
data_type data_member2;
```

```

data_type data_member3;
.

data_type data_membern;
};

For instance,
struct book
{
char book_name[30];
char author[30];
int book_id;
float price;
};

```

4. How to Create Structure Variables?

As stated above, we can't directly access the data members of a structure. We need to create at least one variable to access the structure. The variable is responsible for reserving a particular block of memory for the structure according to its size.

Wait for a minute and revise [Variables in C](#)

There are 2 ways to create a variable to access data members in a structure:

1. Inside the main function:

```

struct book
{
char book_name[30];
char author[30];
int book_id;
float price;
};

int main()
{
struct book b; // Here b is a variable of structure book
}

```

2. Outside the main function:

```

struct book
{
char book_name[30];
char author[30];
int book_id;
float price;
} b; // Here b is a variable of structure book.

```

There is no difference in the way memory is allocated or how the data members are accessed through the variables in both cases.

5. How to access Data Members in Structures in C

There are basically two ways in order to access the data members in a structure through variable(s):

- With the help of the member operator/ dot symbol (.)
- With the help of structure pointer operator (->)

Suppose you want to access the data member book_id from the structure book, this is how you would do it using the member operator.

b.book_id;

For simplicity sake, we would prefer to use the dot operator in most of the cases. We will discuss the structure pointer operator in detail in the later section as it involves the use of pointers.

Let us merge together all the concepts involved in creating a structure by implementing a program in C to take input and display the details of a book using structures:

```
1. #include<stdio.h>
2. struct book
3. {
4.     char book_name[30];
5.     char author[30];
6.     int book_id;
7.     float price;
8. };
9.
10. int main()
11. {
12.
13.     struct book b; // Here b is a variable of structure book
14.
15.     printf("Welcome to DataFlair tutorials!\n\n");
16.
17.     printf("Enter the book name: ");
18.     fgets(b.book_name, 30, stdin);
19.     printf("Enter the author name: ");
20.     fgets(b.author, 30, stdin);
21.     printf("Enter the book ID: ");
22.     scanf("%d",&b.book_id);
23.     printf("Enter the book price: ");
24.     scanf("%f",&b.price);
25.
26.     printf("\nThe details of the book are:\n\n");
27.     printf("The book name is: ");
28.     puts(b.book_name);
29.     printf("The author name is: ");
30.     puts(b.author);
31.     printf("The book ID is: %d\n\n",b.book_id);
```

```
32. printf("The book price is: %0.2f\n",b.price);
33. return 0;
34. }
```

Code –

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

book.c

```
#include<stdio.h>
struct book
{
char book_name[30];
char author[30];
int book_id;
float price;
};

int main()
{
    struct book b; // Here b is a variable of structure book

    printf("Welcome to DataFlair tutorials!\n\n");

    printf("Enter the book name: ");
    fgets(b.book_name, 30, stdin);
    printf("Enter the author name: ");
    fgets(b.author, 30, stdin);
    printf("Enter the book ID: ");
    scanf("%d",&b.book_id);
    printf("Enter the book price: ");
    scanf("%f",&b.price);

    printf("\nThe details of the book are:\n\n");
    printf("The book name is: ");
    puts(b.book_name);
    printf("The author name is: ");
    puts(b.author);
    printf("The book ID is: %d\n\n",b.book_id);
    printf("The book price is: %0.2f\n",b.price);
    return 0;
}
```

Output-

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc book.c -o book
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./book
```

```
Welcome to DataFlair tutorials!
```

```
Enter the book name: Angels and Demons
```

```
Enter the author name: Dan Brown
```

```
Enter the book ID: 1001
```

```
Enter the book price: 299
```

```
The details of the book are:
```

```
The book name is: Angels and Demons
```

```
The author name is: Dan Brown
```

```
The book ID is: 1001
```

```
The book price is: 299.00
```

```
dataflair@asus-System-Product-Name:~/Desktop$ █
```

Key takeaway: It is important to note that in structures, only data members can be defined, they can't be initialized a specific value inside its definition. Also, functions cannot be declared within a structure.

Here is a code in C that illustrates what happens when we try to initialize data members and declare member functions:

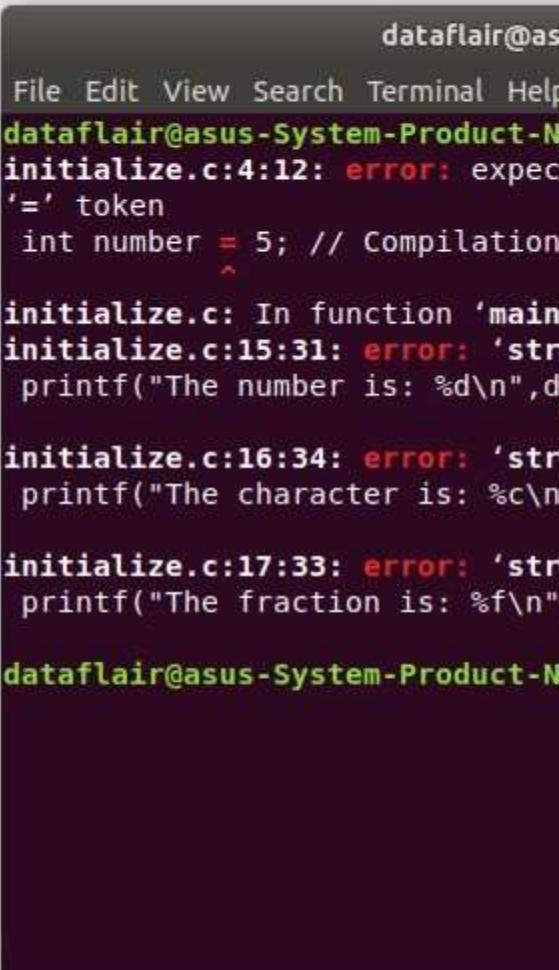
```
1. #include<stdio.h>
2. struct DataFlair
3. {
4.     int number = 5; // Compilation error
5.     char character = 'D'; // Compilation error
6.     float fraction = 33.33; // Compilation error
7. };
8.
9. int main()
10. {
11. }
```

```

12. printf("Welcome to DataFlair tutorials!\n\n");
13.
14. struct DataFlair d;
15. printf("The number is: %d\n",d.number);
16. printf("The character is: %c\n",d.character);
17. printf("The fraction is: %f\n",d.fraction);
18. return 0;
19. }

```

Code-



The terminal window shows the following output:

```

dataflair@asus-System-Product-N:~/Desktop$ initialize.c
dataflair@asus-System-Product-N:~/Desktop$ initialize.c:4:12: error: expected '=', token
int number = 5; // Compilation error
^
dataflair@asus-System-Product-N:~/Desktop$ initialize.c: In function 'main'
dataflair@asus-System-Product-N:~/Desktop$ initialize.c:15:31: error: 'str
printf("The number is: %d\n",d.
^
dataflair@asus-System-Product-N:~/Desktop$ initialize.c:16:34: error: 'str
printf("The character is: %c\n",d.
^
dataflair@asus-System-Product-N:~/Desktop$ initialize.c:17:33: error: 'str
printf("The fraction is: %f\n"
^
dataflair@asus-System-Product-N:~/Desktop$ 

```

6. The array of Structures in C

Before discussing this topic, it is important to have a crystal clear understanding of [arrays in C](#).

We already know that arrays and structures allow the grouping of elements under one roof, but if implemented together, arrays and structures can extend its functionality to another level.

In order to understand the array of structures, let us consider the base problem at hand and make slight modifications. Suppose you want to take the input and display all the records of 3 books, you can do it in two ways:

- Create 5 variables to access the data members of the structure: b1, b2, and b3.
- Create an array of structures b[3].

What sounds more convenient and easy to implement? Obviously the 2nd method. This is where the array of structures comes into play.

Here is a code in C that illustrates the implementation of the array of structures with reference to the above-stated problem:

```
1. #include<stdio.h>
2. struct book
3. {
4.     char book_name[30];
5.     char author[30];
6.     int book_id;
7.     float price;
8. };
9.
10. int main()
11. {
12.
13.     struct book b[3]; // Here b is a variable of structure book
14.     int i;
15.     printf("Welcome to DataFlair tutorials!\n\n");
16.
17.     printf("Enter the record of 5 books:\n");
18.     for(i = 0; i < 3; i++)
19.     {
20.         printf("Enter the book name: ");
21.         scanf("%s",b[i].book_name);
22.         fgets(b[i].book_name, 80, stdin);
23.         printf("Enter the author name: ");
24.         fgets(b[i].author, 80, stdin);
25.         printf("Enter the book ID: ");
26.         scanf("%d",&b[i].book_id);
27.         printf("Enter the book price: ");
28.         scanf("%f",&b[i].price);
29.         fflush(stdin);
30.     }
31.     printf("\nThe details of the book are:\n\n");
32.     for(i = 0; i < 3; i++)
33.     {
34.         printf("The book name is: ");
35.         puts(b[i].book_name);
36.         printf("The author name is: ");
37.         puts(b[i].author);
38.         printf("The book ID is: %d\n",b[i].book_id);
39.         printf("The book price is: %0.2f\n",b[i].price);
40.     }
41.     return 0;
42. }
```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

bookarray.c

```
#include<stdio.h>
struct book
{
    char book_name[30];
    char author[30];
    int book_id;
    float price;
};

int main()
{
    struct book b[3]; // Here b is a variable of structure book
    int i;
    printf("Welcome to DataFlair tutorials!\n\n");

    printf("Enter the record of 5 books:\n");
    for(i = 0; i < 3; i++)
    {
        printf("Enter the book name: ");
        scanf("%s",b[i].book_name);
        fgets(b[i].book_name, 80, stdin);
        printf("Enter the author name: ");
        fgets(b[i].author, 80, stdin);
        printf("Enter the book ID: ");
        scanf("%d",&b[i].book_id);
        printf("Enter the book price: ");
        scanf("%f",&b[i].price);
        fflush(stdin);
    }
    printf("\n\nThe details of the book are:\n\n");
    for(i = 0; i < 3; i++)
    {
        printf("The book name is: ");
        puts(b[i].book_name);
        printf("The author name is: ");
        puts(b[i].author);
        printf("The book ID is: %d\n",b[i].book_id);
        printf("The book price is: %0.2f\n",b[i].price);
    }
    return 0;
}
```

Output-

```

File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ gcc bookarray.c -o bookarray
dataflair@asus-System-Product-Name:~/Desktop$ ./bookarray
Welcome to DataFlair tutorials!

Enter the record of 5 books:
Enter the book name: A walk remember
Enter the author name: Nicholas Sparks
Enter the book ID: 1001
Enter the book price: 200
Enter the book name: Divergent
Enter the author name: Veronica Roth
Enter the book ID: 1002
Enter the book price: 350
Enter the book name: City of Bones
Enter the author name: Cassandra Clare
Enter the book ID: 1003
Enter the book price: 500

The details of the book are:

The book name is: walk remember
The author name is: Nicholas Sparks
The book ID is: 1001
The book price is: 200.00
The book name is:
The author name is: Veronica Roth
The book ID is: 1002
The book price is: 350.00
The book name is: of Bones
The author name is: Cassandra Clare
The book ID is: 1003
The book price is: 500.00
dataflair@asus-System-Product-Name:~/Desktop$ 
```

7. Typedef in C

We have already discussed how *typedef in C* helps in simplifying the syntax of structures.

8. C Nested Structures

A nested structure is basically a structure within a structure. Sometimes we might encounter problems where simply the clubbing together of primitive data types might not help, rather we would require a more complex solution.

We might require the clubbing of 2 user-defined data types, such as the array of structures or nested structures.

Since we have already discussed what array of structures are and how to implement it, let's move on towards nested structures.

Suppose we want to access the marks of a student in 3 subjects, namely physics chemistry and mathematics along with the details of the student such as his name, age, and gender.

We can solve this problem by creating two structures, one for accessing the marks in respective subjects and the other to access the student details.

The nested structure would somewhat look like this:

```
1. struct Marks
2. {
3.     float physics;
4.     float chemistry;
5.     float mathematics;
6. };
7. struct Student
8. {
9.     char name[20];
10.    int age;
11.    Marks m; // Here m is a variable for the structure Marks
12. } s; // Here s is a variable for the structure Student
```

Here, the main structure is Student within which we have declared another structure called Marks.

Here is a code in C that illustrates how a nested structure works with reference to the above-stated problem:

```
1. #include<stdio.h>
2. #include<string.h>
3. struct Marks
4. {
5.     float physics;
6.     float chemistry;
7.     float mathematics;
8. };
9. struct Student
10. {
11.     char name[20];
12.     int age;
13.     struct Marks m; // Here m is a variable for the structure Marks
14. } s; // Here s is a variable for the structure Student
15.
16. int main()
17. {
18.
19.     printf("Welcome to DataFlair tutorials!\n\n");
20.     printf("Enter student name: ");
21.     fgets(s.name, 20, stdin);
22.     printf("Enter student age: ");
23.     scanf("%d", &s.age);
```

```
24. printf("\nEnter the marks in the following subjects:\n");
25. printf("Physics: ");
26. scanf("%f",&s.m.physics);
27. printf("Chemistry: ");
28. scanf("%f",&s.m.chemistry);
29. printf("Mathematics: ");
30. scanf("%f",&s.m.mathematics);
31. printf("\nThe student details are:\n\n");
32. printf("Physics: %0.2f\n",s.m.physics);
33. printf("Chemistry: %0.2f\n",s.m.chemistry);
34. printf("Mathematics: %0.2f\n",s.m.mathematics);
35. return 0;
36. }
```

Code-

dataflair@asus-System-Product-Name: ~

File Edit View Search Terminal Help

GNU nano 2.9.3 stud.c

```
#include<stdio.h>
#include<string.h>
struct Marks
{
float physics;
float chemistry;
float mathematics;
};
struct Student
{
char name[20];
int age;
struct Marks m; // Here m is a variable for the structure Marks
}s; // Here s is a variable for the structure Student

int main()
{
printf("Welcome to DataFlair tutorials!\n\n");
printf("Enter student name: ");
fgets(s.name, 20, stdin);
printf("Enter student age: ");
scanf("%d",&s.age);
printf("\nEnter the marks in the following subjects:\n");
printf("Physics: ");
scanf("%f",&s.m.physics);
printf("Chemistry: ");
scanf("%f",&s.m.chemistry);
printf("Mathematics: ");
scanf("%f",&s.m.mathematics);
printf("\nThe student details are:\n\n");
printf("Physics: %0.2f\n",s.m.physics);
printf("Chemistry: %0.2f\n",s.m.chemistry);
printf("Mathematics: %0.2f\n",s.m.mathematics);
return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ gcc stud.c -o stud  
dataflair@asus-System-Product-Name:~/Desktop$ ./stud  
Welcome to DataFlair tutorials!  
  
Enter student name: Archie  
Enter student age: 16  
  
Enter the marks in the following subjects:  
Physics: 78.5  
Chemistry: 88.75  
Mathematics: 65  
  
The student details are:  
  
Physics: 78.50  
Chemistry: 88.75  
Mathematics: 65.00  
dataflair@asus-System-Product-Name:~/Desktop$
```

9. Array within a Structure

We can solve the same problem using the concept of arrays within a structure.

Key takeaway: It is important to note that the array of structures and array within a structure are 2 entirely different concepts.

Discover the important concept of [Multi-dimensional Arrays](#)

This is how you could have used array within a structure since marks of the 3 subjects would be of similar data type, that is, float.

1. struct Student
2. {

```

3. char name[30];
4. int age;
5. float marks[3];
6.
7. /* Index 0: physics, Index 1- chemistry, Index 2- mathematics */
8.
9. }s;
10.
11. This is how you would access the array within a structure:
12.
13. S.marks[1] // To access the marks obtained in chemistry.

```

Here is a code in C that illustrates the use of array within a structure to implement the use of array within a structure:

```

1. #include<stdio.h>
2. struct Student
3. {
4. char name[20];
5. int age;
6. float marks[3]; // Array within a structure
7. }s; // Here s is a variable for the structure Student
8.
9. int main()
10. {
11. int i;
12. printf("Welcome to DataFlair tutorials!\n\n");
13. printf("Enter student name: ");
14. fgets(s.name, 20, stdin);
15. printf("Enter student age: ");
16. scanf("%d",&s.age);
17. printf("Enter the marks in physics, chemistry and mathematics respectively: ");
18. for(i=0;i<3;i++)
19. {
20. scanf("%f",&s.marks[i]);
21. }
22. printf("\nThe student details are:\n\n");
23. puts(s.name);
24. printf("%d\n",s.age);
25. printf("The marks in physics, chemistry and mathematics:\n");
26. for(i=0;i<3;i++)
27. {
28. printf("%0.2f\n",s.marks[i]);
29. }
30. return 0;
31. }

```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

mark.c

```
#include<stdio.h>
struct Student
{
char name[20];
int age;
float marks[3]; // Array within a structure
}s; // Here s is a variable for the structure Student

int main()
{
int i;
printf("Welcome to DataFlair tutorials!\n\n");
printf("Enter student name: ");
fgets(s.name, 20, stdin);
printf("Enter student age: ");
scanf("%d",&s.age);
printf("Enter the marks in physics, chemistry and mathematics respectively: ");
for(i=0;i<3;i++)
{
scanf("%f",&s.marks[i]);
}
printf("\nThe student details are:\n\n");
puts(s.name);
printf("%d\n",s.age);
printf("The marks in physics, chemistry and mathematics:\n");
for(i=0;i<3;i++)
{
printf("%0.2f\n",s.marks[i]);
}
return 0;
}
```

Output-

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ gcc mark.c -o mark
dataflair@asus-System-Product-Name:~/Desktop$ ./mark
Welcome to DataFlair tutorials!

Enter student name: Betty
Enter student age: 16
Enter the marks in physics, chemistry and mathematics respectively: 90 92 100

The student details are:

Betty

16
The marks in physics, chemistry and mathematics:
90.00
92.00
100.00
dataflair@asus-System-Product-Name:~/Desktop$ 
```

10. Structures to a Function

Before discussing this topic, it is important to have a crystal clear understanding of *functions in C*.

We already know how to pass arguments of different data types to a function. Since a structure is nothing but a user-defined data type, it is not an alien concept to pass a structure to a function. We pass structures to a function when the structure is local to a function.

We can pass structures to a function in 2 ways:

- Call by value
- Call by reference

We will mainly discuss how to pass structures to a function using call by value and then later on briefly discuss how to do the same using call by reference.

Suppose we have a structured box with data members as its dimensions, namely, length, width and height and we wish to create a function to display the dimensions of the box, we need to pass the structures to the function as an argument.

Here is a code in C that illustrates passing a structure to a function with reference to the above-stated problem:

```
1. #include<stdio.h>
2. struct box
3. {
4.     float length;
5.     float width;
6.     float height;
7. };
8. void display(struct box b);
9. int main()
10. {
11.
12.     printf("Welcome to DataFlair tutorials!\n\n");
13.
14.     struct box b;
15.     {
16.         printf("Enter the dimensions of the box:\n\n");
17.         printf("Length: ");
18.         scanf("%f",&b.length);
19.         printf("Width: ");
20.         scanf("%f",&b.width);
21.         printf("Height: ");
22.         scanf("%f",&b.height);
23.         display(b);
24.     }
25.     return 0;
26. }
27. void display(struct box b)
28. {
29.     printf("Length: %0.2f\n", b.length);
30.     printf("Width: %0.2f\n", b.width);
31.     printf("Height: %0.2f\n", b.height);
32. }
```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

cbv.c

```
#include<stdio.h>
struct box
{
    float length;
    float width;
    float height;
};
void display(struct box b);
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    struct box b;
    {
        printf("Enter the dimensions of the box:\n\n");
        printf("Length: ");
        scanf("%f",&b.length);
        printf("Width: ");
        scanf("%f",&b.width);
        printf("Height: ");
        scanf("%f",&b.height);
        display(b);
    }
    return 0;
}
void display(struct box b)
{
    printf("Length: %0.2f\n", b.length);
    printf("Width: %0.2f\n", b.width);
    printf("Height: %0.2f\n", b.height);
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc cbv.c -o cbv
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./cbv
```

```
Welcome to DataFlair tutorials!
```

```
Enter the dimensions of the box:
```

```
Length: 5
```

```
Width: 10
```

```
Height: 15
```

```
Length: 5.00
```

```
Width: 10.00
```

```
Height: 15.00
```

```
dataflair@asus-System-Product-Name:~/Desktop$ █
```

Another point to be noted is that structures to a function can be passed in another way. In the previous method, we passed the entire structure to the function. The alternate method proves to be more flexible as it allows to pass individual data members of the structure to the function.

We can implement the above-stated problem in this manner:

```
struct box
```

```
{
```

```
float length;
```

```
float width;
```

```
float height;
```

```
}b;
```

Then,

void output(b.length, b.width,b.height)

Let us consider a problem where structures to a function are passed by the call by reference method to calculate the sum of 2 distances.

11. Structures using Pointers

Before discussing this topic, it is important to have a crystal clear understanding of pointers in C.

The C programming language gives the programmer the provision to access structures with the help of pointers.

We can access structures with the help of pointers in the following manner:

```
struct structure_name
{
    data_type data_member1;
    data_type data_member2;
    data_type data_member3;
    data_type data_membern;
};

int main()
{
    struct structure_name *p; // Here *p is a pointer to the structure structure_name
    return 0;
}
```

12. How to access Structures through Pointers?

Here is a code in C that illustrates how structures can be accessed with the help of pointers with reference to the box problem discussed earlier in the previous section.

```
1. #include <stdio.h>
2. struct box
3. {
4.     float length;
5.     float width;
6.     float height;
7. }b;
8.
9.
10. int main()
11. {
12.
13.     printf("Welcome to DataFlair tutorials!\n\n");
14.
15.     struct box *box_pointer, box1;
16.     box_pointer = &box1;
```

```

17. printf("Enter the dimensions of the box:\n\n");
18. printf("Length: ");
19. scanf("%f", &box_pointer->length);
20. printf("Width: ");
21. scanf("%f", &box_pointer->width);
22. printf("Height: ");
23. scanf("%f", &box_pointer->height);
24.
25. printf("The dimensions of the box are:\n");
26. printf("Length: %0.2f\n", box_pointer->length);
27. printf("Width: %0.2f\n", box_pointer->width);
28. printf("Height: %0.2f\n", box_pointer->height);
29. return 0;
30. }

```

Code

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

pointer.c

```

#include <stdio.h>
struct box
{
float length;
float width;
float height;
}b;

int main()
{
printf("Welcome to DataFlair tutorials!\n\n");

struct box *box_pointer, box1;
box_pointer = &box1;
printf("Enter the dimensions of the box:\n\n");
printf("Length: ");
scanf("%f", &box_pointer->length);
printf("Width: ");
scanf("%f", &box_pointer->width);
printf("Height: ");
scanf("%f", &box_pointer->height);

printf("The dimensions of the box are:\n");
printf("Length: %0.2f\n", box_pointer->length);
printf("Width: %0.2f\n", box_pointer->width);
printf("Height: %0.2f\n", box_pointer->height);
return 0;
}

```

Output-

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ gcc pointer.c -o pointer
dataflair@asus-System-Product-Name:~/Desktop$ ./pointer
Welcome to DataFlair tutorials!

Enter the dimensions of the box:

Length: 10
Width: 20
Height: 30
The dimensions of the box are:
Length: 10.00
Width: 20.00
Height: 30.00
dataflair@asus-System-Product-Name:~/Desktop$ 
```

13. Structures in C vs C++

We already know that C++ refers to C with classes. One of the salient features of C++ is that it supports [object-oriented programming](#). OOP supports the implementation of classes.

Classes are not available in C and hence it is safe to say that in C++, a structure is a class that can be declared with the help of the keyword struct.

The striking difference between a structure and a class is that a structure is public by default, that is, it can be accessed anywhere inside the entire program, whereas a class is private by default but can be changed to public or protected mode as per the user's requirement.

To be precise, a structure is a collection of only data members and a class is a collection of data members plus member functions.

In this way, we can differentiate both of them-

Structures in C

```
struct structure_name
{
    // Data member declarations
    data_type data_member1;
    data_type data_member2;
    data_type data_member3;
    .
    .
    .
    data_type data_membern;
};
```

Structures in C++

```
struct structure_name
{
    [public:] [private:] [protected:]
    /* Data members and member functions declarations */
    data_type data_member1;
    data_type data_member2;
    data_type data_member3;
    .
    .
    .
    data_type data_membern;
    return_type member_function1;
    return_type member_function2;
    return_type member_function3;
    .
    .
    .
    return_type member_functionn;
};
```

[Typecasting in C](#) – An interesting and important concept of C

14. Summary

In this tutorial, we paved the way to optimize data representation by mastering the concepts of structures by learning how to group together logically related variables. We discussed structures in detail by shedding light on its significance and the various applications using arrays, functions, and pointers.

Union in C Language – Unveil the Difference between Structures and Unions

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 14, 2019

Just like structures, union in C is a user-defined data type used to store data of different types that hold the same memory location.

In this tutorial, we will discuss the following-

- Union in C
- Difference between Structures and Unions in C
- Meaning of Pointers to a Union in C

Before we begin our discussion on unions, it is important to have a crystal clear [understanding of structures in C](#).



1. Union in C Language

As the name itself suggests, *a union refers to the grouping together of data members treated as a single entity. The keyword union is used to indicate the declaration of a union. The format to define a union is the same as that of structures.*

A union can be defined in 2 ways, just like structures. They are:

1.1 When Union variables are created inside the main function

```
union union_name
{
    data_type member1;
    data_type member2;
    data_type member3;
    .
    .
    .
    data_type memben;
};

int main()
{
    union v1, v2, *v3; // Here v1, v2 and *v3 are variables to the union
    /* Body of the main function */
    return 0;
}
```

1.2 When Union variables are created outside the main function

```
union union_name
{
    data_type member1;
    data_type member2;
    data_type member3;
    .
    .
    .
    data_type memben;
}v1, v2, *v3; // Here v1, v2 and *v3 are variables to the union
int main()
{
    /* Body of the main function */
    return 0;
}
```

Key takeaway: In order to access a data member through pointer *v3, the -> operator is used.

Don't forget to check these [6 Types of Operators in C](#) that will enhance your fundamental skills.

2. How to Implement Unions in C Programming?

Here is a code in C that illustrates the implementation of unions in a C program:

```
1. #include <stdio.h>
2. union DataFlair
3. {
4.     char value1, value2;
5. };
6. int main()
7. {
8.     printf("Welcome to DataFlair tutorials!\n\n");
9.     union DataFlair d; // Here d is a variable of union DataFlair
10.    d.value1 = 'a'; // Here d.value2 will automatically become 'a'
11.    printf("value1 = %c, value2 = %c\n",d.value1, d.value2);
12.    d.value2 = 'b'; // Here d.value1 will automatically become 'b'
13.    printf("value1 = %c, value2 = %c\n",d.value1, d.value2);
14.    return 0;
15. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

union.c

```
#include <stdio.h>
union DataFlair
{
char value1, value2;
};
int main()
{

printf("Welcome to DataFlair tutorials!\n\n");

union DataFlair d; // Here d is a variable of union DataFlair

d.value1 = 'a'; // Here d.value2 will automatically become 'a'
printf("value1 = %c, value2 = %c\n",d.value1, d.value2);

d.value2 = 'b'; // Here d.value1 will automatically become 'b'
printf("value1 = %c, value2 = %c\n",d.value1, d.value2);
return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc union.c -o union
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./union
```

```
Welcome to DataFlair tutorials!
```

```
value1 = a, value2 = a
```

```
value1 = b, value2 = b
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

3. What is the Use of Union in C Programming?

Here is another code in C that illustrates the use of [unions](#) in a C program

```
1. #include <stdio.h>
2. union employee
3. {
4.     int ID;
5.     int age;
6. };
7. int main()
8. {
9. }
```

```
10. printf("Welcome to DataFlair tutorials!\n\n");
11.
12. union employee e;
13. e.ID = 1001;
14. e.age = 22;
15.
16. printf("The ID is: %d\n",e.ID);
17. printf("The age is: %d\n",e.age);
18. return 0;
19. }
```

Evidently, it is clear that only one member can be accessed at a time in a union.

Code on Screen-

The screenshot shows a terminal window with a dark background. At the top, there is a header bar with the text "dataflair@asus-System-Product-Name". Below the header is a menu bar with "File Edit View Search Terminal Help". The main area of the terminal shows the following content:

```
File Edit View Search Terminal Help
GNU nano 2.9.3                                         union3.c

#include <stdio.h>
union employee
{
int ID;
int age;
};
int main()
{

printf("Welcome to DataFlair tutorials!\n\n");

union employee e;
e.ID = 1001;
e.age = 22;

printf("The ID is: %d\n",e.ID);
printf("The age is: %d\n",e.age);
return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc union3.c -o union3
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./union3
```

```
Welcome to DataFlair tutorials!
```

```
The ID is: 22
```

```
The age is: 22
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

4. Structures Vs Unions in C

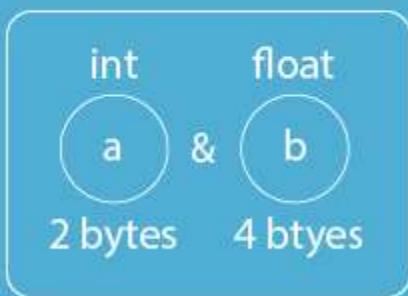
There are a lot of similarities between structures and unions in C programming language, such as they have similar syntax and serve the same purpose used to store data members of different data types.

But, there are a couple of differences between the two. One of the striking differences between the two is that the data members in a structure have individual memory locations, whereas the data members of a union share the same memory location that gives rise to its distinction.

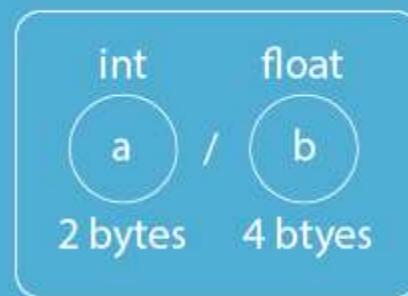
Structures

VS

Unions



Total size- 6 bytes



Total size- 4 bytes

Distinctive Feature	Structure in C	Union in C
Keyword	It is pretty obvious that a structure begins with a struct keyword.	A union begins with a union keyword.
Memory location	All the data members of a structure have their own individual memory location for their storage. Therefore, if changes are made in one data member, the other members would be independent of it.	All the data members of a union share the same memory location. Hence, changes made in one data member inevitably affects the other.
Initialization	It is possible to initialize various data members of a structure through an object at one time.	A union prohibits the initialization of all its data members. The first member of the union can only be initialized through an object.
Size	The total size of a structure is the sum of the individual sizes of its data members.	The total size of a union is the largest size of its data members.
Access	We can access all the data members of the structure at one time.	Only one data member of a union can be accessed at one time.

Here is a code in C that illustrates the difference between a structure and a union:

```

1. #include <stdio.h>
2. struct sample_structure
3. {
4.     char name[30]; // size = 30
5.     int emp_id; // size = 4
6.     float salary; // size = 4
7. };
8. /* Total size of union = Largest size of data member = 30 */
9. union sample_union
10. {
11.     char name[30]; // size = 30
12.     int emp_id; // size = 4
13.     float salary; // size = 4
14. /* Total size of structure = 30 + 4 + 4 = 38 */
15. }u;
16.
17. int main()
18. {
19.
20.     printf("Welcome to DataFlair tutorials!\n\n");
21.
22.     printf("The size of the structure is : %ld bytes\n", sizeof(s));
23.     printf("The size of the union is : %ld bytes\n", sizeof(u));

```

```
24. return 0;  
25. }
```

Code on Screen-

dataflair@asus-System-Product-Name

```
File Edit View Search Terminal Help  
GNU nano 2.9.3 union2.c  
  
#include <stdio.h>  
struct sample_structure  
{  
char name[30]; // size = 30  
int emp_id; // size = 4  
float salary; // size = 4  
};  
/* Total size of union = Largest size of data member = 30 */  
union sample_union  
{  
char name[30]; // size = 30  
int emp_id; // size = 4  
float salary; // size = 4  
/* Total size of structure = 30 + 4 + 4 = 38 */  
};  
  
int main()  
{  
  
printf("Welcome to DataFlair tutorials!\n\n");  
  
printf("The size of the structure is : %ld bytes\n", sizeof(s));  
printf("The size of the union is : %ld bytes\n", sizeof(u));  
return 0;  
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc union2.c -o union2
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./union2
```

```
Welcome to DataFlair tutorials!
```

```
The size of the structure is : 40 bytes
```

```
The size of the union is : 32 bytes
```

```
dataflair@asus-System-Product-Name:~/Desktop$ █
```

5. Pointers to Unions in C

Pointers to unions is a similar concept like pointers to structures in C.

Here is a code in C that illustrates the use of pointers to unions:

```
1. #include <stdio.h>
2. union sample
3. {
4.     int number;
5.     char character;
6. };
7.
8. int main()
```

```
9. {
10.
11. printf("Welcome to DataFlair tutorials!\n\n");
12.
13. union sample s; // Here p is a pointer
14. s.number = 122;
15. union sample *p = &s;
16.
17. printf("The ASCII value %d corresponds to %c\n", p->number, p->character); // Displaying the character corresponding
   to the ASCII value
18. return 0;
19. }
```

Code on Screen-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name:
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: upointer.c
- Code content:

```
#include <stdio.h>
union sample
{
    int number;
    char character;
};

int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    union sample s; // Here p is a pointer
    s.number = 122;
    union sample *p = &s;

    printf("The ASCII value %d corresponds to %c\n", p->number, p->character); // Displaying the character corresponding
    to the ASCII value
    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc upointer.c -o upointer
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./upointer
```

```
Welcome to DataFlair tutorials!
```

```
The ASCII value 122 corresponds to z
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

6. Summary

In this tutorial, we discussed what are unions, how to define them and various C programs associated with it. We saw the contrasting differences between a union and structure, although they seem to be quite similar to each other. Hope, you liked Union in C tutorial, but this is not the end, you may also like to know about [Macros in C](#).

Macros in C Programming – Don't Consider it as an Outdated Feature

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 8, 2019

Today, macros in C is considered as outdated in modern programming practices, it still finds applications in C by making things easier for the programmer. We will help you to develop a clear understanding of macros by covering even the minute concepts.

In this tutorial, we will discuss:

- Macros in C Programming
- Types of macros
- Different predefined macros

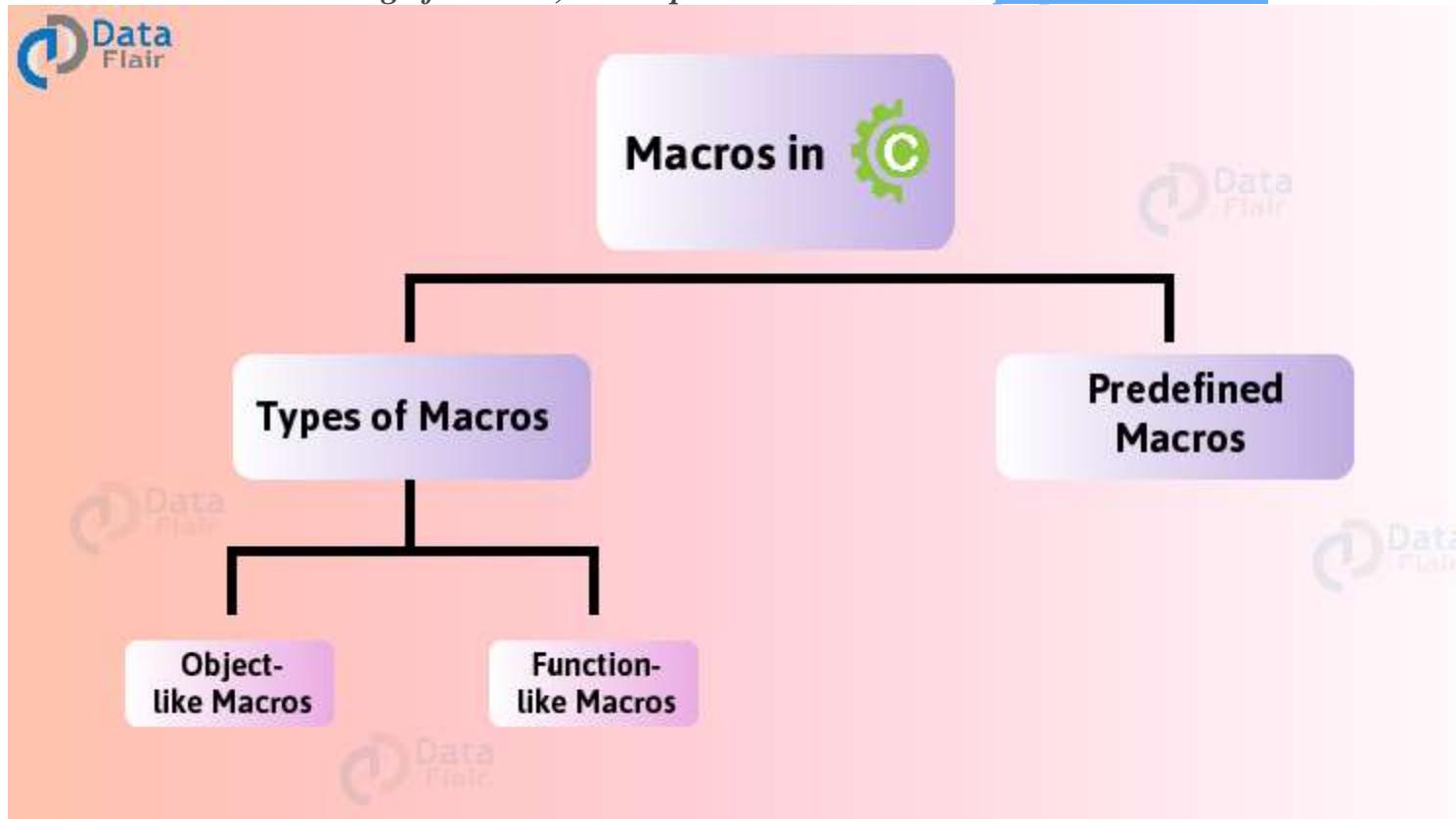
1. Macros in C Programming

Macros are nothing but a piece of code based on the #define preprocessor. In the C programming language, a macro would generally look like:

```
#define MACRO macro_value
```

Key takeaway: It is important to note that the macros are not terminated by a *semicolon* (;)

For a better understanding of Macros, it is important to understand [preprocessors in C](#).



2. Types of Macros in C Programming

In C, Macros are broadly classified into two distinct types. They are namely:

2.1 Object-like Macros

It appears to be a symbolic constant. It can be termed as an alternative way to define an identifier used to represent constant expressions. The simplest example would be:

```
#define PI 3.14
```

2.2 Function-like Macros

It is an expression, used to perform a particular operation. It is an alternative way to define a function. A simple example would be:

```
#define RECTANGLE(l,b) l*b
```

Here is a code in C that illustrates the use of macros to find the area of a rectangle:

```
1. #include<stdio.h>
2. #define RECTANGLE(l,b)l*b
3. int main()
4. {
5.
6. printf("Welcome to DataFlair tutorials!\n\n");
7.
8. int length = 3, breadth = 4;
9. int area = RECTANGLE(length,breadth);
10. printf("The area is: %d\n", area);
11. return 0;
12. }
```

Having problems in understanding the above program? Check this out [Basic Syntax Rules – Learn the ABC's of C Programming](#)

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

rec.c

```
#include<stdio.h>
#define RECTANGLE(l,b) l*b
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int length = 3, breadth = 4;
    int area = RECTANGLE(length,breadth);
    printf("The area is: %d\n", area);
    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc rec.c -o rec
dataflair@asus-System-Product-Name:~/Desktop$ ./rec
Welcome to DataFlair tutorials!
```

```
The area is: 12
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Types of Predefined Macros in C

Here is a table that summarizes some of the basic macros used in the [C](#) programming language:

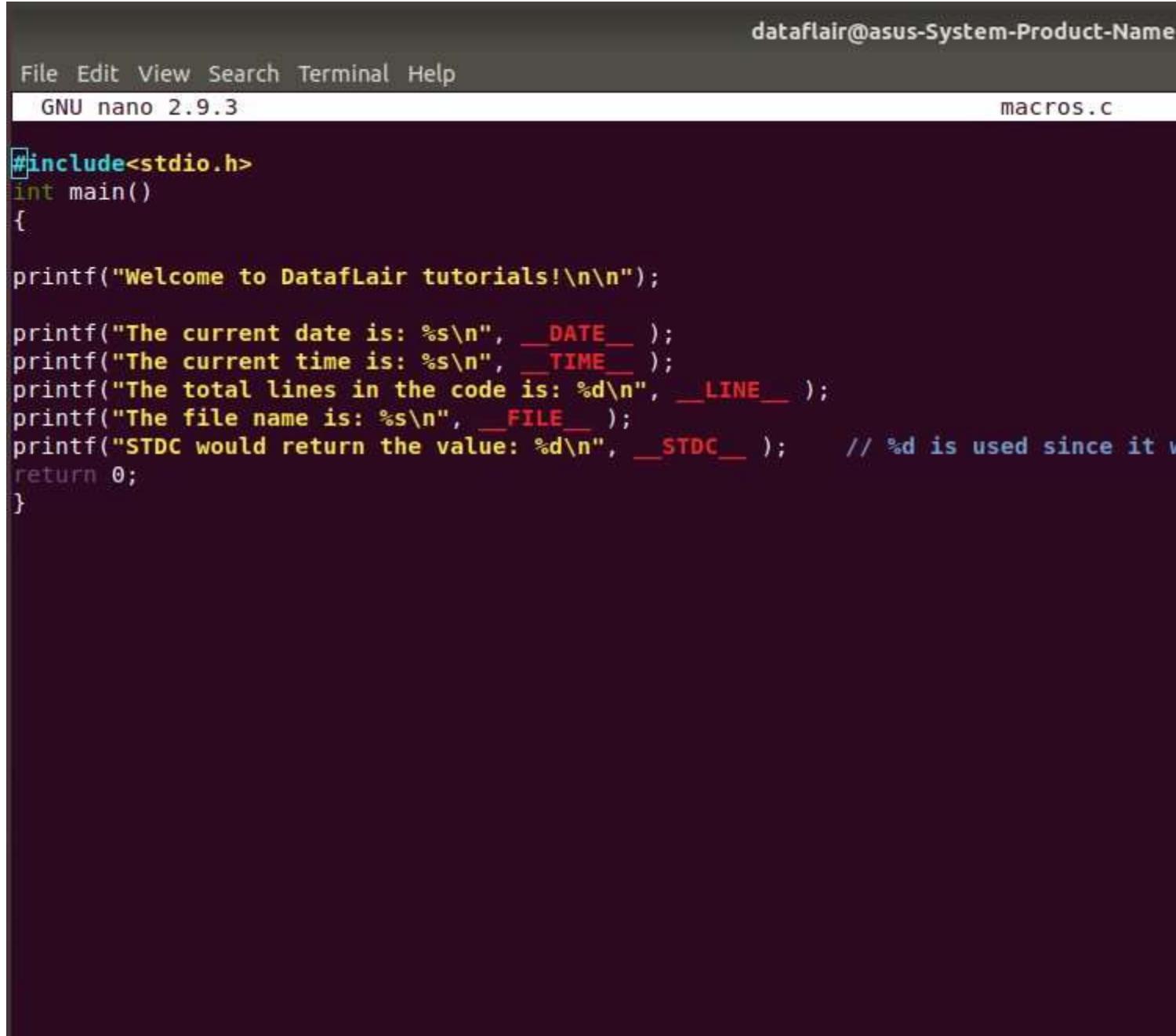
1. `_DATE_` – Used to represent the current date in MM DD YYYY format.
2. `_STDC_` – Used to indicate when the compiler compiles the code successfully by returning the value 1.
3. `_TIME_` – Represent the current time in HH:MM:SS.
4. `_LINE_` – Represent the current line number.
5. `_FILE_` – Represent the current file name.

Here is a code in C that illustrates the use of macros:

1. `#include<stdio.h>`
2. `int main()`

```
3. {
4.
5. printf("Welcome to DatafLair tutorials!\n\n");
6.
7. printf("The current date is: %s\n", __DATE__ );
8. printf("The current time is: %s\n", __TIME__ );
9. printf("The total lines in the code is: %d\n", __LINE__ );
10. printf("The file name is: %s\n", __FILE__ );
11. printf("STDC would return the value: %d\n", __STDC__ ); // %d is used since it would return an integer value
12. return 0;
13. }
```

Code on Screen-



The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: macros.c
- Code content:

```
#include<stdio.h>
int main()
{
    printf("Welcome to DatafLair tutorials!\n\n");
    printf("The current date is: %s\n", __DATE__ );
    printf("The current time is: %s\n", __TIME__ );
    printf("The total lines in the code is: %d\n", __LINE__ );
    printf("The file name is: %s\n", __FILE__ );
    printf("STDC would return the value: %d\n", __STDC__ ); // %d is used since it would return an integer value
    return 0;
}
```

Output-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

dataflair@asus-System-Product-Name:~/Desktop\$ gcc macros.c -o macros

dataflair@asus-System-Product-Name:~/Desktop\$./macros

Welcome to DataFlair tutorials!

The current date is: May 28 2019

The current time is: 15:42:07

The total lines in the code is: 10

The file name is: macros.c

STDC would return the value: 1

dataflair@asus-System-Product-Name:~/Desktop\$ █

Summary

In this tutorial, we discussed the basic meaning of macros, its 2 basic types with appropriate examples. We further carried our discussion on the various types of predefined macros available in the C programming with the help of a program.

It's the right time to explore [Typecasting and Type Conversion in C](#)

Bit Fields in C – An Unrecognised Concept Omitted by C Aspirants

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 20, 2019

Bit fields in C are relatively very simple than all the topics we have covered so far. A bit field is simply a data structure that helps the user to allocate memory to structures and unions.

In this tutorial, we will discuss:

- Need for Bit Fields
- Declaration of Bit Fields
- Working of Bit Fields

In order to understand bit fields, we need to have a clear understanding of Structures and [Unions in C](#).

Bit Fields in C Language

In programming terminology, a *bit field* is a data structure that allows the programmer to allocate memory to structures and unions in bits in order to utilize computer memory in an efficient manner.

Since structures and unions are user-defined [data types in C](#), the user has an idea of how much memory will they occupy. Accordingly, by the implementation of bit fields, memory management becomes easy and efficient.



Need for Bit Fields in C

Bit fields are of great significance in C programming, because of the following reasons:

- Used to reduce memory consumption.
- Easy to implement.
- Provides flexibility to the code.

Declaration of Bit Fields in C

A bit field is pretty easy to declare. Its declaration is as follows:

```
1. struct
2. {
3.     data_type variable_name : size_in_bits;
4. };
```

The formal name for size_in_bits is called the width of the bit field.

How do Bit Fields in C works?

In order to understand how bit fields work, let us consider a problem, in which we are expected to define a structured time to display the time according to 24-hour clock entered by the user with unsigned int hours, minutes and seconds.

Since an unsigned integer occupies 4 bytes of memory according to a 64-bit compiler, the size of the structure would be 12 bytes.

Here is a code in C that illustrates the implementation of a structured time without the use of bit fields:

```
1. #include <stdio.h>
2. struct time
3. {
4.     unsigned int hours;
5.     unsigned int minutes;
6.     unsigned int seconds;
7. };
8. int main()
9. {
10.
11.     struct time t = {11, 30, 10}; // Here t is an object of the structure time
12.     printf("Welcome to DataFlair tutorials!\n\n");
13.     printf("The time is %d : %d : %d\n", t.hours, t.minutes, t.seconds);
14.     printf("The size of time is %ld bytes.\n", sizeof(struct time));
15.     return 0;
16. }
```

Clearly, we know that, for a 24-hour clock, the range of hours should be from 0 to 23, minutes, and seconds should be from 0 to 59.

Code on Screen

File Edit View Search Terminal Help

GNU nano 2.9.3

time.c

```
#include <stdio.h>
struct time
{
    unsigned int hours;
    unsigned int minutes;
    unsigned int seconds;
};
int main()
{
    struct time t = {11, 30, 10}; // Here t is an object of the structure time
    printf("Welcome to DataFlair tutorials!\n\n");
    printf("The time is %d : %d : %d\n", t.hours, t.minutes, t.seconds);
    printf("The size of time is %ld bytes.\n", sizeof(struct time));
    return 0;
}
```

Output

```
File Edit View Search Terminal Help  
dataflair@admin4-H110M-H:~/Desktop$ gcc time.c -o time  
dataflair@admin4-H110M-H:~/Desktop$ ./time  
Welcome to DataFlair tutorials!  
  
The time is 11 : 30 : 10  
The size of time is 12 bytes.  
dataflair@admin4-H110M-H:~/Desktop$ □
```

Therefore, by the implementation of [bit fields](#), we can restrict their sizes.

- **Hours:** Since the range is from 0-23, we consider 5 bits as $2^5 = 32$ which is the nearest larger bit than the upper limit. If we consider 4 bits, then $2^4 = 16$ would be smaller than the upper limit.
- **Minutes and seconds:** Since the range is from 0-59, we consider 6 bits as $2^6 = 64$ which is the nearest larger bit than the upper limit. If we consider 5 bits, then $2^5 = 32$ would be smaller than the upper limit.

Here is a code in C that illustrates the use of bit-fields with the help of the previous example:

```
1. #include <stdio.h>  
2. struct time  
3. {  
4.     unsigned int hours: 5; // Size restricted to 5 bits  
5.     unsigned int minutes:6; // Size restricted to 6 bits  
6.     unsigned int seconds:6; // Size restricted to 6 bits
```

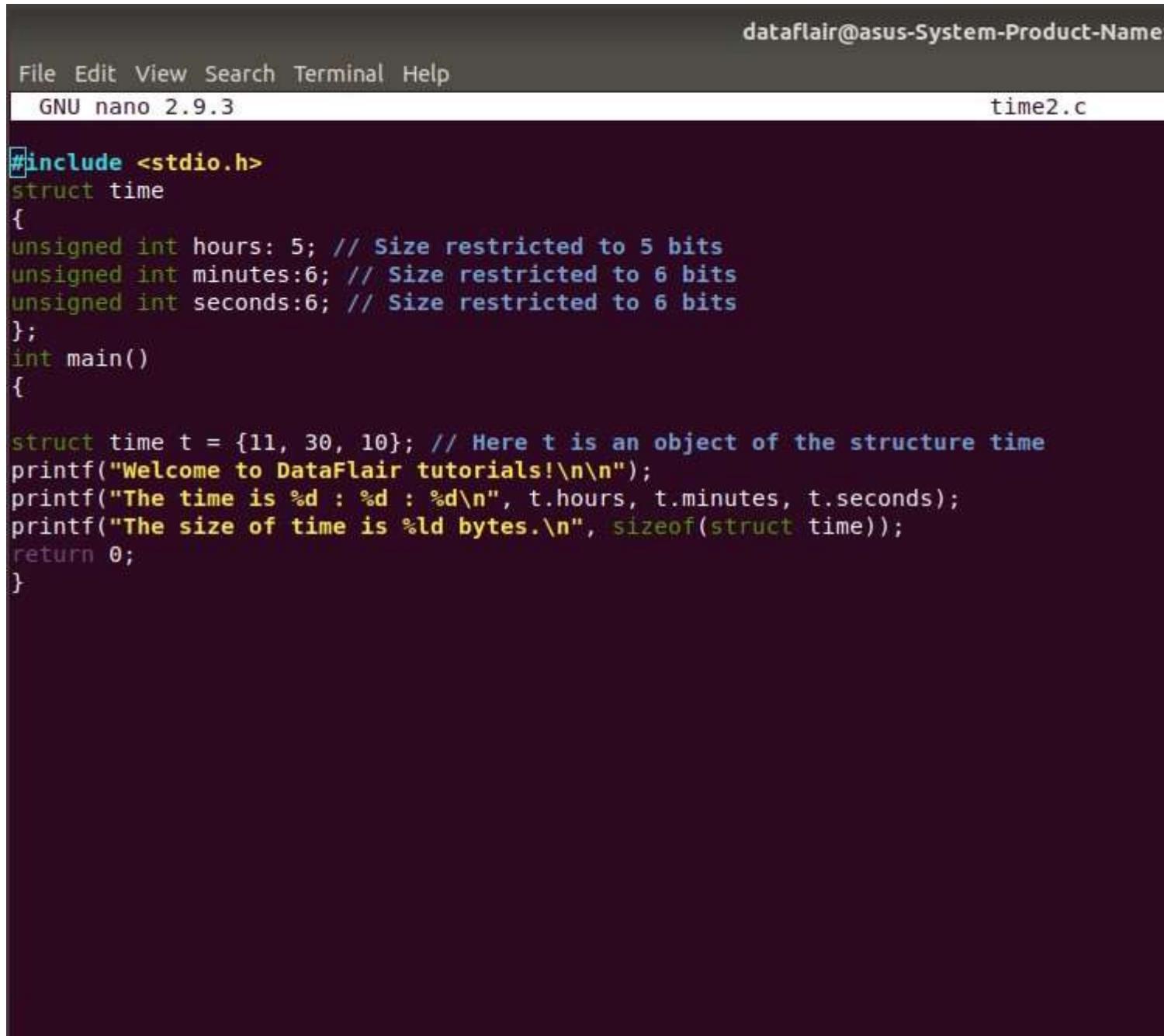
```

7.    };
8.    int main()
9.    {
10.
11.    struct time t = {11, 30, 10}; // Here t is an object of the structure time
12.    printf("Welcome to DataFlair tutorials!\n\n");
13.    printf("The time is %d : %d : %d\n", t.hours, t.minutes, t.seconds);
14.    printf("The size of time is %ld bytes.\n", sizeof(struct time));
15.    return 0;
16. }
```

It is important to note that in C bit fields cannot be declared as static, that is, the data type modifier static cannot be used. Another important thing to keep in mind is that an array of bit fields do not exist and hence can't be implemented.

If you are struggling with [Array in C](#), this is the right place to strengthen your fundamentals.

Code on Screen



The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: time2.c
- Code content:

```

#include <stdio.h>
struct time
{
    unsigned int hours: 5; // Size restricted to 5 bits
    unsigned int minutes:6; // Size restricted to 6 bits
    unsigned int seconds:6; // Size restricted to 6 bits
};
int main()
{
    struct time t = {11, 30, 10}; // Here t is an object of the structure time
    printf("Welcome to DataFlair tutorials!\n\n");
    printf("The time is %d : %d : %d\n", t.hours, t.minutes, t.seconds);
    printf("The size of time is %ld bytes.\n", sizeof(struct time));
    return 0;
}
```

Output

```
File Edit View Search Terminal Help  
dataflair@admin4-H110M-H:~$ cd Desktop  
dataflair@admin4-H110M-H:~/Desktop$ touch time2.c  
dataflair@admin4-H110M-H:~/Desktop$ gcc time2.c -o time2  
dataflair@admin4-H110M-H:~/Desktop$ ./time2  
Welcome to DataFlair tutorials!  
  
The time is 11 : 30 : 10  
The size of time is 4 bytes.  
dataflair@admin4-H110M-H:~/Desktop$ ]
```

Summary

In this tutorial, we discussed what are bit fields, its significance, and working. We have moved a step forward in a better understanding of the C programming language by completing this tutorial.

Binary Tree in C – Explore the Reason behind its Popularity

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 20, 2019

Binary Tree in C is a non-linear data structure in which the node is linked to two successor nodes, namely root, left and right. Binary trees are a very popular concept in the C programming language. But, before we begin this tutorial, it is important to have a crystal clear understanding of pointers and linked lists in C.

In this tutorial, we will discuss:

- Tree in C
- The requirement of a binary tree
- Types of binary tree
- Implementation of a binary tree

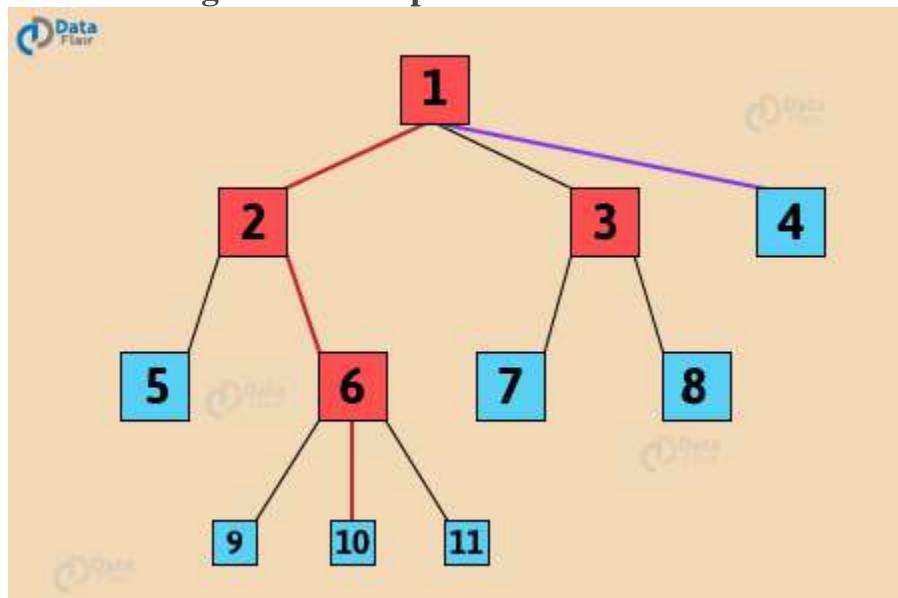
1. What are Trees in C?

In mathematical terminology, a tree is referred to as a finite and non-empty set of elements. It can also be regarded as an abstract model of a hierarchical structure that follows a parent-child relationship.

In programming terminology, a tree is nothing but a non-linear data structure that has multiple nodes, rather than just one like we saw in a **linked list, stack, and queue**.

When we talk about trees in C, we generally refer to a binary tree. We will discuss in detail the meaning of a binary tree in the later section.

Here is a diagrammatic representation of a tree:



Explore 15 types of [Escape Sequence in C](#) that makes your coding better

Before we proceed towards the discussion on a binary tree, let us acquaint ourselves with some of the tree terminologies:

1. **Root:** A node without a parent is called a root. In the above diagram, “1” is the root.
2. **Siblings:** The nodes that have the same parent are called siblings. In the above diagram “2”, “3” and “4” are siblings as they have a common parent “1”.
3. **Internal node:** A node that has at least a single child is called an internal node. In the above diagram, “1”, “2”, “3” and “6” are internal nodes.
4. **External node:** It is also called a leaf. In simple words, a node without any children is called an external node. In the above diagram, “4”, “5”, “7”, “8”, “9”, “10” and “11” are external nodes.
5. **Ancestors:** These include the parent, grandparents or great grandparents and so on of a node. In the above diagram, the ancestors of “6” are “1” and “3”.
6. **Descendants:** These include the child, grandchild or great-grandchild and so on of a node. In the above diagram, the descendants of “2” are “5”, “6”, “9”, “10” and “1”.
7. **Edge:** Connection between one node to another is called an edge. In the above diagram, the purple line from “1” to “4” is an edge.
8. **Path:** It is a sequence of nodes and edges that are connected with each other. In the above diagram, the red line from “1” to “10” is a path. It is important to note that a path simply does not only contain edges but nodes also.
9. **Depth:** It refers to the number of edges from the node to the root of the tree’s node.
10. **Height:** It refers to the maximum depth of a node.
11. **Level:** The level of a node is (the depth of the node + 1).

Key takeaway: *The height and depth of a tree are equal, but the height and depth of a node will always be different.*

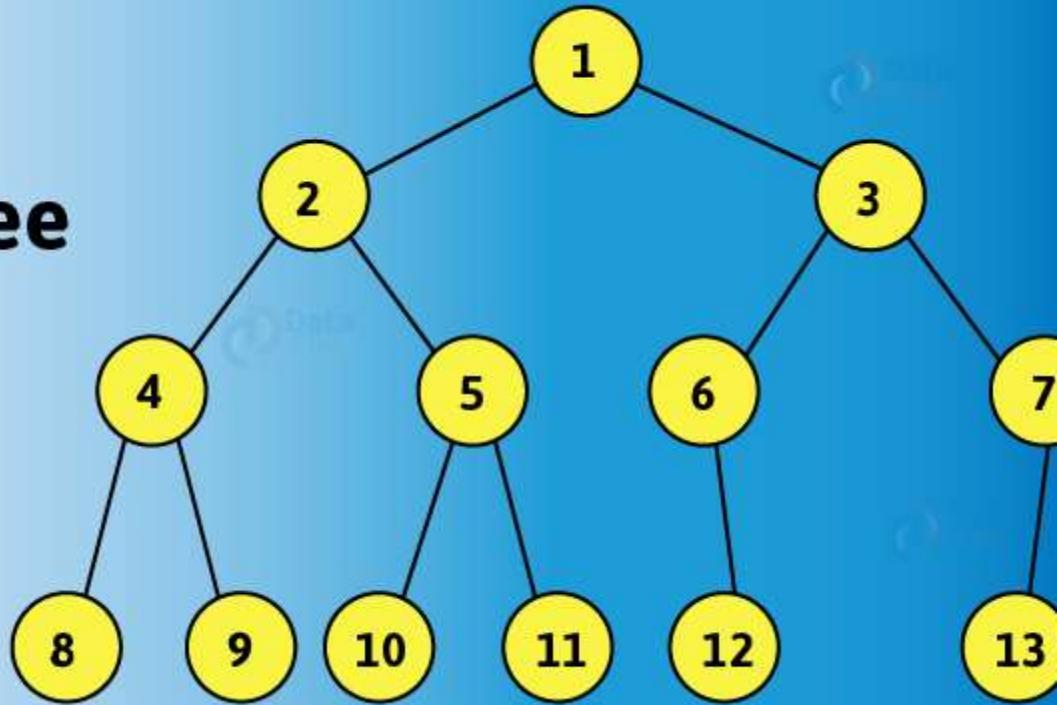
2. Binary Tree in C Programming

This tree consists of zero or more nodes. It is important to note that a binary tree can have no children (leaf node), 1 child or 2 children. No other cases are possible.

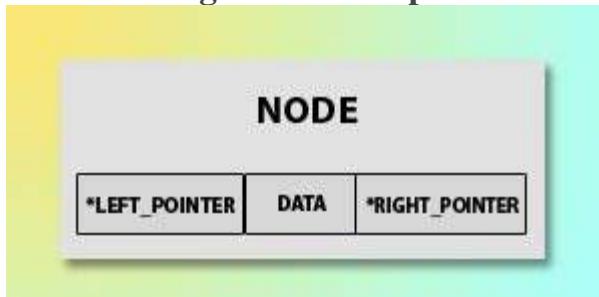
Diagrammatic representation of how a binary tree looks like:

Binary Tree

in



Here is a diagrammatic representation of how data is stored in the node of a binary tree:



Here, *left_pointer is the pointer to a left child which may or may not be NULL and *right_pointer is the pointer to a right child which may or may not be NULL.

Uncover the Difference between [Typecasting & Type Conversion in C](#)

3. Need for Binary Tree in C

This tree proves to be of great importance, which we will discuss in detail one by one.

There are several applications of a binary tree when it comes to C programming. Some of them are:

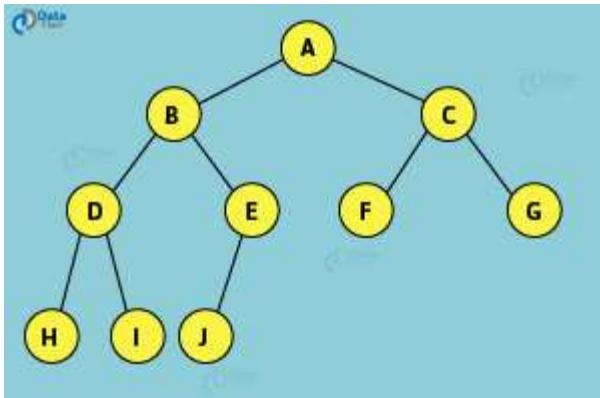
1. The implementation of BST (Binary Search Tree) is a fast and efficient method to find an element in a huge set.
2. This tree gives birth to the concept of heaps through which heapsort can be implemented.
3. In order to implement databases, a binary tree is of utmost importance as they are the base of B-Trees and T-Trees to store information.

4. Types of Binary Tree in C

There are basically two types of a C binary tree. They are namely:

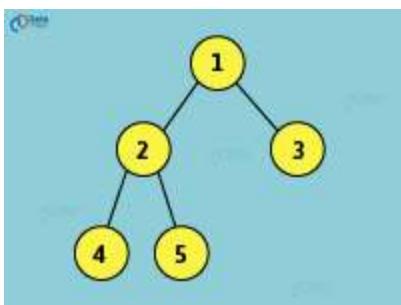
4.1 Complete binary tree

A tree in which every level except the last level is filled completely and all nodes are as far left as possible is called a complete binary tree.



4.2 Full binary tree

A tree in which every node has two children except the external node (leaves) is called a full binary tree.



5. Implementation of Binary Tree in C

In the [C](#) programming language, we can implement a binary tree in a similar fashion like linked lists. We employ the use of **structures in C**.

5.1 Declaration

Before we learn how to implement a binary tree, let us see how to declare it.

```
1. struct node
2. {
3.     int data;
4.     struct node *left;
5.     struct node *right;
6. };
```

5.2 Creating Nodes

This is how a linked list of three data elements is created in C:

```
1. struct node *root;
2. struct node *one = NULL;
3. struct node *two = NULL;
4. struct node *three = NULL;
5.
6. /* Dynamic memory allocation */
7. one = malloc(sizeof(struct node));
8. two = malloc(sizeof(struct node));
9. three = malloc(sizeof(struct node));
10.
11. /* Assign data values */
12. one->data_element = 1;
13. two->data_element = 2;
14. three->data_element = 3;
15.
16. /* Connecting nodes */
17. one->left = two;
18. one->right = three;
19. two->left = NULL;
20. two->right = NULL;
21. three->left = NULL;
22. three->right = NULL;
23.
24. /* Saving the address of the first node in the root */
25. root = one;
```

Before we discuss the implementation process, you should be aware of the [Functions in C](#).

5.3 Implementation of Binary Tree in C

Here is a program in C that illustrates the implementation of a binary tree:

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. struct node
4. {
5.     int data_element;
6.     struct node *left, *right;
7. };
8.
9. struct node *new_node(int data_element)
10. {
11.     struct node *temp = (struct node *)malloc(sizeof(struct node)); // Allocating memory to the node
12.     temp->data_element = data_element;
13.     temp->left = temp->right = NULL;
14.     return temp;
15. }
16.
17. void display(struct node *root) // A function for the inroder traversal of the binary tree
18. {
19.     if (root != NULL)
20.     {
21.         display(root->left);
22.         printf("%d \n", root->data_element);
23.         display(root->right);
24.     }
25. }
26.
27. struct node* insert(struct node* node, int data_element) // Function to insert a new node
28. {
```

```
29.  
30. if (node == NULL) return new_node(data_element); // Return a new node if the tree is empty  
31. if (data_element < node->data_element)  
32. {  
33.     node->left = insert(node->left, data_element);  
34. }  
35. else if (data_element > node->data_element)  
36. {  
37.     node->right = insert(node->right, data_element);  
38. }  
39. return node;  
40. }  
41.  
42. int main()  
43. {  
44.  
45.     printf("Welcome to DataFlair tutorials!\n\n");  
46.  
47.     struct node *root = NULL;  
48.     root = insert(root, 10);  
49.     insert(root, 20);  
50.     insert(root, 30);  
51.     insert(root, 40);  
52.     insert(root, 50);  
53.     insert(root, 60);  
54.     insert(root, 70);  
55.  
56.     display(root); // Function to display the binary tree elements  
57.  
58.     return 0;  
59. }
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ gcc bt.c -o bt  
dataflair@asus-System-Product-Name:~/Desktop$ ./bt  
Welcome to DataFlair tutorials!  
10  
20  
30  
40  
50  
60  
70  
dataflair@asus-System-Product-Name:~/Desktop$ 
```

6. Summary

In this tutorial, we discussed the basic meaning of trees in C, then we further discussed what are the binary tree and their requirement. We even discussed the two types of binary trees. Lastly, we ended our discussion by learning how to implement a binary tree from the start.

6 Types of Operators in C and C++ | Enhance Your Fundamental Skills

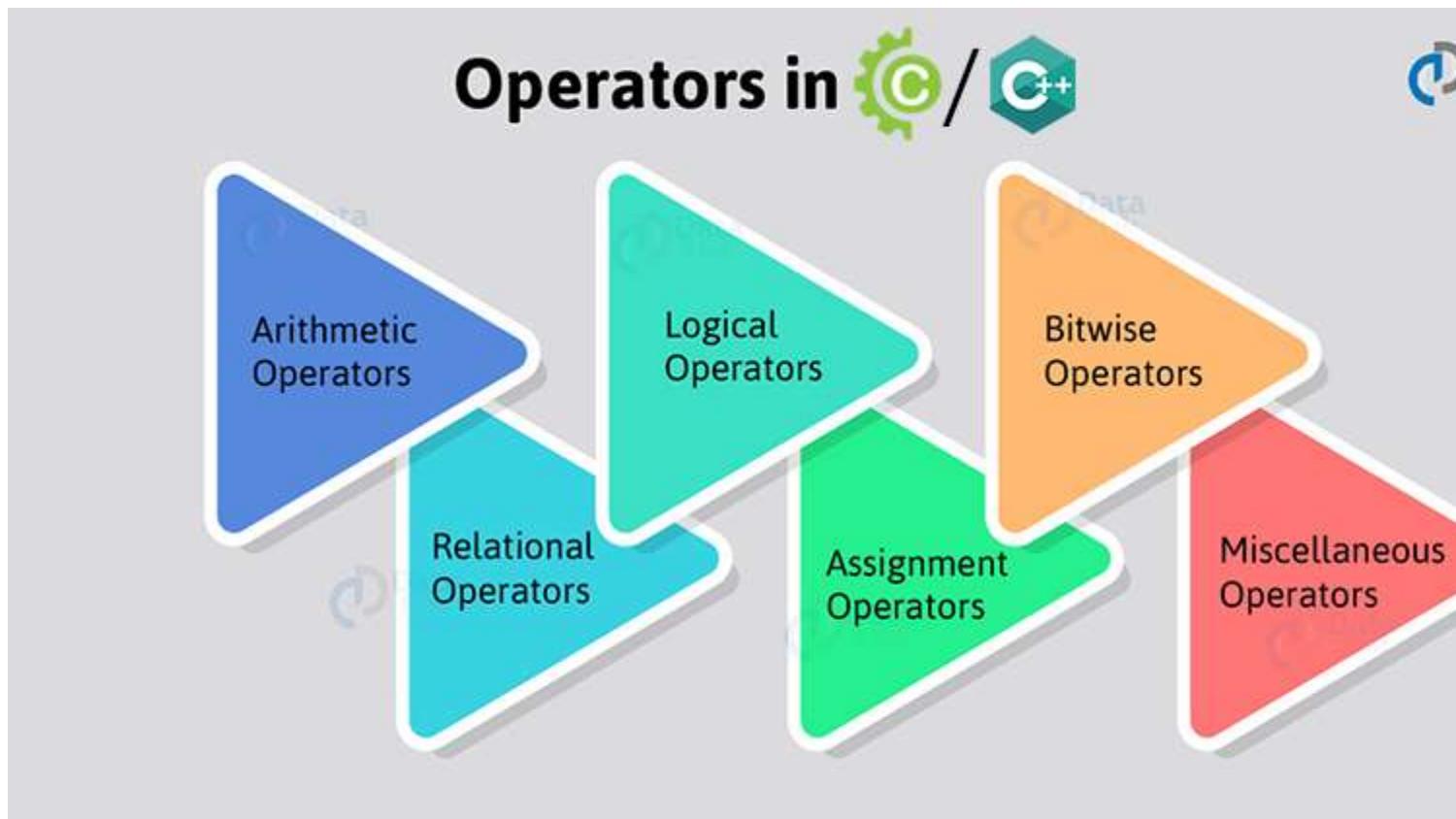
BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 27, 2019

Operators are the basic concept of any programming language, used to build a foundation in programming for freshers. Operators can be defined as basic symbols that help us work on logical and mathematical operations. Operators in C and C++, are tools or symbols that are used to perform mathematical operations concerning arithmetic, logical, conditional and, bitwise operations.

There are many sub-operators presents in each type of Operators in C/C++. Let's discuss one by one with their examples.

Types of Operators in C and C++

There are 6 types of Operators in C/C++



Let us discuss in detail the function of each type of operator.

1. Arithmetic Operators

It includes basic arithmetic operations like addition, subtraction, multiplication, division, modulus operations, increment, and decrement.

The Arithmetic Operators in C and C++ include:

1. + (**Addition**) – This operator is used to add two operands.
2. – (**Subtraction**) – Subtract two operands.
3. * (**Multiplication**) – Multiply two operands.
4. / (**Division**) – Divide two operands and gives the quotient as the answer.
5. % (**Modulus operation**) – Find the remains of two integers and gives the remainder after the division.
6. ++ (**Increment**) – Used to increment an operand.
7. — (**Decrement**) – Used to decrement an operand.

Before we move ahead, you should Master in the [Data Types in C/C++](#)

Example of Arithmetic Operators in C

```
1. #include<stdio.h>
2. int main()
3. {
4.     int a=10, b=7;
5.
6.     printf("Welcome to DataFlair tutorials!\n\n");
7.     printf("The Addition of %d and %d is: %d\n",a,b,a+b);
8.     printf("The Subtraction of %d and %d is: %d\n",a,b,a-b);
9.     printf("The Multiplication of %d and %d is: %d\n",a,b,a*b);
10.    printf("The Division of %d and %d is: %d\n",a,b,a/b);
11.    printf("The Modulus operation of %d and %d is: %d\n",a,b,a%b);
12.    printf("The Incremented value ++a is: %d\n",++a);
13.    printf("The Decrement value --b is: %d\n",--b);
14.    return 0;
15. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

arithop.c

```
#include<stdio.h>
int main()
{
int a=10, b=7;

printf("Welcome to DataFlair tutorials!\n\n");

printf("The Addition of %d and %d is: %d\n",a,b,a+b);
printf("The Subtraction of %d and %d is: %d\n",a,b,a-b);
printf("The Multiplication of %d and %d is: %d\n",a,b,a*b);
printf("The Division of %d and %d is: %d\n",a,b,a/b);
printf("The Modulus operation of %d and %d is: %d\n",a,b,a%b);
printf("The Incremented value ++a is: %d\n",++a);
printf("The Decrementated value --b is: %d\n",--b);
return 0;
}
```

The output will be-

```

File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc arithop.c -o arithop
dataflair@admin4-H110M-H:~/Desktop$ ./arithop
Welcome to DataFlair tutorials!

The Addition of 10 and 7 is: 17
The Subtraction of 10 and 7 is: 3
The Multiplication of 10 and 7 is: 70
The Division of 10 and 7 is: 1
The Modulus operation of 10 and 7 is: 3
The Incremented value ++a is: 11
The Decremented value --b is: 6
dataflair@admin4-H110M-H:~/Desktop$ 
```

[Struggling with *Structure of C Program*?](#)

Example of Arithmetic Operators in C++

Here is a code in C++ which illustrates all the basic arithmetic operators:

```

1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6. cout<<"Welcome to DataFlair tutorials!\n\n"<<endl<<endl;
7. int a = 10, b = 7;
8. cout<<"The Addition of "<< a << " and " << b << " are: " << a + b << endl;
9. cout<<"The Subtraction of "<< a << " and " << b << " are: " << a - b << endl;
10. cout<<"The Multiplication of "<< a << " and " << b << " are: " << a * b << endl;
11. cout<<"The Division of "<< a << " and " << b << " are: " << a / b << endl;
12. cout<<"The Modulus operation between "<< a << " and " << b << " is: " << a % b << endl;
13. cout<<"The Incremented value ++a is: "<< ++a << endl;
14. cout<<"The Decremented value --a is: "<< --a << endl;
15. return 0;
```

16. }

Code-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

GNU nano 2.9.3

arithop.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int a = 10, b = 7;

    cout<<"The Addition of "<< a << " and " << b << " are: " << a + b << endl;
    cout<<"The Subtraction of "<< a << " and " << b << " are: " << a - b << endl;
    cout<<"The Multiplication of "<< a << " and " << b << " are: " << a * b << endl;
    cout<<"The Division of "<< a << " and " << b << " are: " << a / b << endl;
    cout<<"The Modulus operation between "<< a << " and " << b << " is: " << a % b << endl;
    cout<<"The Incremented value ++a is: "<< ++a << endl;
    cout<<"The Decrementated value --a is: "<< --a << endl;

    return 0;
}
```

Output-

```

File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ g++ arithop.cpp -o arithop
dataflair@asus-System-Product-Name:~/Desktop$ ./arithop
Welcome to DataFlair tutorials!

The Addition of 10 and 7 are: 17
The Subtraction of 10 and 7 are: 3
The Multiplication of 10 and 7 are: 70
The Division of 10 and 7 are: 1
The Modulus operation between 10 and 7 is: 3
The Incremented value ++a is: 11
The Decremented value --a is: 10
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Table for Arithmetic Operators in C and C++

Operator	Operand	Operation	Elucidation
+	a, b	a + b	Addition
-	a, b	a - b	Subtraction
*	a, b	a * b	Multiplication
/	a, b	a / b	Division
%	a, b	a % b	Modulus operator – to find the remainder when two integral digits are divided
++	a	a ++	Increment
--	a	a --	Decrement

2. Relational Operators

It is used to compare two numbers by checking whether they are equal or not, less than, less than or equal to, greater than, greater than or equal to.

1. == (Equal to)– This operator is used to check if both operands are equal.

2. **$!=$ (Not equal to)**– Can check if both operands are not equal.
3. **$>$ (Greater than)**– Can check if the first operand is greater than the second.
4. **$<$ (Less than)**- Can check if the first operand is lesser than the second.
5. **\geq (Greater than equal to)**– Check if the first operand is greater than or equal to the second.
6. **\leq (Less than equal to)**– Check if the first operand is lesser than or equal to the second

If the relational statement is satisfied (it is true), then the program will return the value 1, otherwise, if the relational statement is not satisfied (it is false), the program will return the value 0.

It's time to gear up your skills with [Variables in C and C++](#)

Example of Relational Operators in C-

```

1. #include <stdio.h>
2. int main()
3. {
4.     int a=10, b=10, c=20;
5.
6.     printf("Welcome to DataFlair tutorials!\n\n");
7.
8.     printf("For %d == %d : The output is: %d \n", a, b, a == b); // condition is true
9.     printf("For %d == %d : The output is: %d \n", a, c, a == c); // condition is false
10.
11.    printf("For %d != %d : The output is: %d \n", a, c, a != c); // condition is true
12.    printf("For %d != %d : The output is: %d \n", a, b, a != b); // condition is false
13.
14.    printf("For %d > %d : The output is: %d \n", a, b, a > b); // condition is false
15.    printf("For %d > %d : The output is: %d \n", a, c, a > c); // condition is false
16.
17.    printf("For %d < %d : The output is: %d \n", a, b, a < b); // condition is false
18.    printf("For %d < %d : The output is: %d \n", a, c, a < c); // condition is true
19.
20.    printf("For %d >= %d : The output is: %d \n", a, b, a >= b); // condition is true
21.    printf("For %d >= %d : The output is: %d \n", a, c, a >= c); // condition is false
22.
23.    printf("For %d <= %d : The output is: %d \n", a, b, a <= b); // condition is true
24.    printf("For %d <= %d : The output is: %d \n", a, c, a <= c); // condition is true
25.    return 0;
26. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

relationop.c

```
#include <stdio.h>
int main()
{
int a=10, b=10, c=20;

printf("Welcome to DataFlair tutorials!\n\n");

printf("For %d == %d : The output is: %d \n", a, b, a == b); // condition is true
printf("For %d == %d : The output is: %d \n", a, c, a == c); // condition is false

printf("For %d != %d : The output is: %d \n", a, c, a != c); // condition is true
printf("For %d != %d : The output is: %d \n", a, b, a != b); // condition is false

printf("For %d > %d : The output is: %d \n", a, b, a > b); // condition is false
printf("For %d > %d : The output is: %d \n", a, c, a > c); // condition is false

printf("For %d < %d : The output is: %d \n", a, b, a < b); // condition is false
printf("For %d < %d : The output is: %d \n", a, c, a < c); // condition is true

printf("For %d >= %d : The output is: %d \n", a, b, a >= b); // condition is true
printf("For %d >= %d : The output is: %d \n", a, c, a >= c); // condition is false

printf("For %d <= %d : The output is: %d \n", a, b, a <= b); // condition is true
printf("For %d <= %d : The output is: %d \n", a, c, a <= c); // condition is true
return 0;
}
```

The output will be-

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc relationop.c -o relationop
dataflair@admin4-H110M-H:~/Desktop$ ./relationop
Welcome to DataFlair tutorials!

For 10 == 10 : The output is: 1
For 10 == 20 : The output is: 0
For 10 != 20 : The output is: 1
For 10 != 10 : The output is: 0
For 10 > 10 : The output is: 0
For 10 > 20 : The output is: 0
For 10 < 10 : The output is: 0
For 10 < 20 : The output is: 1
For 10 >= 10 : The output is: 1
For 10 >= 20 : The output is: 0
For 10 <= 10 : The output is: 1
For 10 <= 20 : The output is: 1
dataflair@admin4-H110M-H:~/Desktop$ □
```

Example of Relational Operators in C++

Here is a code in C++ which illustrates all the basic relational operators:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int a = 10, b = 10, c = 20;
10.
11. cout<<"For " << a << " == " << b << " The output is: " << (a == b) << endl; // condition is true
12. cout<<"For " << a << " == " << c << " The output is: " << (a == c) << endl; // condition is false
13.
14. cout<<"For " << a << " != " << c << " The output is: " << (a != c) << endl; // condition is true
15. cout<<"For " << a << " != " << b << " The output is: " << (a != b) << endl; // condition is false
16.
17. cout<<"For " << a << " > " << b << " The output is: " << (a > b) << endl; // condition is false
18. cout<<"For " << a << " > " << c << " The output is: " << (a > c) << endl; // condition is false
```

```

19.
20. cout<<"For " << a << " < " << b << " The output is: " << (a < b) << endl; // condition is false
21. cout<<"For " << a << " < " << c << " The output is: " << (a < c) << endl; // condition is true
22.
23. cout<<"For " << a << " >= " << b << " The output is: " << (a >= b) << endl; // condition is true
24. cout<<"For " << a << " >= " << c << " The output is: " << (a >= c) << endl; // condition is false
25.
26. cout<<"For " << a << " <= " << b << " The output is: " << (a <= b) << endl; // condition is true
27. cout<<"For " << a << " <= " << c << " The output is: " << (a <= c) << endl; // condition is true
28.
29. return 0;
30. }

```

Code-

The screenshot shows a terminal window with the following details:

- Title Bar:** dataflair@asus-System-Product-Name
- Menu Bar:** File Edit View Search Terminal Help
- Version:** GNU nano 2.9.3
- File Name:** relationop.cpp
- Code Content:**

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int a = 10, b = 10, c = 20;

    cout<<"For " << a << " == " << b << " The output is: " << (a == b) << endl; // condition is true
    cout<<"For " << a << " == " << c << " The output is: " << (a == c) << endl; // condition is false

    cout<<"For " << a << " != " << c << " The output is: " << (a != c) << endl; // condition is false
    cout<<"For " << a << " != " << b << " The output is: " << (a != b) << endl; // condition is true

    cout<<"For " << a << " > " << b << " The output is: " << (a > b) << endl; // condition is false
    cout<<"For " << a << " > " << c << " The output is: " << (a > c) << endl; // condition is false

    cout<<"For " << a << " < " << b << " The output is: " << (a < b) << endl; // condition is true
    cout<<"For " << a << " < " << c << " The output is: " << (a < c) << endl; // condition is true

    cout<<"For " << a << " >= " << b << " The output is: " << (a >= b) << endl; // condition is true
    cout<<"For " << a << " >= " << c << " The output is: " << (a >= c) << endl; // condition is true

    cout<<"For " << a << " <= " << b << " The output is: " << (a <= b) << endl; // condition is true
    cout<<"For " << a << " <= " << c << " The output is: " << (a <= c) << endl; // condition is true

    return 0;
}
```

Output-

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ relationop.cpp -o relationop
dataflair@asus-System-Product-Name:~/Desktop$ ./relationop
Welcome to DataFlair tutorials!
```

```
For 10 == 10 The output is: 1
For 10 == 20 The output is: 0
For 10 != 20 The output is: 1
For 10 != 10 The output is: 0
For 10 > 10 The output is: 0
For 10 > 20 The output is: 0
For 10 < 10 The output is: 0
For 10 < 20 The output is: 1
For 10 >= 10 The output is: 1
For 10 >= 20 The output is: 0
For 10 <= 10 The output is: 1
For 10 <= 20 The output is: 1
dataflair@asus-System-Product-Name:~/Desktop$
```

Table for Relational Operators in C and C++

Operator	Operand	Operation	Elucidation
==	a, b	(a==b)	Used to check if both operands are equal
!=	a, b	(a!=b)	Used to check if both operands are not equal
>	a, b	(a>b)	Used to check if the first operand is greater than the second
<	a, b	(a<b)	Used to check if the first operand is lesser than the second
>=	a, b	(a>=b)	Used to check if the first operand is greater than or equal to the second
<=	a, b	(a<=b)	Used to check if the first operand is lesser than or equal to the second

3. Logical Operators

It refers to the boolean values which can be expressed as:

- Binary logical operations, which involves two variables: AND and OR
- Unary logical operation: NOT

Logical Operators in C/C++ Includes –

1. **&& (AND)** – It is used to check if both the operands are true.
2. **|| (OR)** – These operators are used to check if at least one of the operand is true.
3. **! (NOT)** – Used to check if the operand is false

If the logical statement is satisfied (it is true), then the program will return the value 1, otherwise, if the relational statement is not satisfied (it is false), the program will return the value 0.

7 Basic C Programs that will help you to rise from Noob to Pro

Example of Logical Operators in C Programming-

```

1. #include <stdio.h>
2. int main()
3. {
4.     int a = 10, b = 10, c = 20, answer;
5.
6.     printf("Welcome to DataFlair tutorials!\n\n");
7.
8.     answer = (a == b) && (c > b);
9.     printf("For (%d == %d) && (%d != %d), the output is: %d \n",a,b,b,c,answer); //condition is true
10.
11.    answer = (a == b) && (c < b) && (c>0);
12.    printf("For (%d == %d) && (%d <= %d), the output is: %d \n",a,b,b,c,answer); //condition is false
13.
14.    answer = (a == b) || (b > c);
15.    printf("For (%d == %d) || (%d < %d), the output is: %d \n",a,b,c,b,answer); / condition is true
16.
17.    answer = (a != b) || (a <= b) || (a>c);
18.    printf("For (%d != %d) || (%d < %d), the output is: %d \n",a,b,c,b,answer); //condition is true
19.
20.    answer = !(a == b);
21.    printf("For !(%d == %d), the output is: %d \n",a,b,answer); //condition is false
22.
23.    answer = !(a != b);
24.    printf("For !(%d == %d), the output is: %d \n",a,b,answer); //condition is true
25.    return 0;
26. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

logicop.c

```
#include <stdio.h>
int main()
{
    int a = 10, b = 10, c = 20, answer;

    printf("Welcome to DataFlair tutorials!\n\n");

    answer = (a == b) && (c > b);
    printf("For (%d == %d) && (%d != %d), the output is: %d \n", a, b, b, c, answer); //cond

    answer = (a == b) && (c < b) && (c>0);
    printf("For (%d == %d) && (%d <= %d), the output is: %d \n", a, b, b, c, answer); //cond

    answer = (a == b) || (b > c);
    printf("For (%d == %d) || (%d < %d), the output is: %d \n", a, b, c, b, answer); //cond

    answer = (a != b) || (a <= b) || (a>c);
    printf("For (%d != %d) || (%d < %d), the output is: %d \n", a, b, c, b, answer); //cond

    answer = !(a == b);
    printf("For !(%d == %d), the output is: %d \n", a, b, answer); //cond

    answer = !(a != b);
    printf("For !(%d == %d), the output is: %d \n", a, b, answer); //cond

    return 0;
}
```

The output will be-

```

File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc logicop.c -o logicop
dataflair@admin4-H110M-H:~/Desktop$ ./logicop
Welcome to DataFlair tutorials!

For (10 == 10) && (10 != 20), the output is: 1
For (10 == 10) && (10 <= 20), the output is: 0
For (10 == 10) || (20 < 10), the output is: 1
For (10 != 10) || (20 < 10), the output is: 1
For !(10 == 10), the output is: 0
For !(10 == 10), the output is: 1
dataflair@admin4-H110M-H:~/Desktop$ 

```

Example of Logical Operators in C++

Here is a code in C++ which illustrates all the basic logical operators:

```

1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     int a = 10, b = 10, c = 20, answer;
7.
8.     cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
9.     answer = (a == b) && (c > b);
10.    cout<<"For (" << a << " == " << b << ") && (" << b << " != " << c << "), the output is: " << answer << endl; //condition is
11.    true
12.    answer = (a == b) && (c < b) && (c>0);
13.    cout<<"For (" << a << " == " << b << ") && (" << b << " <= " << c << "), the output is: " << answer << endl; //condition is
14.    false
15.    answer = (a == b) || (b > c);
16.    cout<<"For (" << a << " == " << b << ") && (" << c << " < " << b << "), the output is: " << answer << endl; //condition is
17.    true

```

```
17.  
18. answer = (a != b) || (a <= b) || (a>c);  
19. cout<<"For (" << a << " != " << b << ") && (" << c << " < " << b << "), the output is: " << answer << endl; //condition is  
true  
20.  
21. answer = !(a == b);  
22. cout<<"For !( " << a << " == " << b << "), the output is: " << answer << endl; //condition is false  
23.  
24. answer = !(a != b);  
25. cout<<"For !( " << a << " != " << b << "), the output is: " << answer << endl; //condition is true  
26.  
27. return 0;  
28. }
```

Code-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: logicop.cpp
- Code content:

```
#include <iostream>
using namespace std;

int main()
{
int a = 10, b = 10, c = 20, answer;

cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

answer = (a == b) && (c > b);
cout<<"For (" << a << " == " << b << ") && (" << b << " != " << c << "), the output is: " <<

answer = (a == b) && (c < b) && (c>0);
cout<<"For (" << a << " == " << b << ") && (" << b << " <= " << c << "), the output is: " <<

answer = (a == b) || (b > c);
cout<<"For (" << a << " == " << b << ") && (" << c << " < " << b << "), the output is: " << a

answer = (a != b) || (a <= b) || (a>c);
cout<<"For (" << a << " != " << b << ") && (" << c << " < " << b << "), the output is: " << a

answer = !(a == b);
cout<<"For !( " << a << " == " << b << "), the output is: " << answer << endl; //condition is f

answer = !(a != b);
cout<<"For !( " << a << " != " << b << "), the output is: " << answer << endl; //condition is t

return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ touch logicop.cpp
dataflair@asus-System-Product-Name:~/Desktop$ g++ logicop.cpp -o logicop
dataflair@asus-System-Product-Name:~/Desktop$ ./logicop
Welcome to DataFlair tutorials!

For (10 == 10) && (10 != 20), the output is: 1
For (10 == 10) && (10 <= 20), the output is: 0
For (10 == 10) && (20 < 10), the output is: 1
For (10 != 10) && (20 < 10), the output is: 1
For !(10 == 10), the output is: 0
For !(10 != 10), the output is: 1
dataflair@asus-System-Product-Name:~/Desktop$
```

Table for Logical Operators in C and C++

Operator	Operand	Operation	Elucidation
&&	a, b	(a && b)	AND: Used to check if both the operands are true
	a, b	(a b)	OR: Used to check if at least one of the operand is true
!	a	!a	NOT: Used to check if the operand is false

4. Assignment Operators

It is used to assign a particular value to a variable. We will discuss it in detail in the later section with its shorthand notations.

1. **= (Assignment)**- Used to assign a value from right side operand to left side operand.
2. **+= (Addition Assignment)**- To store the sum of both the operands to the left side operand.
3. **-= (Subtraction Assignment)** – To store the difference of both the operands to the left side operand.

4. ***= (Multiplication Assignment)** – To store the product of both the operands to the left side operand.
5. **/= (Division Assignment)** – To store the division of both the operands to the left side operand.
6. **%= (Remainder Assignment)** – To store the remainder of both the operands to the left side operand.

Explore the [Career Opportunities in C](#) with current salary structures

Example of Assignment Operators in C

```

1. #include<stdio.h>
2. int main()
3. {
4.     printf("Welcome to Dataflair tutorials!\n\n");
5.
6.     int number = 10, result;
7.     result = number;
8.
9.     printf("result = %d \n", result);
10.
11.    result += number; //Same as result = result + a
12.    printf("result = %d \n", result);
13.
14.    result -= number; //Same as result = result - a
15.    printf("result = %d \n", result);
16.
17.    result *= number; //Same as result = result * a
18.    printf("result = %d \n", result);
19.
20.    result /= number; //Same as result = result / a
21.    printf("result = %d \n", result);
22.
23.    result %= number; //Same as result = result % a
24.    printf("result = %d \n", result);
25.    return 0;
26. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

assignop.c

```
#include<stdio.h>
int main()
{
printf("Welcome to Dataflair tutorials!\n\n");

int number = 10, result;
result = number;

printf("result = %d \n", result);

result+= number; //Same as result = result + a
printf("result = %d \n", result);

result -= number; //Same as result = result - a
printf("result = %d \n", result);

result *= number; //Same as result = result * a
printf("result = %d \n", result);

result /= number; //Same as result = result / a
printf("result = %d \n", result);
return 0;
}
```

The output will be-

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc assignop.c -o assignop
dataflair@admin4-H110M-H:~/Desktop$ ./assignop
Welcome to Dataflair tutorials!

result = 10
result = 20
result = 10
result = 100
result = 10
result = 0
dataflair@admin4-H110M-H:~/Desktop$ 
```

Example of Assignment Operators in C++

Here is a code in C++ which illustrates all the basic assignment operators:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6. cout<<"Welcome to Dataflair tutorials!"<<endl<<endl;
7.
8. int number = 10, result;
9. result = number;
10.
11. cout<<"result = "<< result<<endl;
12.
13. result += number; //Same as result = result + a
14. cout<<"result = "<< result<<endl;
15.
16. result -= number; //Same as result = result - a
17. cout<<"result = "<< result<<endl;
```

```
18.  
19. result *= number; //Same as result = result * a  
20. cout<<"result = "<< result<<endl;  
21.  
22. result /= number; //Same as result = result / a  
23. cout<<"result = "<< result<<endl;  
24.  
25. result %= number; //Same as result = result % a  
26. cout<<"result = "<< result<<endl;  
27. return 0;  
28. }
```

Code-

dataflair@asus-System-Product-Name: ~

File Edit View Search Terminal Help

GNU nano 2.9.3 assignop.cpp

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    cout<<"Welcome to Dataflair tutorials!"<<endl<<endl;  
  
    int number = 10, result;  
    result = number;  
  
    cout<<"result = "<< result<<endl;  
  
    result += number; //Same as result = result + a  
    cout<<"result = "<< result<<endl;  
  
    result -= number; //Same as result = result - a  
    cout<<"result = "<< result<<endl;  
  
    result *= number; //Same as result = result * a  
    cout<<"result = "<< result<<endl;  
  
    result /= number; //Same as result = result / a  
    cout<<"result = "<< result<<endl;  
    result %= number; //Same as result = result % a  
    cout<<"result = "<< result<<endl;  
    return 0;  
}
```

Output-

```

File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ g++ assignop.cpp -o assignop
dataflair@asus-System-Product-Name:~/Desktop$ ./assignop
Welcome to Dataflair tutorials!

result = 10
result = 20
result = 10
result = 100
result = 10
result = 0
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Table for Assignment Operators in C and C++

Operator	Operand	Operation	Elucidation
=	a, b	a=b	Used to assign a value from right side operand to left side operand
+=	a, b	a+=b	$a=a+b$: The value of $a+b$ is stored in a
-=	a, b	a-=b	$a=a-b$: The value of $a-b$ is stored in a
=	a, b	a=b	$a=a*b$: The value of $a*b$ is stored in a
/=	a, b	a/=b	$a=a/b$: The value of a/b is stored in a
%=	a, b	a%-=b	$a=a \% b$: The value of $a \% b$ is stored in a

5. Bitwise Operators

It is based on the principle of performing operations bit by bit which is based on boolean algebra. It increases the processing speed and hence the efficiency of the program.

The Bitwise [Operators](#) in C/C++ Includes –

1. **& (Bitwise AND)** – Converts the value of both the operands into binary form and performs AND operation bit by bit.
2. **| (Bitwise OR)** – Converts the value of both the operands into binary form and performs OR operation bit by bit.
3. **^ (Bitwise exclusive OR)** – Converts the value of both the operands into binary form and performs EXCLUSIVE OR operation bit by bit.
4. **~ (One's complement operator)**: Converts the operand into its complementary form.
5. **<< – Left shift**
6. **>> – Right shift**

Key takeaway: Bitwise operators are not applicable in the case of float and double [data type in C](#).

In order to clearly understand bitwise operators, let us see the truth table for various bitwise operations and understand how it is associated with boolean algebra.

Since there are 2 variables, namely, a and b, there are 2² combinations for values a and b can take simultaneously.

AND – Both the operands should have boolean value 1 for the result to be 1.

OR – At least one operand should have boolean value 1 for the result to be 1.

XOR (EXCLUSIVE OR) – Either the first operand should have boolean value 1 or the second operand should have boolean value 1. Both cannot have the boolean value 1 simultaneously.

One Complement: iF

a	b	a & b	a b	a ^ b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

The left and right shift operators are responsible for shifting the binary values by some specific number of places.

Left shift: It specifies the value of the left operand to be shifted to the left by the number of bits specified by its right operand

Right shift: It specifies the value of the left operand to be shifted to the right by the number of bits specified by its right operand.

Let us take an example each of performing bitwise AND, OR, EXCLUSIVE OR and ONE'S COMPLEMENT operation.

Consider two operands, a and b with values:

a = 26 and b=14

Therefore, a & b is computed as follows:

1. Find the binary equivalent of a and b:
2. Perform boolean AND/OR/EXCLUSIVE OR operation bit by bit
3. Convert the answer into its corresponding decimal form.

- **Bitwise AND**

a = 26 = 1 1 0 1 0

b = 14 = 0 1 1 1 0

a & b = 0 1 0 1 0 which is equal to 10

- **Bitwise OR**

a = 26 = 1 1 0 1 0

b = 14 = 0 1 1 1 0

a | b = 1 1 1 1 0 which is equal to 30

- **Bitwise XOR**

a = 26 = 1 1 0 1 0

b = 14 = 0 1 1 1 0

a | b = 1 0 1 0 0 which is equal to 20

- **Bitwise One's Complement**

a = 26 = 1 1 0 1 0

Reversing its bits, we get 0 0 1 0 1 which is equal to 5 but this is not the correct answer!

The correct answer is: -(a+1) which is -27 which is in accordance with two's complement.

Example of Bitwise Operators in C

```
1. #include <stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     int a = 26, b = 14;
8.     printf(" Bitwise AND operation %d & %d : %d\n",a,b,a&b);
9.     printf(" Bitwise OR operation %d | %d : %d\n",a,b,a|b);
10.    printf(" Bitwise XOR operation %d ^ %d : %d\n",a,b,a^b);
11.    printf(" Bitwise ONE'S COMPLEMENT ~ %d operation : %d\n",a,~a);
12.
13. }
```

Code on Screen-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

GNU nano 2.9.3

bitwiseop.c

```
#include <stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int a = 26, b = 14;
    printf(" Bitwise AND operation %d & %d : %d\n",a,b,a&b);
    printf(" Bitwise OR operation %d | %d : %d\n",a,b,a|b);
    printf(" Bitwise XOR operation %d ^ %d : %d\n",a,b,a^b);
    printf(" Bitwise ONE'S COMPLEMENT ~ %d operation : %d\n",a,~a);
    return 0;
}
```

The Output is –

File Edit View Search Terminal Help

```
dataflair@admin4-H110M-H:~/Desktop$ gcc bitwiseop.c -o bitwiseop
```

```
dataflair@admin4-H110M-H:~/Desktop$ ./bitwiseop
```

```
Welcome to DataFlair tutorials!
```

```
Bitwise AND operation 26 & 14 : 10
```

```
Bitwise OR operation 26 | 14 : 30
```

```
Bitwise XOR operation 26 ^ 14 : 20
```

```
Bitwise ONE'S COMPLEMENT ~ 26 operation : -27
```

```
dataflair@admin4-H110M-H:~/Desktop$ █
```

Example of Bitwise Operators in C++

Here is a code in C++ which illustrates all the basic bitwise operators:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int a = 26, b = 14;
10.
11. cout<<" Bitwise AND operation" << a << " & " << b << " :" << (a&b) << endl;
12. cout<<" Bitwise OR operation" << a << " | " << b << " :" << (a|b) << endl;
13. cout<<" Bitwise XOR operation" << a << " ^ " << b << " :" << (a^b) << endl;
14. cout<<" Bitwise ONE'S COMPLEMENT ~" << a << " operation :" << (~a) << endl;
15. return 0;
```

16. }

Code-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

GNU nano 2.9.3

bitwiseop.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
    int a = 26, b = 14;
    cout<<" Bitwise AND operation" << a << " & " << b << " : " << (a&b) << endl;
    cout<<" Bitwise OR operation" << a << " | " << b << " : " << (a|b) << endl;
    cout<<" Bitwise XOR operation" << a << " ^ " << b << " : " << (a^b) << endl;
    cout<<" Bitwise ONE'S COMPLEMENT ~"<< a << " operation :"<< (~a) << endl;
    return 0;
}
```

Output-

```

File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ g++ bitwiseop.cpp -o bitwiseop
dataflair@asus-System-Product-Name:~/Desktop$ ./bitwiseop
Welcome to DataFlair tutorials!

Bitwise AND operation 26 & 14 : 10
Bitwise OR operation 26 | 14 : 30
Bitwise XOR operation 26 ^ 14 : 20
Bitwise ONE'S COMPLEMENT ~26 operation : -27
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Table for Bitwise Operators in C and C++

Operator	Operand	Operation	Elucidation
&	a, b	(a & b)	Bitwise AND: Converts the value of both the operands into binary form and performs AND- operation bit by bit
	a, b	(a b)	Bitwise OR: Converts the value of both the operands into binary form and performs OR- operation bit by bit
^	a, b	(a ^ b)	Bitwise exclusive OR: Converts the value of both the operands into binary form and performs EXCLUSIVE OR operation bit by bit
~	a	(~ a)	One's complement operator: Converts the operand into its complementary form
<<	a	a<<	Left shift
>>	a	a>>	Right shift

Before we start discussing Miscellaneous Operators, you should know [What is Pointers in C/C++](#)

6. Miscellaneous Operators

Apart from the above-discussed operators, there are certain operators which fall under this category which include sizeof and ternary (conditional) operators.

Here is a table which illustrates the use of these operators:

1. **sizeof** – It returns the memory occupied by the particular data type of the operand
2. **& (Pointer)** – It refers to the address (memory location) in which the operand is stored.
3. *** (Pointer)** – It is a pointer operator
4. **? (Condition)** – It is an alternative for if-else condition

Example of Miscellaneous Operators in C

```
1. #include<stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to Data Flair tutorials!\n\n");
6.
7. // Use of * and & operator
8.
9.     int number = 10, *pointer;
10.    pointer=&number; //Here the pointer stores the memory address of variable number
11.    printf("The value of the number is: %d\n",*pointer);
12.
13. // Use of ?: operator
14.    int expression1 = 10, expression2 = 20, expression3;
15.    expression3 = ( expression1 > expression2 ) ? expression1 : expression2;
16.    printf("The Output of the ternary statement is: %d", expression3);
17.    return 0;
18. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

misco.c

```
#include<stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    // Use of * and & operator
    int number = 10, *pointer;
    pointer=&number; //Here the pointer stores the memory address of variable number
    printf("The value of the number is: %d\n",*pointer);

    // Use of ?: operator
    int expression1 = 10, expression2 = 20, expression3;
    expression3 = ( expression1 > expression2 ) ? expression1 : expression2;
    printf("The Output of the ternary statement is: %d", expression3);
    return 0;
}
```

The output is-

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ cd Desktop
dataflair@admin4-H110M-H:~/Desktop$ gcc miscop.c -o miscop
dataflair@admin4-H110M-H:~/Desktop$ ./miscop
Welcome to DataFlair tutorials!

The value of the number is: 10
The Output of the ternary statement is: 20dataflair@admin4-H110M-H:~/Desktop$ □
```

Example of Miscellaneous Operators in C++

Here is a code in C++ which illustrates some of the basic misc operators:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to Data Flair tutorials!"<<endl<<endl;
8.
9. // Use of * and & operator
10.
11. int number = 10, *pointer;
12. pointer=&number; //Here the pointer stores the memory address of variable number
13. cout<<"The value of the number is: "<<*pointer<<endl;
14.
15. // Use of ?: operator
16. int expression1 = 10, expression2 = 20, expression3;
17. expression3 = ( expression1 > expression2 ) ? expression1 : expression2;
```

```
18. cout<<"The Output of the ternary statement is: "<< expression3 << endl;
19. return 0;
20. }
```

Code-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

GNU nano 2.9.3

miscop.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to Data Flair tutorials!"<<endl<<endl;
    // Use of * and & operator
    int number = 10, *pointer;
    pointer=&number; //Here the pointer stores the memory address of variable number
    cout<<"The value of the number is: "<<*pointer<<endl;
    // Use of ?: operator
    int expression1 = 10, expression2 = 20, expression3;
    expression3 = ( expression1 > expression2 ) ? expression1 : expression2;
    cout<<"The Output of the ternary statement is: "<< expression3 << endl;
    return 0;
}
```

Output-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ miscop.cpp -o miscop
^[[Adataflair@asus-System-Product-Name:~/Desktop$ ./miscop
Welcome to Data Flair tutorials!
```

The value of the number is: 10

The Output of the ternary statement is: 20

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

A table for Misc Operators in C and C++

Operator	Operand	Operation	Elucidation
sizeof	a	sizeof(a)	It returns the memory occupied by the particular data type of the operand
&	a	& a	It refers to the address (memory location) in which the operand is stored.
*	a	* a	It is a pointer
?:	a,b	a? b: statement	It is an alternative for if-else condition

Summary

Operators are the basic foundation of the C/C++ Programming language. Now, you can perform any operation of mathematical, logical, relational, with other condition. We learned each operator in C and C++ with their examples. As a beginner, you should know each operator, and how, why, when to use it.

Strings in C/C++ | Concepts you need to Learn to Stand Alone

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 27, 2019

Are you aware of strings in C/C++?

Strings in C and C++ are nothing but an array of characters terminated by the null character '\0'. Let us acknowledge the significance of strings in C/C++ by considering a simple problem.

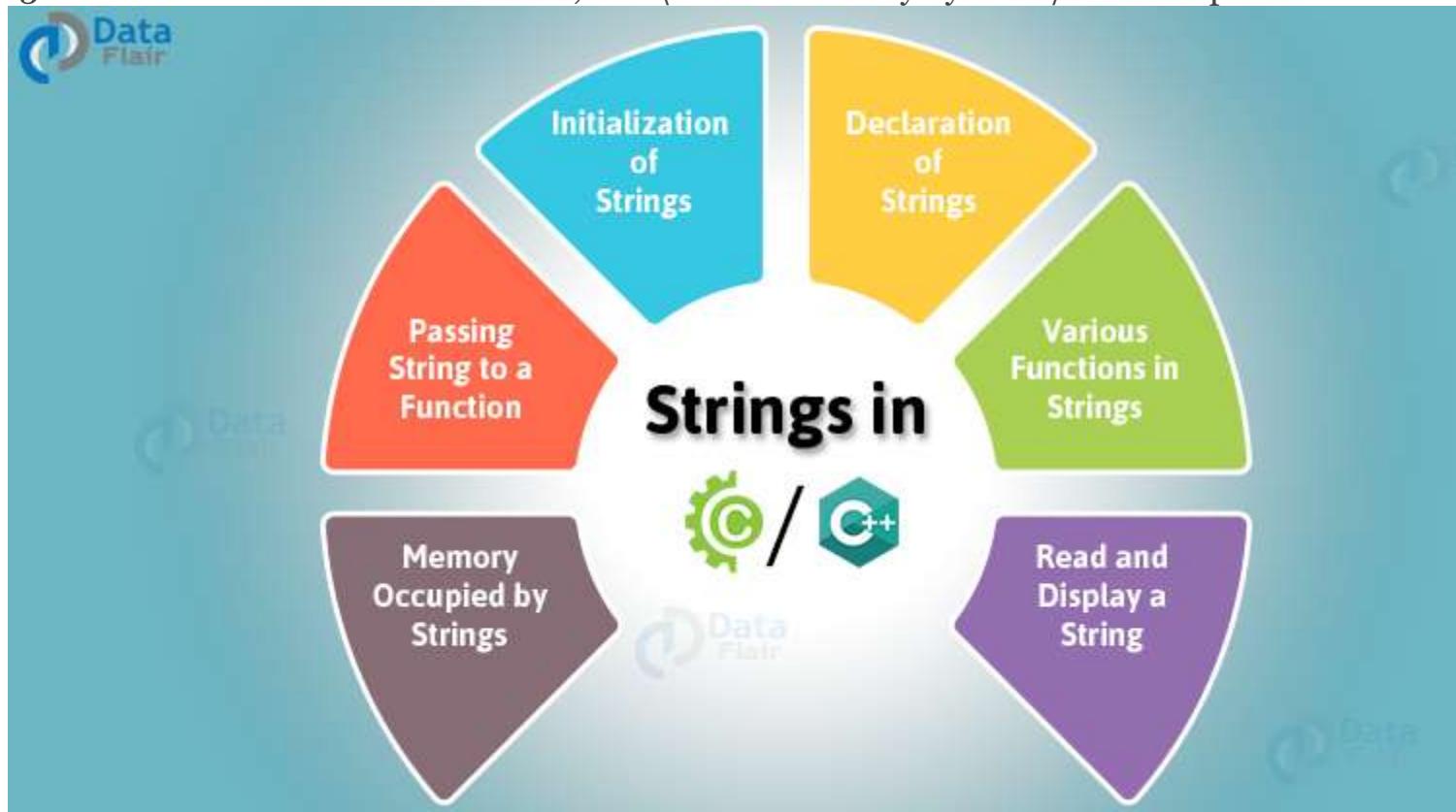
Suppose you want to store the details of several employees such as their names, address, and qualification. It is pretty obvious that we can't use any other data type except strings, as these records contain more than one letter. We can perform strings manipulation in the C/C++. This gives birth to even more applications of strings and makes it all the more significant.

1. What are Strings in C and C++?

In layman language, strings are nothing but a set of characters treated as a single entity.

In programming terminology, a string is:

1. A one-dimensional array of characters.
2. Enclosed within double quotes.
3. Terminated with a null character, i.e. '\0' automatically by the C/C++ compiler.



2. Declare Strings

Strings are declared in the same way as [arrays in C/C++](#), but restricted to the char data type. It is important to declare the string before using it. The C/C++ compiler reserves a specific block of memory for the sequence of characters. At the end of the string, we append the null character.

C++ has an upper hand over C as a string in C++ can be implemented in 2 ways:

1. As an array of characters
2. Using the predefined data type – “string”.

Its basic syntax is in C/C++

char string_name [30]; // Here 30 blocks of memory is reserved for the string

The basic syntax exclusively available in C++ is:

/ A string data type occupies 32 bytes in the computer memory according to a 64-bit compiler */*

string string_name;

In order to perform string manipulations, it is mandatory to use the header file supported by the C/C++ standard library, namely:

#include<string.h>

Create your own [Header Files in C](#) within Seconds

3. Initialize Strings

You can initialize Strings in C in different ways:

Let us consider some examples to demonstrate the various methods to declare a string:

1. `char name[] = "DataFlair";`
1. `char name[] = {'D', 'a', 't', 'a', 'F', 'l', 'i', 'r', '\0' };`
1. `char name [30] = "DataFlair";`
1. `char name [30] = {'D', 'a', 't', 'a', 'F', 'l', 'i', 'r', '\0' };`

You can initialize a string in C++ in different ways:

Let us consider some examples to demonstrate the various methods to declare a string:

1. `char name[] = "DataFlair";`
1. `char name[] = {'D', 'a', 't', 'a', 'F', 'l', 'i', 'r', '\0' };`
1. `char name [30] = "DataFlair";`
1. `char name [30] = {'D', 'a', 't', 'a', 'F', 'l', 'i', 'r', '\0' };`
1. `string name = "DataFlair";`

4. Memory Occupied by Strings in C/C++

We have already discussed that a character occupies 1 byte of memory in [Data Types in C/C++](#). Similarly, a string, if initialized first, would occupy the memory:

(Number of characters in the string) x 1 byte + 1 byte

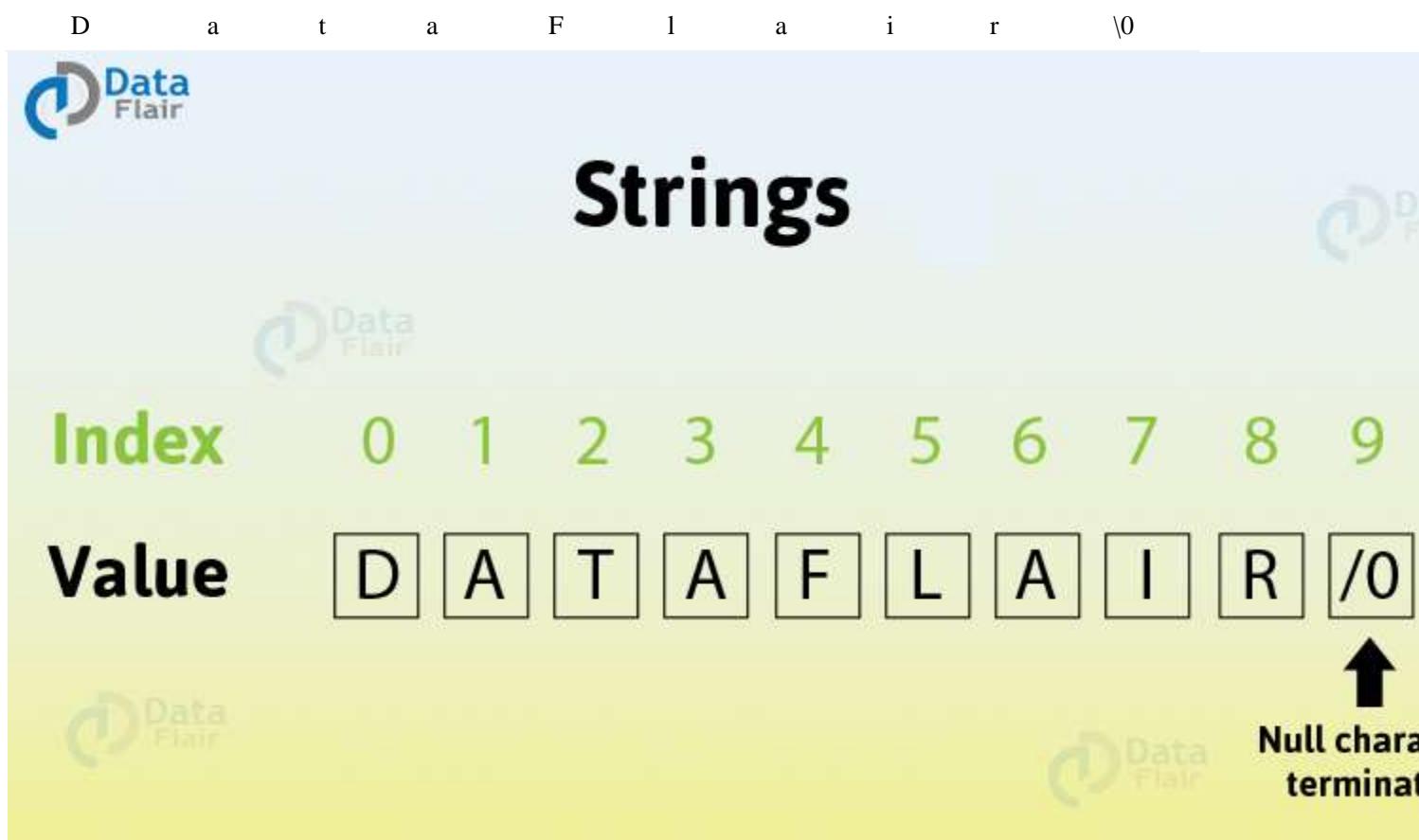
Key takeaway: 1 byte is added while calculating the size of the string. It is because the C/C++ compiler automatically appends the null character, i.e. '\0'. This null character consumes 1 byte of memory in the system.

Let us get back to the above example:

```
1. char name[ ] = "DataFlair";
```

There data given below should be made in the form of a table

name[0] name[1] name[2] name[3] name[4] name[5] name[6] name[7] name[8]
name[9]



Example of sizeof() operator in C

Here is a program in C that illustrates the amount of memory allocated to strings with the help of sizeof() operator.

```
1. #include<stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     char name1[] = "DataFlair";
8.     char name2[] = {'D', 'a', 't', 'a', 'F', 'l', 'a', 'i', 'r', '\0'};
9.     char name3[30] = "DataFlair"; // 30 bytes is already allocated
10.    char name4[30] = {'D', 'a', 't', 'a', 'F', 'l', 'a', 'i', 'r', '\0'}; //30 bytes is already allocated
```

```

11. char name5[] = "Data Flair"; //a blank character occupies 1 byte of memory
12.
13. printf("Size of %s is: %ld \n",name1,sizeof(name1));
14. printf("Size of %s is: %ld \n",name2,sizeof(name2));
15. printf("Size of %s is: %ld \n",name3,sizeof(name3));
16. printf("Size of %s is: %ld \n",name4,sizeof(name4));
17. printf("Size of %s is: %ld \n",name5,sizeof(name5));
18. return 0;
19. }
```

It's time to explore Standard Library Function in C

Code on Screen-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: string.c
- Code content:

```

#include<stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");

    char name1[] = "DataFlair";
    char name2[] = {'D' , 'a' , 't' , 'a' , 'F' , 'l' , 'a' , 'i' , 'r' , '\0' };
    char name3[30] = "DataFlair"; // 30 bytes is already allocated
    char name4[30] = {'D' , 'a' , 't' , 'a' , 'F' , 'l' , 'a' , 'i' , 'r' , '\0' }; //30
    char name5[] = "Data Flair"; //a blank character occupies 1 byte of memory

    printf("Size of %s is: %ld \n",name1,sizeof(name1));
    printf("Size of %s is: %ld \n",name2,sizeof(name2));
    printf("Size of %s is: %ld \n",name3,sizeof(name3));
    printf("Size of %s is: %ld \n",name4,sizeof(name4));
    printf("Size of %s is: %ld \n",name5,sizeof(name5));
    return 0;
}
```

Output-

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc string.c -o string
dataflair@admin4-H110M-H:~/Desktop$ ./string
Welcome to DataFlair tutorials!

Size of DataFlair is: 10
Size of DataFlair is: 10
Size of DataFlair is: 30
Size of DataFlair is: 30
Size of Data Flair is: 11
dataflair@admin4-H110M-H:~/Desktop$ 
```

Example of sizeof() operator in C++

Here is a program in C++ that illustrates the amount of memory allocated to strings with the help of **sizeof()** operator:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. char name1[] = "DataFlair";
10. char name2[] = {'D', 'a', 't', 'a', 'F', 'l', 'i', 'r', '\0'};
11. char name3[30] = "DataFlair"; // 30 bytes is already allocated
```

```

12. char name4[30] = {'D', 'a', 't', 'a', 'F', 'T', 'a', 'i', 'r', '\0'}; //30 bytes is already allocated
13. char name5[] = "Data Flair"; //a blank character occupies 1 byte of memory
14. string name6 = "DataFlair";
15.
16. cout<<"Size of " << name1 << " is: "<< sizeof(name1) << endl;
17. cout<<"Size of " << name2 << " is: "<< sizeof(name2) << endl;
18. cout<<"Size of " << name3 << " is: "<< sizeof(name3) << endl;
19. cout<<"Size of " << name4 << " is: "<< sizeof(name4) << endl;
20. cout<<"Size of " << name5 << " is: "<< sizeof(name5) << endl;
21. cout<<"Size of " << name6 << " is: "<< sizeof(name6) << endl;
22.
23. return 0;
24. }

```

Code-

The screenshot shows a terminal window with the following details:

- Title Bar:** dataflair@asus-System-Product-Name
- File Menu:** File Edit View Search Terminal Help
- Version:** GNU nano 2.9.3
- File Name:** string.cpp
- Code Content:**

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    char name1[] = "DataFlair";
    char name2[] = {'D', 'a', 't', 'a', 'F', 'T', 'a', 'i', 'r', '\0'}; // 30 bytes is already allocated
    char name3[30] = "DataFlair"; // 30 bytes is already allocated
    char name4[30] = {'D', 'a', 't', 'a', 'F', 'T', 'a', 'i', 'r', '\0'}; //30 bytes is already allocated
    char name5[] = "Data Flair"; //a blank character occupies 1 byte of memory
    string name6 = "DataFlair";

    cout<<"Size of " << name1 << " is: "<< sizeof(name1) << endl;
    cout<<"Size of " << name2 << " is: "<< sizeof(name2) << endl;
    cout<<"Size of " << name3 << " is: "<< sizeof(name3) << endl;
    cout<<"Size of " << name4 << " is: "<< sizeof(name4) << endl;
    cout<<"Size of " << name5 << " is: "<< sizeof(name5) << endl;
    cout<<"Size of " << name6 << " is: "<< sizeof(name6) << endl;

    return 0;
}
```
- Output:** The terminal shows the output of the program, which includes the welcome message and the sizeof() results for each variable.

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ touch string.cpp
dataflair@asus-System-Product-Name:~/Desktop$ g++ string.cpp -o string
dataflair@asus-System-Product-Name:~/Desktop$ ./string
Welcome to DataFlair tutorials!

Size of DataFlair is: 10
Size of DataFlair is: 10
Size of DataFlair is: 30
Size of DataFlair is: 30
Size of Data Flair is: 11
Size of DataFlair is: 32
dataflair@asus-System-Product-Name:~/Desktop$
```

5. Read and Display String in C and C++

Example to read and display strings in C

Here is a code in C that illustrates how to take a [string](#) as an input from the user and display it:

```
1. #include <stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     char name[30];
8.     printf("Enter the string: ");
9.     scanf("%s", name);
10.    printf("The string is %s\n", name);
11.
12. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

string2.c

```
#include <stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\\n\\n");
    char name[30];
    printf("Enter the string: ");
    scanf("%s", name);
    printf("The string is %s\\n", name);
    return 0;
}
```

Output-

```
dataflair@admin4-H110M-H: ~/De
```

```
File Edit View Search Terminal Help
```

```
dataflair@admin4-H110M-H:~/Desktop$ gcc string2.c -o string2
```

```
dataflair@admin4-H110M-H:~/Desktop$ ./string2
```

```
Welcome to DataFlair tutorials!
```

```
Enter the string: DataFlair
```

```
The string is DataFlair
```

```
dataflair@admin4-H110M-H:~/Desktop$ 
```

Example to read and display strings in C++

Here is a code in C++ that illustrates how to take a string as an input from the user and display it:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl;
8.
9. string name;
10. cout<<"Enter the string: ";
```

```
11. cin>>name;
12. cout<<"The string is "<< name <<endl;
13. return 0;
14. }
```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

string2.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl;

    string name;
    cout<<"Enter the string: ";
    cin>>name;
    cout<<"The string is "<< name <<endl;
    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ touch string2.cpp  
dataflair@asus-System-Product-Name:~/Desktop$ g++ string2.cpp -o string2  
dataflair@asus-System-Product-Name:~/Desktop$ ./string2  
Welcome to DataFlair tutorials!  
Enter the string: DataFlair  
The string is DataFlair  
dataflair@asus-System-Product-Name:~/Desktop$ ]
```

6. Read and Display a Line using Strings in C/C++

There are 2 important inbuilt [*functions in C/C++*](#) that help you to read and print a line.

1. **gets()**: It is similar to **scanf()** which helps you take a line as input.
2. **puts()**: It is similar to **printf()** which helps you display a line as output.

But there is a problem associated with **gets()** which makes it dangerous to use because of Buffer Overflow as **gets()** as it is not capable of performing the array bound test. Hence, it is deprecated. To overcome this problem, **fgets()** is used instead of **gets()**.

Example to read and display line using strings in C

Here is a code in C that illustrates how to read and display a line using strings with the help of **fgets()** and **puts()** functions:

1. #define STRING_SIZE 30
2. #include <stdio.h>
3. int main()

```
4. {
5.
6. printf("Welcome to DataFlair tutorials!\n\n");
7.
8. char line[STRING_SIZE];
9. printf("Enter a sentence ");
10. fgets(line, STRING_SIZE, stdin); // To read the string
11. printf("The sentence is: ");
12. puts(line); // To display the string
13. return 0;
14. }
```

Code on Screen-

The screenshot shows a terminal window with a dark background. At the top, there is a header bar with the text "dataflair@asus-System-Product-Name". Below the header is a menu bar with "File Edit View Search Terminal Help". The main area of the terminal shows the following content:

```
File Edit View Search Terminal Help
GNU nano 2.9.3                                         string3.c

#define STRING_SIZE 50
#include <stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");

    char line[STRING_SIZE];
    printf("Enter a sentence ");
    fgets(line, STRING_SIZE, stdin); // To read the string
    printf("The sentence is: ");
    puts(line); // To display the string
    return 0;
}
```

Output-

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ gcc string3.c -o string3
dataflair@asus-System-Product-Name:~/Desktop$ ./string3
Welcome to DataFlair tutorials!

Enter a sentence Welcome to DataFlair tutorials!
The sentence is: Welcome to DataFlair tutorials!

dataflair@asus-System-Product-Name:~/Desktop$ 
```

In C++, there are 5 important inbuilt functions that help you perform input/output operations:

- **getchar()**: This function returns a single character from the standard input device.

Syntax – *character_type_variable = getchar();*

For instance,

```
char letter;
letter = getchar();
```

Take a Break and [Learn Variables in C/C++](#)

- **putchar()**: This function transmits a single character to the standard output device.

Syntax – *putchar (character_type_variable);*

For instance,

```
char letter;  
putchar (letter);
```

- **gets()**: This function is used to take string input. It accepts only one argument.

Syntax – gets (character_type_variable)

For instance,

```
char letter;  
gets ( letter );
```

- **puts()**: This function is used to display the string output. It also accepts only one argument.

Syntax – puts (character_type_variable)

For instance,

```
char letter;  
puts ( letter );
```

- **getline()**: This function is used to read a line to the standard input device.

Syntax – getline (cin, string_type_variable)

For instance,

```
string letter;  
getline ( cin, letter );
```

Key takeaway: In C++, there is no such thing as “**putline()**” function.

Example of C++ string input/output functions

Here is a code in C++ that illustrates the use of some basic string input/output functions:

```
1. #include <iostream>  
2. #include<string.h>  
3. #include<sstream>  
4. using namespace std;  
5.  
6. int main()  
7. {  
8.  
9. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;  
10.  
11. char letter;  
12. char word[30];  
13. string line;  
14.  
15. cout<<"Enter a character: ";  
16. letter = getchar();  
17.  
18. cout<<"Enter a string: ";  
19. cin.ignore();
```

```
20. cin>>word;
21.
22. cout<<"Enter a sentence: ";
23. cin.ignore();
24. getline(cin, line);
25.
26. cout<<"\nThe character is: ";
27. putchar(letter);
28. cout<<endl;
29.
30. cout<<"The string is: "<< word << endl;
31.
32. cout<<"The sentence is: "<< line << endl;
33.
34. return 0;
35. }
```

Code-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: string3.cpp
- Code content:

```
#include <iostream>
#include<string.h>
#include<sstream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
    char letter;
    char word[30];
    string line;

    cout<<"Enter a character: ";
    letter = getchar();

    cout<<"Enter a string ";
    cin.ignore();
    cin>>word;

    cout<<"Enter a sentence: ";
    cin.ignore();
    getline(cin, line);

    cout<<"\nThe character is: ";
    putchar(letter);
    cout<<endl;

    cout<<"The string is: "<< word << endl;
    cout<<"The sentence is: "<< line << endl;
}

return 0;
```
- Output:

```
Welcome to DataFlair tutorials!
The character is: 
The string is: DataFlair
The sentence is: DataFlair tutorials!
```

Output-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

dataflair@asus-System-Product-Name:~/Desktop\$ g++ string3.cpp -o string3

dataflair@asus-System-Product-Name:~/Desktop\$./string3

Welcome to DataFlair tutorials!

Enter a character: A

Enter a string DataFlair

Enter a sentence: I understood how to perform string input/output operations! Yay!

The character is: A

The string is: DataFlair

The sentence is: I understood how to perform string input/output operations! Yay!

dataflair@asus-System-Product-Name:~/Desktop\$

Master the Concept of [Polymorphism in C++](#) in a few minutes

7. Various Functions in Strings

The C/C++ language offers various functions associated with strings.

Here is a table that succinctly summarizes some of the basic functions available for string manipulations:

Function	Meaning	Elucidation
strlen(s)	String Length	We use it to find the length of the string.
strcpy(s1,s2)	String Copy	We use to copy the value of the second string argument to the first string argument.
strcat(s1,s2)	String Concatenation	We use to join the second string argument to the end of the first string argument.
strcmp(s1,s2)	String Compare	We use it to compare both the string arguments. It will return: 0, if s1 and s2 are equal; Less than 0 if s1 < s2; Greater than 0 if s1 > s2
strlwr	String Lower	We use it to convert all the characters in the string to lowercase characters.

strupr	String Upper	We use it to convert all the characters in the string to uppercase characters.
strncat	String Concatenation of n characters	We use it to join the first n characters to the end of the first string.
strncpy	String Copy of n characters	We use it to copy the first n characters of one string to the other.
strcmp / strcmp	String Compare of n characters	We use it to compare the first n characters of both the strings in C.
stricmp / stricmp	String Compare without being case sensitive	We use it to compare both the strings without being case sensitive.
strnicmp	String Compare of n characters without being case sensitive	We use it to compare the first n characters of the string without being case sensitive.
strdup	String Duplicate	We use it in the duplication of a string.
strchr	String First Character Occurrence	We use it to find the first occurrence of a specific character in the string.
strrchr	String Last Character Occurrence	We use it to find the last occurrence of a specific character in the string.
strstr	String Occurrence in String	We use it to find the first occurrence of a string in another string.
strset	String set	We use it to set all the characters of a string to a specific character.
strnset	String set of n characters	We use it to set the first n characters of a string to a specific character.

Learn Virtual Function in C++ with Real-time Example

Example in C

Here is a code in C that illustrates the use of the 4 basic functions:

1. strlen()
2. strcpy()
3. strcmp()
4. strcat()

```

1. #include <stdio.h>
2. #include <string.h>
3. int main ()
4. {
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     char s1[30] = "Data";
8.     char s2[30] = "Flair";
9.     char s3[30];
10.    int length;
11.
12.    length = strlen(s1); // length of s1
13.    printf("The length of s1 = %s is: %d\n", s1, length);
14.
15.    strcpy(s3, s1); // s1 is copied in s3
16.    printf("The copied value of s3 is: %s\n", s3 );
17.
18.    if (strcmp(s1, s3) == 0) // both are equal as s1 is copied in s3
19.    {
20.        printf("s1 and s3 are equal\n");

```

```

21. }
22. else
23. {
24.     printf("s1 and s3 are not equal\n");
25. }
26.
27. strcat( s1, s2); // appends s2 at the end of s1
28. printf("The concatenation of s1 = %s and s2 = %s is: %s\n", s1, s2, s1);
29. return 0;
30. }

```

Code on Screen-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

string4.c

```

#include <stdio.h>
#include <string.h>
int main ()
{

printf("Welcome to DataFlair tutorials!\n\n");

char s1[30] = "Data";
char s2[30] = "Flair";
char s3[30];
int length;

length = strlen(s1); // length of s1
printf("The length of s1 = %s is: %d\n", s1, length);

strcpy(s3, s1); // s1 is copied in s3
printf("The copied value of s3 is: %s\n", s3 );

if (strcmp(s1, s3) == 0) // both are equal as s1 is copied in s3
{
printf("s1 and s3 are equal\n");
}
else
{
printf("s1 and s3 are not equal\n");
}

strcat( s1, s2); // appends s2 at the end of s1
printf("The concatenation of s1 = %s and s2 = %s is: %s\n", s1, s2, s1);
return 0;
}

```

Output-

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc string4.c -o string4
dataflair@admin4-H110M-H:~/Desktop$ ./string4
Welcome to DataFlair tutorials!

The length of s1 = Data is: 4
The copied value of s3 is: Data
s1 and s3 are equal
The concatenation of s1 = DataFlair and s2 = Flair is: DataFlair
dataflair@admin4-H110M-H:~/Desktop$ █
```

Example in C++

Here is a code in C++ that illustrates the use of the 4 basic functions:

- `strlen()`
- `strcpy()`
- `strcmp()`
- `strcat()`

```
1. #include <iostream>
2. #include <string.h>
3. using namespace std;
4.
5. int main ()
6. {
7.
8. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
9.
```

```
10. char s1[30] = "Data";
11. char s2[30] = "Flair";
12. char s3[30];
13. int length;
14.
15. length = strlen(s1); // length of s1
16. cout<<"The length of s1 = "<< s1 << " is: " << length << endl;;
17.
18. strcpy(s3, s1); // s1 is copied in s3
19. cout<<"The copied value of s3 is: "<< s3 << endl;
20.
21. if (strcmp(s1, s3) == 0) // both are equal as s1 is copied in s3
22. {
23.     cout<<"s1 and s3 are equal"<< endl;
24. }
25. else
26. {
27.     cout<<"s1 and s3 are not equal"<< endl;
28. }
29.
30. strcat( s1, s2); // appends s2 at the end of s1
31. cout<<"The concatenation of s1 and s2 is: "<< s1 << endl;
32. return 0;
33. }
```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

string4.cpp

```
#include <iostream>
#include <string.h>
using namespace std;

int main ()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    char s1[30] = "Data";
    char s2[30] = "Flair";
    char s3[30];
    int length;

    length = strlen(s1); // Length of s1
    cout<<"The length of s1 = "<< s1 << " is: " << length << endl;

    strcpy(s3, s1); // s1 is copied in s3
    cout<<"The copied value of s3 is: "<< s3 << endl;

    if (strcmp(s1, s3) == 0) // both are equal as s1 is copied in s3
    {
        cout<<"s1 and s3 are equal"<<endl;
    }
    else
    {
        cout<<"s1 and s3 are not equal"<<endl;
    }

    strcat( s1, s2); // appends s2 at the end of s1
    cout<<"The concatenation of s1 and s2 is: "<< s1 << endl;
    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ g++ string4.cpp -o string4
dataflair@asus-System-Product-Name:~/Desktop$ ./string4
Welcome to DataFlair tutorials!

The length of s1 = Data is: 4
The copied value of s3 is: Data
s1 and s3 are equal
The concatenation of s1 and s2 is: DataFlair
dataflair@asus-System-Product-Name:~/Desktop$ 
```

8. Passing String to a Function

Functions are nothing but a fragment of code written to serve a specific purpose. We use it instead of piling all the logical statements onto the main function. It increases the code readability and enhances the reusability feature in the C programming language.

Various arguments or parameters can be passed to a function according to the programmer's requirement.

Example of Passing String to a Function in C

Here is a code in C that illustrates how a string is passed to a function:

```
1. #include <stdio.h>
2. void display(char string[]); // Function declaration with string as parameter
3.
4. int main()
5. {
6.
```

```
7. printf("Welcome to DataFlair tutorials!\n\n");
8.
9. char string[30];
10. printf("Enter the string: ");
11. fgets(string, 30, stdin);
12. display(string); // Passing the string to the function display
13. return 0;
14. }
15. void display(char string[])
16. {
17. printf("The string is: ");
18. puts(string);
19. }
```

Code on Screen-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Current file: stringf.c
- Code content (in nano editor):

```
#include <stdio.h>
void display(char string[]); // Function declaration with string as parameter

int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");

    char string[30];
    printf("Enter the string: ");
    fgets(string, 30, stdin);
    display(string);      // Passing the string to the function display
    return 0;
}
void display(char string[])
{
    printf("The string is: ");
    puts(string);
}
```
- Output area (right side):

```
stringf.c
```

Output-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

dataflair@asus-System-Product-Name:~/Desktop\$ gcc stringf.c -o stringf

dataflair@asus-System-Product-Name:~/Desktop\$./stringf

Welcome to DataFlair tutorials!

Enter the string: DataFlair

The string is: DataFlair

dataflair@asus-System-Product-Name:~/Desktop\$ █

Example of Passing String to a Function in C++

Here is a code in C++ that illustrates how a string is passed to a function:

```
1. #include <iostream>
2. using namespace std;
3.
4. void display(string line); // Function declaration with string as parameter
5.
6. int main()
7. {
8.
9. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
10.
11. string line;
12. cout<<"Enter the string: ";
```

```
13. getline(cin, line);
14. display(line); // Passing the string to the function display
15. return 0;
16. }
17. void display(string line)
18. {
19. cout<<"The string is: "<<line<<endl;
20. }
```

Code-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: stringf.cpp
- Code content:

```
#include <iostream>
using namespace std;

void display(string line); // Function declaration with string as parameter

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    string line;
    cout<<"Enter the string: ";
    getline(cin, line);
    display(line); // Passing the string to the function display
    return 0;
}
void display(string line)
{
    cout<<"The string is: "<<line<<endl;
}
```
- Output:

```
Welcome to DataFlair tutorials!
Enter the string: DataFlair
The string is: DataFlair
```

Output-

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ g++ stringf.cpp -o stringf
dataflair@asus-System-Product-Name:~/Desktop$ ./stringf
Welcome to DataFlair tutorials!

Enter the string: I have learnt how to pass a string to a function!
The string is: I have learnt how to pass a string to a function!
dataflair@asus-System-Product-Name:~/Desktop$ 
```

9. Summary

In this tutorial, we discussed the meaning of Strings in the C and C++ Programming Languages. We acknowledged its significance by considering a simple problem. By understanding that *strings are nothing but an array of characters*, we inferred how to declare and initialize them. Further, we carried on our discussion by developing an understanding of how memory is allocated to strings. Then, we saw how to read and display a word and a sentence using strings. We concluded our discussion by stating the various string functions followed by illustrative programs.

Samurai Technique to Learn Arrays in C and C++

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 26, 2019

Arrays in C and C++ are a collection of similar data types that are accessed using a common name. For a clear understanding, let's consider a problem, where we want to store and display the salary of 10 employees in an organization. Before learning the concept of arrays, you would find this problem a tedious task and would probably come up with the solution: “Declare 10 floating-type variables, each representing the salary of the 10 employees respectively, use **scanf/cin statement** to store their respective values and **printf/cout statement** to display it.”

Now, what if the number of employees whose salary to be stored and displayed goes up to 100? It is practically not feasible to go by this conventional method. The simple solution to this onerous problem is by the implementation of arrays. In a nutshell, *arrays in C/C++ are used when we have a large number of objects in hand. It proves to be quite helpful to manage data.*

Arrays in C and C++ tutorial cover the following topics-



1. Arrays in C and C++

An array is a data structure, a sequential collection of similar data types that can easily be accessed using a common variable name. It is similar to the concept of a [list in Python](#) when talking about a single-dimensional array.

Arrays in C/C++ can have multiple dimensions, starting from one-dimension to numerous. Dimensions are positive integral numbers. It cannot have a floating-type value.

The trivial names of the various dimensions of arrays are as follows:

1. One-dimensional array – Vectors
2. Two-dimensional array – Matrix

An array is a generalized term when its dimension is unspecified.

2. Declaration of Arrays in C/C++

Arrays are declared in a similar fashion like any other variable of the same data type with addition to the use of square brackets, which is followed by the variable name. As soon as an array is declared in C and C++, a block of memory is reserved for it. The total size of an array is the product of the number of elements it has in its square bracket and the size of the data type it stores.

Don't forget to check, [how to declare variables in C/C++](#)

When an array is multidimensional, let's say:

`data_type array_name [order-1] [order-2] [order-3].....[order-n]`

The total size occupied by the array would be:

(Number of bytes occupied by data_type) x (order-1 x order-2 x order-3 xorder-n)

Once the orders of the array are specified, it cannot be changed. In other words, the size of an array remains constant which is used in static memory allocation i.e. it is not dynamic in nature.

Syntax of Arrays

`data_type array_name [size];`

For example- float marks [10];

3. Data Type of an Array

An array can have any data type like int, float, char but there are 2 exceptions in the C language which do not support arrays. They are:

- **void data type:** It is not supported in C and hence would display a compilation error.

- **Functions:** Array of pointers and function pointers are not possible

Take a break of 4 min and learn [data types in C and C++](#)

4. Initializing Arrays in C and C++

Arrays in C/C++ may or may not be initialized at the time of declaration.

This is how an array looks like if it is initialized at the time of declaration.

- **Without size specification**

1. int array [] = {4 , 9 , 2 , 1};

- **With size specification**

1. int array [10] = {4 , 9 , 2 , 1};

Let us now try to understand how elements are stored when they are initialized.

Since there are 4 elements in the array, 4×4 bytes, that is, 16 bytes of memory is reserved for the array which can be visualized as:

array[0]	array [1]	array[2]	array[3]
4	9	2	1

The indexing of an array starts from 0. Hence the indexing of the elements goes from 0 to size of the array – 1.

5. Accessing Arrays

Elements in an array are accessed with the help of their index. The index helps us in traversing the array.

How to Access Arrays in C?

Let's discuss with an example, how elements are accessed in an array:

```

1. #include<stdio.h>
2. int main()
3. {
4.
5. printf("Welcome to DataFlair tutorials!\n\n");
6.
7. int size_of_array, iteration;
8. int array [30];
9. printf("Enter the size of the array: ");
10. scanf("%d", &size_of_array);
11. printf("Enter the elements of the array:\n");
12. for(iteration = 0 ; iteration < size_of_array ; iteration ++ )
13. {
14. scanf("%d", &array[iteration]);
15. }
16. printf("The array is:\n");
17. for(iteration = 0 ; iteration < size_of_array ; iteration ++ )
18. {

```

```
19. printf("The element at index %d\n",iteration, array[iteration]);
20. }
21. return 0;
22. }
```

Struggling with programs in C? Get a free tutorial for the [basic structure of C program](#)
Code on Screen-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Current file: array.c
- Code content:

```
#include<stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int size_of_array, iteration;
    int array [30];
    printf("Enter the size of the array: ");
    scanf("%d", &size_of_array);
    printf("Enter the elements of the array:\n");
    for(iteration = 0 ; iteration < size_of_array ; iteration ++ )
    {
        scanf("%d", &array[iteration]);
    }
    printf("The array is:\n");
    for(iteration = 0 ; iteration < size_of_array ; iteration ++ )
    {
        printf("The element at index %d is: %d\n",iteration, array[iteration]);
    }
    return 0;
}
```

Output-

dataflair@admin4-H110M-H: ~/De

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc array.c -o array
dataflair@admin4-H110M-H:~/Desktop$ ./array
Welcome to DataFlair tutorials!

Enter the size of the array: 5
Enter the elements of the array:
12
6
32
11
8
The array is:
The element at index 0 is: 12
The element at index 1 is: 6
The element at index 2 is: 32
The element at index 3 is: 11
The element at index 4 is: 8
dataflair@admin4-H110M-H:~/Desktop$ 
```

How to Access Arrays in C++?

Here is a code in C++ that illustrates how elements are accessed in an array:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int size_of_array, iteration;
10. int array [30];
11. cout<<"Enter the size of the array: ";
12. cin>>size_of_array;
13. cout<<"Enter the elements of the array: "<<endl;
14. for(iteration = 0 ; iteration < size_of_array ; iteration ++ )
15. {
```

```
16. cin>>array[iteration];
17. }
18. cout<<"The array is: "<<endl;
19. for(iteration = 0 ; iteration < size_of_array ; iteration ++ )
20. {
21. cout<<"The element at index "<< iteration << "is:" << array[iteration] <<endl;
22. }
23. return 0;
24. }
```

Code –

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name:
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: array.cpp
- Code content:

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int size_of_array, iteration;
    int array [30];
    cout<<"Enter the size of the array: ";
    cin>>size_of_array;
    cout<<"Enter the elements of the array: "<<endl;
    for(iteration = 0 ; iteration < size_of_array ; iteration ++ )
    {
        cin>>array[iteration];
    }
    cout<<"The array is: "<<endl;
    for(iteration = 0 ; iteration < size_of_array ; iteration ++ )
    {
        cout<<"The element at index "<< iteration << "is:" << array[iteration] <<endl;
    }
    return 0;
}
```

Output-

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ array.cpp -o array
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./array
```

```
Welcome to DataFlair tutorials!
```

```
Enter the size of the array: 5
```

```
Enter the elements of the array:
```

```
12
```

```
6
```

```
32
```

```
11
```

```
8
```

```
The array is:
```

```
The element at index 0is:12
```

```
The element at index 1is:6
```

```
The element at index 2is:32
```

```
The element at index 3is:11
```

```
The element at index 4is:8
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

We declare a variable **size_of_array**, so that the user can specify the length of the array and a variable iteration to traverse the array with the help of a **for loop**. We know that the indexing of the array starts from 0. Hence, we initialize the variable ‘iteration’ to 0 till ‘iteration’ becomes less than the size of the array. Thereafter, we increment the variable ‘iteration’ so that the loop continues to run until the condition is satisfied.

[Loops in C and C++ – The Survival Guide for Every Beginner](#)

6. Multidimensional Arrays

The most popular form of **multidimensional arrays** is the two-dimensional arrays by which matrices can be implemented. Three-dimensional arrays and other higher

dimensions aren't as popularly used and hence we will restrict our discussion to simply two-dimensional arrays.

7. Passing Array to a Function

Before we begin our discussion on passing arrays to a function, it is important to understand [functions in C/C++](#).

Functions are nothing but a group of statements designed to serve a specific purpose. It helps the user to divide the code into smaller fragments instead of loading everything inside the main function. It even offers the benefit of code reusability.

An array can be passed to a function in 2 ways, namely:

1. Call by value

In this method, the actual parameter gets copied to the formal parameters. As the name itself suggests, the actual value is passed to the function. Any changes made to the formal parameters are not reflected in the actual parameters passed to the function

2. Call by reference

In this method, instead of the actual parameter, the address of the actual parameters is passed to the formal parameters. Hence, any changes made in the actual parameters are reflected in the formal parameters.

Let us consider some problems that involve the use of passing an array to the function using call by value and call by reference.

7.1 Example of Call By Value (Passing Single/Entire Element)

C program to pass a single element to the function

Here is a code in C that illustrates the use of call by value to pass an array to the function:

```
1. #include <stdio.h>
2. void output(int values)
3. {
4.     printf("The value of the 5th element of the array is: %d\n", values);
5. }
6. int main()
7. {
8.     printf("Welcome to DataFlair tutorials!\n\n");
9.     int array[6] = {10, 20, 30, 40, 50, 60};
10.    output(array[5]); // Passing the 5th element of the array
11.    return 0;
12. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

afun.c

```
#include <stdio.h>
void output(int values)
{
printf("The value of the 5th element of the array is: %d\n", values);
}

int main()
{
printf("Welcome to DataFlair tutorials!\n\n");
int array[6] = {10, 20, 30, 40, 50, 60};
output(array[5]); // Passing the 5th element of the array
return 0;
}
```

Output-

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc afun.c -o afun
dataflair@asus-System-Product-Name:~/Desktop$ ./afun
Welcome to DataFlair tutorials!
```

```
The value of the 5th element of the array is: 60
dataflair@asus-System-Product-Name:~/Desktop$ 
```

2. C++ program to pass a single element to the function

```
1. #include <iostream>
2. using namespace std;
3.
4. void output(int values)
5. {
6. cout<<"The value of the 5th element of the array is: "<< values << endl;
7. }
8.
9. int main()
10. {
11.
12. cout<<"Welcome to DataFlair tutorials!"<< endl << endl;
13. int array[6] = { 10, 20, 30, 40, 50, 60 };
14. output(array[5]); // Passing the 5th element of the array
15. return 0;
16. }
```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

afun.cpp

```
#include <iostream>
using namespace std;

void output(int values)
{
cout<<"The value of the 5th element of the array is: "<< values << endl;
}

int main()
{
cout<<"Welcome to DataFlair tutorials!"<< endl << endl;
int array[6] = {10, 20, 30, 40, 50, 60};
output(array[5]); // Passing the 5th element of the array
return 0;
}
```

Output-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

dataflair@asus-System-Product-Name:~/Desktop\$ g++ afun.cpp -o afun

dataflair@asus-System-Product-Name:~/Desktop\$./afun

Welcome to DataFlair tutorials!

The value of the 5th element of the array is: 60

dataflair@asus-System-Product-Name:~/Desktop\$ █

3. C program to pass the entire array to the function

```
1. #include <stdio.h>
2. void display( char c)
3. {
4.     printf("%c ", c);
5. }
6. int main()
7. {
8.     printf("Welcome to DataFlair tutorials!\n\n");
9.     int i;
10.    char vowels[] = { 'a', 'e', 'i', 'o', 'u' };
11.    for (i=0; i<5; i++)
12.    {
13.        display(vowels[i]);
14.    }
```

```
15. return 0;  
16. }
```

Code on Screen-

The screenshot shows a terminal window titled "dataflair@asus-System-Product-Name:~". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". Below the menu is the text "GNU nano 2.9.3". To the right of the menu, it says "afun2.c". The main area of the terminal contains the following C code:

```
#include <stdio.h>  
void display( char c)  
{  
    printf("%c ", c);  
}  
int main()  
{  
  
printf("Welcome to DataFlair tutorials!\n\n");  
  
int i;  
char vowels[] = {'a', 'e', 'i', 'o', 'u'};  
for (i=0; i<5; i++)  
{  
display(vowels[i]);  
}  
return 0;  
}
```

Output-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc afun2.c -o afun2
dataflair@asus-System-Product-Name:~/Desktop$ ./afun2
Welcome to DataFlair tutorials!
```

```
a e i o u dataflair@asus-System-Product-Name:~/Desktop$ 
```

4. C++ program to pass the entire array to the function

```
1. #include <iostream>
2. using namespace std;
3.
4. void display( char c)
5. {
6. cout<<c;
7. }
8. int main()
9. {
10.
11. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
12.
13. int i;
14. char vowels[] = {'a', 'e', 'i', 'o', 'u'};
15. for (i=0; i<5; i++)
```

```
16. {
17.     display(vowels[i])<<endl;
18. }
19. return 0;
20. }
```

Code-

The screenshot shows a terminal window with the following details:

- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: afun2.cpp
- Code content:

```
#include <iostream>
using namespace std;

void display( char c)
{
cout<<c;
}
int main()
{

cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

int i;
char vowels[] = {'a', 'e', 'i', 'o', 'u'};
for (i=0; i<5; i++)
{
display(vowels[i]);
}
return 0;
}
```
- Output: The terminal shows the output of the program, which is "Welcome to DataFlair tutorials!" followed by two newlines.

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ afun2.cpp -o afun2
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./afun2
```

```
Welcome to DataFlair tutorials!
```

```
aeioudataflair@asus-System-Product-Name:~/Desktop$ 
```

7.2 Example of Call by Reference

Call by reference to pass an array to the function in C-

```
1. #include <stdio.h>
2. void display( int *powers)
3. {
4.     printf("%d ", *powers);
5. }
6. int main()
7. {
8.     printf("Welcome to Dataflair tutorials!\n\n");
9.     int i;
10.    int array[] = {1, 2, 4, 9, 25, 36, 49, 64, 81, 100};
11.    for (i=0; i<10; i++)
```

```
12. {
13.     display(&array[i]); // Passing the address of array to the function
14. }
15. return 0;
16. }
```

Code on Screen-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name:
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: afun3.c
- Code content:

```
#include <stdio.h>
void display( int *powers)
{
printf("%d ", *powers);
}

int main()
{
printf("Welcome to Dataflair tutorials!\n\n");

int i;
int array[] = {1, 2, 4, 9, 25, 36, 49, 64, 81, 100};
for (i=0; i<10; i++)
{
display(&array[i]); // Passing the address of array to the function
}

return 0;
}
```
- Output:

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc afun3.c
dataflair@asus-System-Product-Name:~/Desktop$ ./afun3
1 2 4 9 25 36 49 64 81 100
Welcome to Dataflair tutorials!
```

Output-

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc afun3.c -o afun3
dataflair@asus-System-Product-Name:~/Desktop$ ./afun3
Welcome to Dataflair tutorials!
```

```
1 2 4 9 25 36 49 64 81 100 dataflair@asus-System-Product-Name:~/Desktop$ 
```

Call by reference to pass an array to the function in C++

```
1. #include <iostream>
2. using namespace std;
3.
4. void display( int *powers)
5. {
6. cout<<*powers;
7. }
8.
9. int main()
10. {
11.
12. cout<<"Welcome to Dataflair tutorials!"<<endl<<endl;
13.
14. int i;
15. int array[] = { 1, 2, 4, 9, 25, 36, 49, 64, 81, 100};
16. for (i=0; i<10; i++)
17. {
18. display(&array[i]); // Passing the address of array to the function
```

```
19. cout<<"\t";
20. }
21.
22. return 0;
23. }
```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

afun3.cpp

```
#include <iostream>
using namespace std;

void display( int *powers)
{
cout<<*powers;
}

int main()
{

cout<<"Welcome to Dataflair tutorials!"<<endl<<endl;

int i;
int array[] = {1, 2, 4, 9, 25, 36, 49, 64, 81, 100};
for (i=0; i<10; i++)
{
display(&array[i]); // Passing the address of array to the function
cout<<"\t";
}

return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ afun3.cpp -o afun3
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./afun3
```

```
Welcome to Dataflair tutorials!
```

```
1      2      4      9      25     36     49     64     81     100
```

```
dataflair@asu
```

8. Pointer to an Array

Before we begin our discussion on a pointer to an array, it is important to understand [pointers in C and C++](#).

Pointers are nothing but simple variables that allow you to access the memory address of another variable to which the pointer points to.

In C, you can access the value of an element of an array, or better, the entire array with the help of a pointer.

Example of Implementation of a pointer to an array in C:

```
1. #include<stdio.h>
```

```
2. int main()
3. {
4.     printf("Welcome to DataFlair tutorials!\n\n");
5.     int i;
6.     double array[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
7.     double *p;
8.     p = array;
9.     printf( "The array is:\n");
10.    for ( i = 0; i < 5; i++ )
11.    {
12.        printf("%0.2f\n", *(p + i) ); // Pointer to an array
13.    }
14.    return 0;
15. }
```

Code on Screen-

The screenshot shows a terminal window with a dark background. At the top, there is a menu bar with options: File, Edit, View, Search, Terminal, Help. To the right of the menu, the text "dataflair@asus-System-Product-Name" is displayed. Below the menu, the title bar shows "File Edit View Search Terminal Help" and "GNU nano 2.9.3" on the left, and "parray.c" on the right. The main area of the terminal contains the C code provided in the previous block. The code is syntax-highlighted, with keywords in blue and identifiers in green. The code prints a welcome message, initializes an array, prints its size, and then prints each element of the array using a pointer to the array's memory location.

```
include<stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int i;
    double array[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
    double *p;
    p = array;
    printf( "The array is:\n");
    for ( i = 0; i < 5; i++ )
    {
        printf("%0.2f\n", *(p + i) ); // Pointer to an array
    }
    return 0;
}
```

Output-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~/Desktop$ gcc parray.c -o parray
dataflair@asus-System-Product-Name:~/Desktop$ ./parray
Welcome to DataFlair tutorials!
```

The array is:

```
1.10
2.20
3.30
4.40
5.50
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Example of Implementation of a pointer to an array in C++:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int i;
10. double array[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
11. double *p;
12. p = array;
13. cout<<"The array is: "<<endl;
14.
15. for ( i = 0; i < 5; i++ )
16. {
17. cout<<*(p + i); // Pointer to an array
18. cout<<endl;
```

```
19. }
20. return 0;
21. }
```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

parray.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int i;
    double array[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
    double *p;
    p = array;
    cout<<"The array is: "<<endl;

    for ( i = 0; i < 5; i++ )
    {
        cout<<*(p + i); // Pointer to an array
        cout<<endl;
    }
    return 0;
}
```

Output-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

dataflair@asus-System-Product-Name:~/Desktop\$ g++ parray.cpp -o parray

^[[Adataflair@asus-System-Product-Name:~/Desktop\$./parray

Welcome to DataFlair tutorials!

The array is:

1.1
2.2
3.3
4.4
5.5

dataflair@asus-System-Product-Name:~/Desktop\$ □

It is important to note that the address of the first element is stored in ‘p’. Thereafter, you can easily access the elements of the array with the help of $*p$, $*(p+1)$, $*(p+2)$ to get the address of the 1st, 2nd, 3rd element respectively.

Key takeaway: The pointer to the array $*(p+i)$ will give you the value of the array with each iteration whereas $(p+i)$ will give you the address of the [array](#) with each iteration.

9. Summary

After completing Arrays in C and C++ tutorial, we got a firm understanding of Arrays, one of the most important and significant topics in C/C++ programming.

We discussed the meaning of arrays, how they are declared, the different data types associated with it, how they are initialized and accessed. Further, we carried on our discussion by giving a brief insight on the multidimensional array, array implementation in C and C++ like passing an array to a function and pointer to an array by intertwining the concepts of arrays and functions and arrays and pointers respectively.

Multi-dimensional Arrays in C/C++ (2D & 3D Arrays) – Unveil the Important Concepts

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 26, 2019

In our previous article, we talked about [arrays in C and C++](#). Now, it's time to uncover the secrets of Multi-dimensional Arrays in C/C++ programming language. As the name itself suggests, *arrays that possess more than one dimension are multi-dimensional arrays*. In this tutorial, we will restrict our focus on two-dimensional and three-dimensional arrays as they are the most commonly used and applied multi-dimensional arrays in C and C++.

Multi-dimensional Array in C/C++

The arrays of an array are known as multi-dimensional arrays. It is a pretty clear and comprehensible concept to grasp.

The diagram illustrates the concept of multi-dimensional arrays in C/C++. It features a 2D matrix and a 3D cube, both labeled with their respective dimensions and indices.

2-D Matrix:

	Column 1	Column 2	Column 3	Column 4
Row 1	Matrix[0][0]	Matrix[0][1]	Matrix[0][2]	Matrix[0][3]
Row 2	Matrix[1][0]	Matrix[1][1]	Matrix[1][2]	Matrix[1][3]
Row 3	Matrix[2][0]	Matrix[2][1]	Matrix[2][2]	Matrix[2][3]

3-D Cube:

A 3D cube is shown with its edges labeled with indices. The top face contains values 335, 227, 81, 126, 174, 88, and 88. The bottom face contains values 236, 44, 113, 113, 133, 313, 612, and 612. The vertical edges are labeled with 126, 174, 88, and 88.

Labels:

- 2-D** is positioned below the 2D matrix.
- 3-D** is positioned below the 3D cube.
- The **Data Flair** logo is present in the top left corner of the slide.

1. 2D Arrays

An array of an array is referred to as a two-dimensional array. In simpler words, it is a sequence of strings within a sequence of strings.

In order to understand the working of 2D arrays in C/C++, let us begin by discussing its basic syntax-

1.1 Declaration of 2D arrays in C/C++

return_type array_name [size of row][size of column];

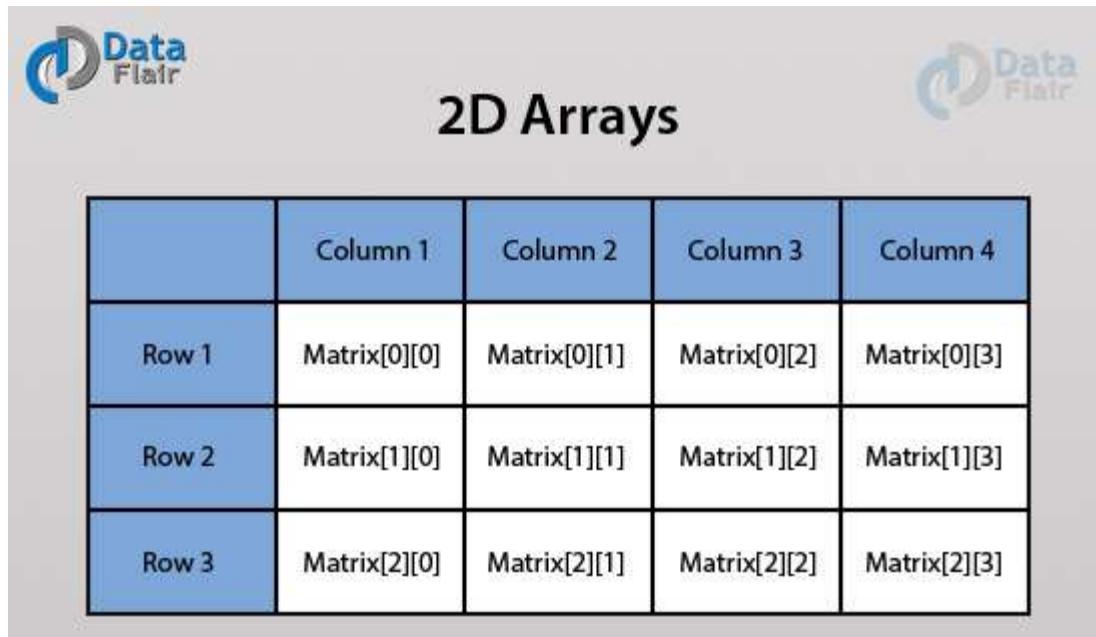
For instance,

int matrix [3] [3];

If you aren't aware of the Syntax of C, then you can't afford to miss checking out the [Basic Syntax Rules in C](#)

Here is a diagram, which would help you visualize how a matrix looks like:

The name of the **2-D array is a matrix** of the integer data type with 3 rows and 4 columns.



The diagram illustrates a 2D array or matrix with 3 rows and 4 columns. The columns are labeled "Column 1", "Column 2", "Column 3", and "Column 4". The rows are labeled "Row 1", "Row 2", and "Row 3". Each cell contains a label representing the element's memory address: Matrix[0][0], Matrix[0][1], Matrix[0][2], Matrix[0][3] in Row 1; Matrix[1][0], Matrix[1][1], Matrix[1][2], Matrix[1][3] in Row 2; and Matrix[2][0], Matrix[2][1], Matrix[2][2], Matrix[2][3] in Row 3.

	Column 1	Column 2	Column 3	Column 4
Row 1	Matrix[0][0]	Matrix[0][1]	Matrix[0][2]	Matrix[0][3]
Row 2	Matrix[1][0]	Matrix[1][1]	Matrix[1][2]	Matrix[1][3]
Row 3	Matrix[2][0]	Matrix[2][1]	Matrix[2][2]	Matrix[2][3]

Clearly, from the table, we observe that there would be 12 elements in the matrix, which is the product of the size of its rows and columns.

1.2 Initialization of 2D arrays in C/C++

Instead of simply declaring the multi-dimensional arrays, the C programming language gives you the provision to initialize it as well.

Let us consider an example of a 2D array for simplicity sake.

We can do it multiple ways-

- int array [2] [4] = { {2, 4, 0, -2}, {-1, 4, -7, 0} };
- int array [2] [4] = { 2, 4, 0, -2, -1, 4, -7, 0};

- int array [] [4] = { {2, 4, 0, -2}, {-1, 4, -7, 0} };

The 2D array would have 2 rows and 4 columns in each case.

Similarly, we can initialize multi-dimensional arrays in a similar manner of nth dimension

int array [][][]....nth[] = { { { ...nth { values } } }nth }

Don't forget to check [Data Types in C Language](#)

1.3 How to define 2D array in C?

If you are working with the for loop, you require 2 for loops to store the 2D array. One of the most common features of the 2D array is the implementation of matrices.

In mathematics, the matrix is a rectangular array of elements (numbers or expressions), that is represented with the help of rows and columns.

The basic concept behind a matrix is the same in programming. In addition to that, we can call it a grid useful in storing, displaying and manipulating data items.

Here is a simple code in C which illustrates how to take a matrix input and display it:

```

1. #include<stdio.h>
2. void main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     int matrix[10][10];
8.     int no_of_rows, no_of_columns;
9.     int iteration1, iteration2;
10.
11.    printf("Enter the number of rows of the matrix: ");
12.    scanf("%d",&no_of_rows);
13.    printf("Enter the number of columns of the matrix: ");
14.    scanf("%d",&no_of_columns);
15.
16.    printf("Enter the elements of the matrix:\n");
17.    for(iteration1 = 0; iteration1 < no_of_rows; iteration1++)
18.    {
19.        for(iteration2 = 0; iteration2 < no_of_columns; iteration2++)
20.        {
21.            scanf("%d", &matrix[iteration1][iteration2]);
22.        }
23.    }
24.    printf("The matrix is:\n");
25.    for(iteration1 = 0; iteration1 < no_of_rows; iteration1++)
26.    {
27.        for(iteration2 = 0; iteration2 < no_of_columns; iteration2++)
28.        {
29.            printf("%d\t",matrix[iteration1][iteration2]);
30.        }
31.        printf("\n");
32.    }
33. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

array2.c

```
#include<stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int matrix[10][10];
    int no_of_rows, no_of_columns;
    int iteration1, iteration2;

    printf("Enter the number of rows of the matrix: ");
    scanf("%d",&no_of_rows);
    printf("Enter the number of columns of the matrix: ");
    scanf("%d",&no_of_columns);

    printf("Enter the elements of the matrix:\n");
    for(iteration1 = 0; iteration1 < no_of_rows; iteration1++ )
    {
        for(iteration2 = 0; iteration2 < no_of_columns; iteration2++)
        {
            scanf("%d", &matrix[iteration1][iteration2]);
        }
    }
    printf("The matrix is:\n");
    for(iteration1 = 0; iteration1 < no_of_rows; iteration1++ )
    {
        for(iteration2 = 0; iteration2 < no_of_columns; iteration2++)
        {
            printf("%d\t",matrix[iteration1][iteration2]);
        }
        printf("\n");
    }
    return 0;
}
```

Output-

dataflair@admin4-H110M-H: ~/De

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~$ cd Desktop
dataflair@admin4-H110M-H:~/Desktop$ gcc array2.c -o array2
dataflair@admin4-H110M-H:~/Desktop$ ./array2
Welcome to DataFlair tutorials!

Enter the number of rows of the matrix: 3
Enter the number of columns of the matrix: 3
Enter the elements of the matrix:
1 2 3 4 5 6 7 8 9
The matrix is:
1      2      3
4      5      6
7      8      9
dataflair@admin4-H110M-H:~/Desktop$ █
```

1.4 How to define 2D array in C++?

Here is a simple code in C++ that illustrates how to take a matrix input and display it:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int matrix[10][10];
10. int no_of_rows, no_of_columns;
11. int iteration1, iteration2;
12.
13. cout<<"Enter the number of rows of the matrix: ";
14. cin>>no_of_rows;
15. cout<<"Enter the number of columns of the matrix: ";
16. cin>>no_of_columns;
```

```
17.  
18. cout<<"Enter the elements of the matrix: "<<endl;  
19. for(iteration1 = 0; iteration1 < no_of_rows; iteration1++)  
20. {  
21.     for(iteration2 = 0; iteration2 < no_of_columns; iteration2++)  
22.     {  
23.         cin>>matrix[iteration1][iteration2];  
24.     }  
25. }  
26. cout<<"The matrix is: "<<endl;  
27. for(iteration1 = 0; iteration1 < no_of_rows; iteration1++)  
28. {  
29.     for(iteration2 = 0; iteration2 < no_of_columns; iteration2++)  
30.     {  
31.         cout<< "\t" <<matrix[iteration1][iteration2];  
32.     }  
33.     cout<<endl;  
34. }  
35. return 0;  
36. }
```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

array2.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int matrix[10][10];
    int no_of_rows, no_of_columns;
    int iteration1, iteration2;

    cout<<"Enter the number of rows of the matrix: ";
    cin>>no_of_rows;
    cout<<"Enter the number of columns of the matrix: ";
    cin>>no_of_columns;
    cout<<"Enter the elements of the matrix: "<<endl;
    for(iteration1 = 0; iteration1 < no_of_rows; iteration1++)
    {
        for(iteration2 = 0; iteration2 < no_of_columns; iteration2++)
        {
            cin>>matrix[iteration1][iteration2];
        }
    }
    cout<<"The matrix is: "<<endl;
    for(iteration1 = 0; iteration1 < no_of_rows; iteration1++)
    {
        for(iteration2 = 0; iteration2 < no_of_columns; iteration2++)
        {
            cout<< "\t" <<matrix[iteration1][iteration2];
        }
        cout<<endl;
    }
    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ array2.cpp -o array2
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./array2
```

```
Welcome to DataFlair tutorials!
```

```
Enter the number of rows of the matrix: 3
```

```
Enter the number of columns of the matrix: 3
```

```
Enter the elements of the matrix:
```

```
1 2 3 4 5 6 7 8 9
```

```
The matrix is:
```

1	2	3
4	5	6
7	8	9

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

2. 3D Arrays

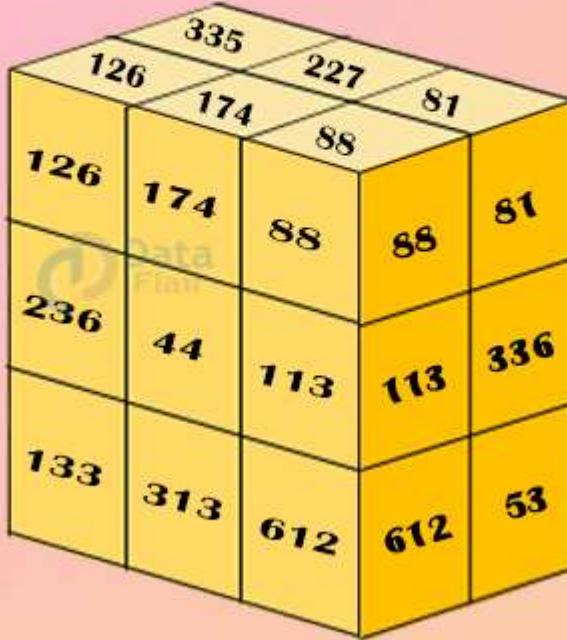
Three-dimensional arrays in C/C++ are referred to as an array of arrays. Its syntax is similar to a 1D or 2D array:

return_type array_name [size-1] [size-2][size-3];

For instance,

int sample [3] [2] [3] ;

3D Arrays



Clearly, from the diagram, we observe that there would be 18 elements in the 3D [array](#), which is the product of dimensions ($3*2*3$) of the array.

If you are working with the for loop, you require 3 for loops to store the 3D array.

2.1 How to define 3D arrays in C?

Here is a simple code in C which illustrates the use of a three-dimensional matrix:

```
1. #include <stdio.h>
2. void main()
3. {
4.
5. printf("Welcome to DataFlair tutorials!\n\n");
6.
7. int i, j, k, sample[3][2][3], size;
8. size=3*2*3; // Size of the 3D array is the product of size of each array
9. printf("Enter %d elements: \n",size);
10.
11. for(i = 0; i < 3; ++i)
12. {
13.     for (j = 0; j < 2; ++j)
14.     {
15.         for(k = 0; k < 3; ++k )
16.         {
17.             scanf("%d", &sample[i][j][k]);
18.         }
19.     }
20. }
21. printf("The values are:\n\n"); // To display the values of elements according to their index
22. for(i = 0; i < 3; i++)
```

```
23. {
24.     for(j = 0; j < 2; j++)
25.     {
26.         for(k = 0; k < 3; k++)
27.         {
28.             printf("sample[%d][%d][%d] = %d\n", i, j, k, sample[i][j][k]);
29.         }
30.     }
31. }
32. }
```

Code on Screen-

dataflair@asus-System-Product-Name:~

File Edit View Search Terminal Help

GNU nano 2.9.3

mat3d.c

```
#include <stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int i, j, k, sample[3][2][3], size;
    size=3*2*3; // Size of the 3D array is the product of size of each array
    printf("Enter %d elements: \n",size);

    for(i = 0; i < 3; ++i)
    {
        for (j = 0; j < 2; ++j)
        {
            for(k = 0; k < 3; ++k )
            {
                scanf("%d", &sample[i][j][k]);
            }
        }
    }
    printf("The values are:\n\n"); // To display the values of elements according to the
    for(i = 0; i < 3; i++)
    {
        for (j = 0; j < 2; j++)
        {
            for(k = 0; k < 3; k++)
            {
                printf("sample[%d][%d][%d] = %d\n", i, j, k, sample[i][j][k]);
            }
        }
    }
    return 0;
}
```

Output-

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc mat3d.c -o mat3d
dataflair@admin4-H110M-H:~/Desktop$ ./mat3d
Welcome to DataFlair tutorials!

Enter 18 elements:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
The values are:

sample[0][0][0] = 1
sample[0][0][1] = 2
sample[0][0][2] = 3
sample[0][1][0] = 4
sample[0][1][1] = 5
sample[0][1][2] = 6
sample[1][0][0] = 7
sample[1][0][1] = 8
sample[1][0][2] = 9
sample[1][1][0] = 10
sample[1][1][1] = 11
sample[1][1][2] = 12
sample[2][0][0] = 13
sample[2][0][1] = 14
sample[2][0][2] = 15
sample[2][1][0] = 16
sample[2][1][1] = 17
sample[2][1][2] = 18
dataflair@admin4-H110M-H:~/Desktop$ 
```

2.2 How to define 3D arrays in C++?

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int i, j, k, sample[3][2][3], size;
10. size=3*2*3; // Size of the 3D array is the product of size of each array
11. cout<<"Enter "<< size << " elements "<<endl;
12.
13. for(i = 0; i < 3; ++i)
14. {
15.     for (j = 0; j < 2; ++j)
16. {
```

```

17. for(k = 0; k < 3; ++k )
18. {
19.     cin>>sample[i][j][k];
20. }
21. }
22. }
23. cout<<"The values are: "<<endl; // To display the values of elements according to their index
24. for(i = 0; i < 3; i++)
25. {
26.     for (j = 0; j < 2; j++)
27.     {
28.         for(k = 0; k < 3; k++)
29.         {
30.             cout<<"sample [ "<<i <<" ][ "<<j <<" ] [ " <<k <<" ] = "<< sample[i][j][k] <<endl;
31.         }
32.     }
33. }
34. return 0;
35. }

```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3 mat3d.cpp

```

#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int i, j, k, sample[3][2][3], size;
    size=3*2*3; // Size of the 3D array is the product of size of each array
    cout<<"Enter "<< size << " elements "<<endl;

    for(i = 0; i < 3; ++i)
    {
        for (j = 0; j < 2; ++j)
        {
            for(k = 0; k < 3; ++k )
            {
                cin>>sample[i][j][k];
            }
        }
    }

    cout<<"The values are: "<<endl; // To display the values of elements according to their index
    for(i = 0; i < 3; i++)
    {
        for (j = 0; j < 2; j++)
        {
            for(k = 0; k < 3; k++)
            {
                cout<<"sample [ "<<i <<" ][ "<<j <<" ] [ " <<k <<" ] = "<< sample[i][j][k] <<endl;
            }
        }
    }

    return 0;
}

```

Output-

dataflair@asus-System-Product-Name:

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ g++ mat3d.cpp -o mat3d
dataflair@asus-System-Product-Name:~/Desktop$ ./mat3d
Welcome to DataFlair tutorials!

Enter 18 elements
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
The values are:
sample [ 0 ][ 0 ] [ 0 ] = 1
sample [ 0 ][ 0 ] [ 1 ] = 2
sample [ 0 ][ 0 ] [ 2 ] = 3
sample [ 0 ][ 1 ] [ 0 ] = 4
sample [ 0 ][ 1 ] [ 1 ] = 5
sample [ 0 ][ 1 ] [ 2 ] = 6
sample [ 1 ][ 0 ] [ 0 ] = 7
sample [ 1 ][ 0 ] [ 1 ] = 8
sample [ 1 ][ 0 ] [ 2 ] = 9
sample [ 1 ][ 1 ] [ 0 ] = 10
sample [ 1 ][ 1 ] [ 1 ] = 11
sample [ 1 ][ 1 ] [ 2 ] = 12
sample [ 2 ][ 0 ] [ 0 ] = 13
sample [ 2 ][ 0 ] [ 1 ] = 14
sample [ 2 ][ 0 ] [ 2 ] = 15
sample [ 2 ][ 1 ] [ 0 ] = 16
sample [ 2 ][ 1 ] [ 1 ] = 17
sample [ 2 ][ 1 ] [ 2 ] = 18
dataflair@asus-System-Product-Name:~/Desktop$
```

Summary

In this tutorial, we discussed 2D and 3D arrays in C/C++, how are they declared, their indexing, and how are they stored and displayed. It has various applications in algorithms of:

- Bubble sort
- Binary search
- Matrix manipulations that is Strassen's matrix multiplication and many more.

Therefore, learning about multi-dimensional arrays in C/C++ would give you an upper edge over programmers who are alien to the concept of multi-dimensional arrays.

Variables in C and C++ | A Complete Guide for Beginners

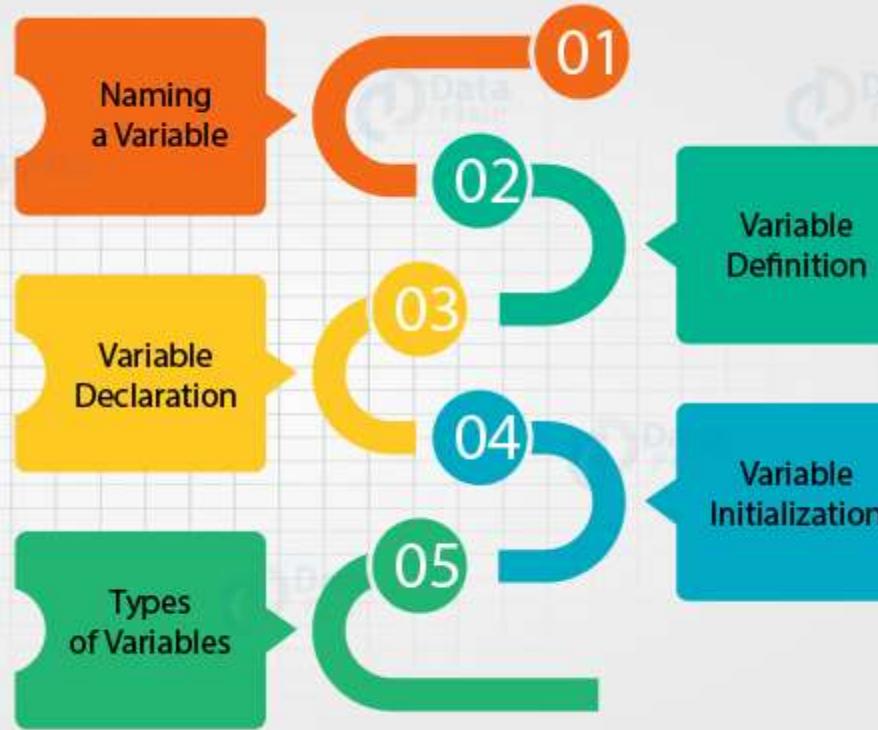
BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 26, 2019

When we hear about variables, we tend to think about mathematical formulas and questions in which variables are unknown values or quantities limited to some numbers. But, in the C/C++ programming language, variables connote a different meaning. In mathematical terms, we use representations of variables such as x or y that indicates an unknown value that we are supposed to find. Here, *variables in the C and C++ programming language are the basic units, which help us to build C Programs.* So, without wasting time, start exploring C/C++ variables.

What are Variables in C and C++?

A C/C++ program performs many tasks and operations that help you to resolve many problems by storing values into computer memory. But, how does the compiler understand the names given to these values? A variable helps to specify the existence of these values by defining and declaring them.

Variables in



In simple words, *a variable is a storage space associated with a unique name to identify them*. When you want to store some data on your system in the computer memory, is it possible for you to be able to remember these memory addresses? The answer is no, and that is the reason why we use variables.

Variables in C/C++ programming help us to store values depending upon the size of the variable. With the help of variables, we can decide what amount and type of data store in a variable. When you assign a data type and name to some space in the memory, variables are defined.

Variables reserve some memory in the storage space, that you can access later in the program. By declaring a variable, you inform the operating system to reserve memory indicated by some name. i.e. variable_name.

Enhance your fundamental skills with [Operators in C](#)

Naming a Variable in C/C++

You need to follow some rules, before naming a variable in C and C++:

1. A variable must not start with a digit.
2. A variable can begin with an alphabet or an underscore.
3. Variables in C and C++ are case-sensitive which means that uppercase and lowercase characters are treated differently.
4. A variable must not contain any special character or symbol.
5. White spaces are not allowed while naming a variable.
6. Variables should not be of the same name in the same scope.

7. A variable name cannot be a keyword.
8. The name of the variable should be unique.

Let's see some examples of both valid and invalid variable names.

Valid variable names

ticketdata
_ticketdata
ticket_data

Invalid variable names

56ticketdata
ticket@data
ticket data

Variable Definition

A variable definition in C and C++ defines the variable name and assigns the data type associated with it in some space in computer memory. After giving its definition, this variable can be used in the program depending upon the scope of that variable.

By defining a variable, you indicate the name and data type of the variable to the compiler. The compiler allocates some memory to the variable according to its size specification.

[Learn Data Types in C and C++ with Example in Just 4 mins.](#)

Rules for Defining Variables in C and C++

1. Must contain data_type of that variable.

Example: int start;

float width;

char choice;

2. The variable name should follow all the rules of the naming convention.
3. After defining the variable, terminate the statement with a semicolon otherwise it will generate a termination error. **Example:** int sum;
4. The variable with the same data type can work with a single line definition. **Example:** float height, width, length;

Defining Variables in C and C++ with Example

```
1. int var;
```

Here, a variable of integer type with the variable name var is defined. This variable definition allocates memory in the system for var.

Another example,

```
1. char choice ;
```

When we define this variable named choice, it allocates memory in the storage space according to the type of *data type in C*, i.e., character type.

Variable Declaration

There is a huge difference between defining a variable and declaring a variable.

By declaring a variable in C and C++, we simply tell the compiler that this variable exists somewhere in the program. But, the declaration does not allocate any memory for that variable.

Declaration of variable informs the compiler that some variable of a specific type and name exists. The definition of variable allocates memory for that variable in the program.

So, it is safe to say that the variable definition is a combination of declaration and memory allocation. A variable declaration can occur many times but variable definition occurs only once in a program, or else it would lead to wastage of memory.

Variable Initialization

Variable initialization means *assigning some value to that variable*. The initialization of a variable and declaration can occur in the same line.

```
1. int demo = 23;
```

By this, you initialize the variable demo for later use in the program.

Before we start to discuss types of variables, you should know [Functions and their importance in C](#)

Types of Variables in C and C++

There are 5 types of Variables in C/C++; let's discuss each variable with example.

- 1 Local Variables
- 2 Global Variables
- 3 Static Variables
- 4 Automatic Variables
- 5 External Variables

Types of Variable in



1. Local Variables

The scope of local variables lies only within the function or the block of code. These variables stay in the memory till the end of the program.

Example of Local Variable in C

```
1. #include <stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     int result = 5; //local variable
8.     printf("The result is %d \n", result);
9.     return 0;
10. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

localv.c

```
#include <stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int result = 5; //local variable
    printf("The result is %d \n", result);
    return 0;
}
```

Output-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

dataflair@asus-System-Product-Name:~/Desktop\$ gcc localv.c -o localv

dataflair@asus-System-Product-Name:~/Desktop\$./localv

Welcome to DataFlair tutorials!

The result is 5

dataflair@asus-System-Product-Name:~/Desktop\$

Example of Local Variable in C++

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
7.
8.     int result = 5; //local variable
9.     cout<<"The result is: "<< result<<endl;
10.    return 0;
11. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

localv.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int result = 5; //local variable
    cout<<"The result is: "<< result<<endl;
    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ g++ localv.cpp -o localv  
dataflair@asus-System-Product-Name:~/Desktop$ ./localv  
Welcome to DataFlair tutorials!  
  
The result is: 5  
dataflair@asus-System-Product-Name:~/Desktop$
```

2. Global Variables

A global variable has a global scope, that is, *this variable is valid until the end of the program*. These variables are available to all the functions in that program.

Let us now see an example to understand the difference between local and global variables in C and C++.

Example of Global Variables in C

```
1. #include <stdio.h>  
2. int sumf();  
3. int sum = 2; //global variable  
4. int main ()  
5. {  
6.  
7. printf("Welcome to DataFlair tutorials!\n\n");  
8.  
9. int result =5; //local variable  
10. sumf();  
11. return 0;
```

```
12. }
13. int sumf()
14. {
15.     printf("\n Sum is %d \n", sum);
16.     printf("\n The result is %d \n", result);
17. }
```

Code on Screen-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

globalv.c

```
#include <stdio.h>
int sumf();
int sum = 2; //global variable
int main ()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int result =5; //local variable
    sumf();
    return 0;
}
int sumf()
{
    printf("\n Sum is %d \n", sum);
    printf("\n The result is %d \n", result);
}
```

When we compile this program,

As the variable result is not defined globally, that is why the code is not compiled, and an error occurs when we use this variable out of that function.

Let us correct it, by making the result as the global variable in C/C++.

```
1. #include <stdio.h>
2. int sumf();
3. int sum = 2; //global variable
4. int result =5; //global variable
5. int main ()
6. {
```

```
7.  
8. printf("Welcome to DataFlair tutorials!\n\n");  
9.  
10. sumf();  
11. return 0;  
12. }  
13. int sumf()  
14. {  
15. printf("\n Sum is %d \n", sum);  
16. printf("\n The result is %d \n", result);  
17. }
```

Code on Screen-

The screenshot shows a terminal window with a dark background. At the top, there's a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. Below the menu, it says 'GNU nano 2.9.3' on the left and 'globalv.c' on the right. The main area contains the following C code:

```
#include <stdio.h>  
int sumf();  
int sum = 2; //global variable  
int result =5; //global variable  
int main ()  
{  
  
printf("Welcome to DataFlair tutorials!\n\n");  
  
sumf();  
return 0;  
}  
int sumf()  
{  
printf("\n Sum is %d \n", sum);  
printf("\n The result is %d \n", result);  
}
```

At the top right of the terminal window, the text 'dataflair@asus-System-Product-Name' is visible.

Output-

dataflair@asus-System-Product-Name:

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ gcc globalv.c -o glovalv
dataflair@asus-System-Product-Name:~/Desktop$ ./glovalv
Welcome to DataFlair tutorials!

Sum is 2

The result is 5
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Example of Global Variables in C++

Let us take the same example in C++ to understand the difference between a local and a global variable:

```
1. #include <iostream>
2. using namespace std;
3.
4. int sumf();
5. int sum = 2; // global variable
6. int main ()
7. {
8.
9. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;;
10. int result =5; //local variable
11. sumf();
12. return 0;
13. }
14. int sumf()
15. {
```

```
16. cout<<"Sum is: "<< sum << endl;
17. cout<<"The result is: "<< result << endl;
18. }
```

Error-

The screenshot shows a terminal window titled "globall.cpp" with the path "~/Desktop". The code in the terminal is identical to the one above, but it results in an error:

```
#include <iostream>
using namespace std;

int sumf();
int sum = 2; // global variable
int main ()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;;
    int result =5; //local variable
    sumf();
    return 0;
}
int sumf()
{
    cout<<"Sum is: "<< sum << endl;
    cout<<"The result is: "<< result << endl;
}
```

The terminal output shows the command "dataflair@asus-System-Pr" followed by "globall.cpp: In function '_main'". The error message "globall.cpp:17:27: error: variable 'sum' has already been defined" is displayed, along with the line "cout<<"The result is: "

In this way, we can implement a global variable in C++

```
1. #include <iostream>
2. using namespace std;
3.
4. int sumf();
5. int sum = 2; //global variable
6. int result =5; //global variable
7. int main ()
8. {
9.     cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;;
10.    sumf();
11.    return 0;
12. }
13. int sumf()
14. {
15.     cout<<"Sum is: "<< sum << endl;
```

```
16. cout<<"The result is: "<< result <<endl;
17. }
```

Code on Screen-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

GNU nano 2.9.3

globalv.cpp

```
#include <iostream>
using namespace std;

int sumf();
int sum = 2; //global variable
int result =5; //global variable
int main ()
{
cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
sumf();
return 0;
}
int sumf()
{
cout<<"Sum is: "<< sum <<endl;
cout<<"The result is: "<< result <<endl;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ g++ globalv.cpp -o globalv  
dataflair@asus-System-Product-Name:~/Desktop$ ./globalv  
Welcome to DataFlair tutorials!  
  
Sum is: 2  
The result is: 5  
dataflair@asus-System-Product-Name:~/Desktop$ ]
```

3. Static Variables

Static variable retains its value within the function calls. ‘static’ keyword is used to define a static variable. A static variable can preserve its value, and you can not initialize a static variable again.

Example of Static Variables in C

Let us now see how a local variable and static variable are different.

```
1. #include <stdio.h>  
2. void statf();  
3. int main ()  
4. {  
5.  
6. printf("Welcome to DataFlair tutorials!\n\n");  
7.  
8. int i;  
9. for(i = 0; i < 5; i++)
```

```
10. {
11.     statf();
12. }
13. }
14. void statf()
15. {
16.     int a = 20; //local variable
17.     static int b = 20; //static variable
18.     a = a + 1;
19.     b = b + 1;
20.     printf("\n Value of a %d\t,Value of b %d\n",a ,b);
21. }
```

Code on Screen-

dataflair@asus-System-Product-Name:

```
File Edit View Search Terminal Help
GNU nano 2.9.3                                     static.c

#include <stdio.h>
void statf();
int main ()
{
printf("Welcome to DataFlair tutorials!\n\n");
int i;
for(i = 0; i < 5; i++)
{
statf();
}
void statf()
{
int a = 20; //local variable
static int b = 20; //static variable
a = a + 1;
b = b + 1;
printf("\n Value of a %d\t,Value of b %d\n",a ,b);
}
```

Output-

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~/Desktop$ touch static.c
dataflair@asus-System-Product-Name:~/Desktop$ gcc static.c -o static
dataflair@asus-System-Product-Name:~/Desktop$ ./static
Welcome to DataFlair tutorials!
```

Value of a 21 ,Value of b 21

Value of a 21 ,Value of b 22

Value of a 21 ,Value of b 23

Value of a 21 ,Value of b 24

Value of a 21 ,Value of b 25

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Example of Static Variables in C++

Let us now see how a local variable and static variable are different in C++:

```
1. #include <iostream>
2. using namespace std;
3. void statf();
4. int main ()
5. {
6. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
7. int i;
8. for(i = 0; i < 5; i++)
9. {
10. statf();
11. }
12. }
13. void statf()
14. {
15. int a = 20; //local variable
16. static int b = 20; //static variable
17. a = a + 1;
```

```
18. b = b + 1;
19. cout<<"Value of a: " << a << ",Value of b: "<< b << endl;
20. }
```

Code on Screen-

dataflair@asus-System-Product-Name

File Edit View Search Terminal Help

GNU nano 2.9.3

static.cpp

```
#include <iostream>
using namespace std;

void statf();
int main ()
{
cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
int i;
for(i = 0; i < 5; i++)
{
statf();
}
}
void statf()
{
int a = 20; //local variable
static int b = 20; //static variable
a = a + 1;
b = b + 1;
cout<<"Value of a: " << a << ",Value of b: "<< b << endl;
}
```

Output-

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ g++ static.cpp -o static
dataflair@asus-System-Product-Name:~/Desktop$ ./static
Welcome to DataFlair tutorials!

Value of a: 21 ,Value of b: 21
Value of a: 21 ,Value of b: 22
Value of a: 21 ,Value of b: 23
Value of a: 21 ,Value of b: 24
Value of a: 21 ,Value of b: 25
dataflair@asus-System-Product-Name:~/Desktop$ 
```

With every function call, static variable uses the preserved value and increments the value of a [variable](#), whereas a local variable re-initializes the value every time function calls takes place.

4. Automatic Variables

The variable declared inside a block is called an automatic variable.

The automatic variables in C are different from the local variables, automatic variable allocates memory upon the entry to that block and frees the occupied space when the control exits from the block.

‘auto’ keyword defines an automatic variable.

Example

```
1. auto int var = 39;
```

5. External Variables

We use ‘extern’ keyword to increase the visibility of the program. An extern variable is available to other files too.

The difference between global and extern variable is, a global variable is available anywhere within the file or program in which global variable declares but an extern variable is available for other files too.

Note: Variables with keyword ‘extern’ are only declared; they are not defined. Also, initialization of extern keywords can also be considered as its definition.

```
1. extern int var;
```

It means that extern variable var is available or valid in the program and in the other files too.

Note: Declaration of var takes place and not the definition.

Summary

In this tutorial, we focused on all significant points related to variables. Follow these rules for variables in C and C++,

1. Every variable uses some memory in the storage space when it is defined not declared.
2. Initializing a variable can be done along with a definition.
3. There are various types of variables that C support and these variables are categorized based on their scope.

5 Types of Constants in C and C++ and How they're Different from Literals

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 25, 2019

Have you ever thought, what are constants and why they are an important part of the programming world? We have got all the answers to your queries. *Constants in C and C++ programming are nothing but fixed values that cannot be altered throughout the program run. These fixed values are also called as literals.*

In layman language, *we can use the terms constants and literals interchangeably*. But, we will highlight the key difference between the two in this tutorial.

Before we start, you must be aware of the [Variables in C](#)

Now, let us acknowledge the significance of constants and literals in C and C++ by considering the following problem:

There are certain situations where variables do not change their value, let's say, the value of pi, approximately equal to 3.14159 is constant and it can never change which is a universal fact. We might encounter several situations, where we would require pi for mathematical calculations. Similarly, we can assign constant values to several variables according to our convenience.

1. Variables and Constants in C and C++

It is important to note that a variable connotes a different meaning in programming and mathematics. A *variable is nothing but a value that we can store in computer memory*. We can easily change its value during run-time. In contrast to that, constants never change their value throughout the program run. Constants can hold any of the [data types available in C and C++](#).

2. Declare or Define Constants

We can assign C/C++ constant value to a variable in two ways:

1. **Using #define, a preprocessor directive:** We have already discussed #define in detail in preprocessors.
2. **Using the keyword const:** It is similar to variable declaration except that we should add the keyword “const” prior to it. It is important to assign a value to the constant as soon as we declare it.

Learn more about [Preprocessors in C](#), get aware of the fact that is it worth or not?

2.1 Declare or Define Constants in C

Here is a program in C that illustrates what happens when we try to modify the value of a constant:

```
1. #include<stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     const float pi = 3.14;
8.     const float e = 2.71;
9.
10.    pi = 3.14159;
11.    printf("The value of pi is: %f", pi);
12.    return 0;
13. }
```

Output-

The screenshot shows a terminal window with the following text:

```
const.c: In function 'main':  
const.c:10:4: error: assignment  
      pi = 3.14159;  
           ^
```

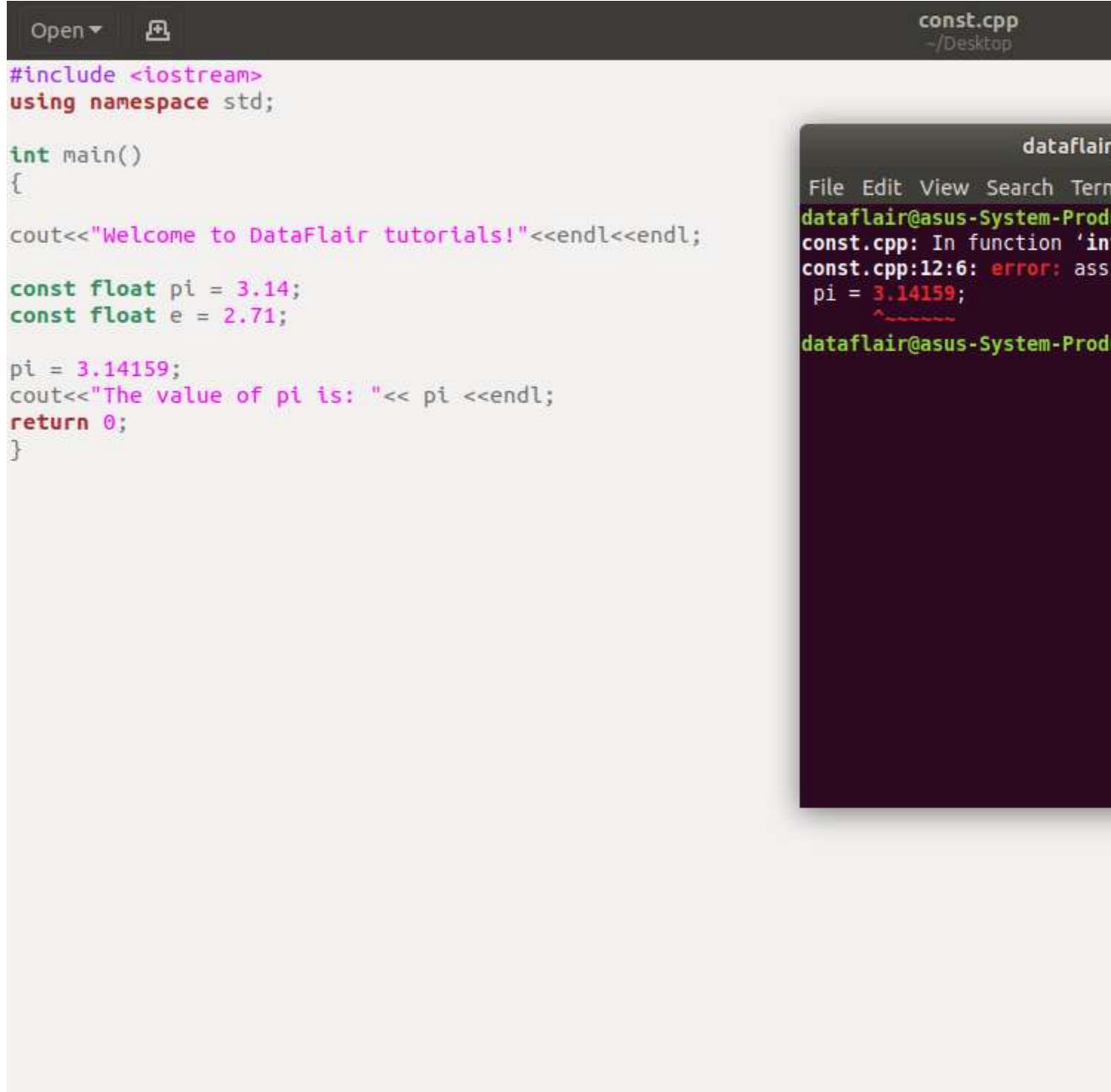
2.1 Declare or Define Constants in C++

The same issue arises when we try to modify the value of a constant:

1. #include <iostream>
2. using namespace std;
- 3.
4. int main()
5. {
- 6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
- 8.
9. const float pi = 3.14;
10. const float e = 2.71;
- 11.
12. pi = 3.14159;
13. cout<<"The value of pi is: "<< pi << endl;

```
14. return 0;
15. }
```

Error-



The screenshot shows a terminal window titled "const.cpp" located in the "Desktop" directory. The terminal window has a dark background and white text. At the top, it says "File Edit View Search Term". Below that, it shows the command "dataflair@asus-System-Prod:~/Desktop\$". The main text area of the terminal shows the following code and its error:

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
    const float pi = 3.14;
    const float e = 2.71;

    pi = 3.14159;
    cout<<"The value of pi is: "<< pi <<endl;
    return 0;
}
```

The terminal output shows an error message: "const.cpp: In function 'int main()': const.cpp:12:6: error: assignment of read-only variable 'pi'". The line "pi = 3.14159;" is highlighted in red, with the assignment part "pi = 3.14159;" underlined.

3. Constants vs Literals in C/C++

As discussed earlier, constants and literals may be used interchangeably but there is a slight difference between the two.

Let us consider a simple example to better understand it.

In India, the legal voting age is 18.

Therefore, we define,

const voting_age = 18;

Here, 18 is a literal, a value that is expressed as itself whereas a constant can be considered a data type that is substituted in place of a literal to enhance the functionality of the code.

Now, we wish to print the message, “ You are allowed to vote! ” for people of age greater than or equal to 18.

Here is a segment of code in C in accordance with the statement above.

```
if(age >= voting_age)
{
printf(" You are allowed to vote! ");
}
```

Here, the identifier `voting_age` is constant.

Here is a segment of code in C++ in accordance with the statement above.

```
if(age >= voting_age)
{
cout<<" You are allowed to vote! "<<endl;
}
```

Here, the identifier `voting_age` is constant.

4. Types of Constants in C and C++

In the C/C++, there are 5 different types of constants depending upon their *Data type*:



4.1 Integer Constants

As the name itself suggests, an integer constant is an integer with a fixed value, that is, it cannot have fractional value like 10, -8, 2019.

For example,

const signed int limit = 20;

We may use different combinations of U and L suffixes to denote unsigned and long modifiers respectively, keeping in mind that its repetition does not occur.

We can further classify it into three types, namely:

- **Decimal number system constant:** It has the base/radix 10. (0 to 9)
For example, 55, -20, 1.
In the decimal number system, no prefix is used.
- **Octal number system constant:** It has the base/radix 8. (0 to 7)
For example, 034, 087, 011.
In the octal number system, 0 is used as the prefix.
- **Hexadecimal number system constant:** It has the base/radix 16. (0 to 9, A to F)
In the hexadecimal number system, ox is used as the prefix. C language gives you the provision to use either uppercase or lowercase alphabets to represent hexadecimal numbers.

4.2 Floating or Real Constants

We use a floating-point constant to represent all the real numbers on the number line, which includes all fractional values.

For instance,

```
const long float pi = 3.14159;
```

We may represent it in 2 ways:

- **Decimal form:** The inclusion of the decimal point (.) is mandatory.
For example, 2.0, 5.98, -7.23.
- **Exponential form:** The inclusion of the signed exponent (either e or E) is mandatory.
For example, the universal gravitational constant G = 6.67 x 10-11 is represented as 6.67e-11 or 6.67E-11.

4.3 Character Constants

Character constants are used to assign a fixed value to characters including alphabets and digits or special symbols enclosed within single quotation marks(‘ ’).

Each character is associated with its specific numerical value called the ASCII (American Standard Code For Information Interchange) value.

Apart from these values, there is a set in C known as *Escape Sequences*

For example, ‘+’, ‘A’, ‘d’.

4.4 String Constants

A string constant is an array of characters that has a fixed value enclosed within double quotation marks (“ ”).

For example, “DataFlair”, “Hello world!”

4.5 Enumeration Constants

Enumeration constants are user-defined data-types in C with a fixed value used to assign names to integral constants.

For example,

```
enum rainbow = { Violet, Indigo, Blue, Green, Yellow, Orange, Red }
```

The enumeration rainbow has integral values as:

Violet : 0

Indigo: 1

Blue: 2

Green : 3

Yellow: 4

Orange: 5

Red: 6

How to use Constant in C?

Here is a code in C that illustrates the use of some constants:

```
1. #include<stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     const int value = 4;
8.     const float marks = 98.98;
9.     const char grade = 'A';
10.    const char name[30] = "DataFlair";
11.
12.    printf("The constant int value is: %d\n",value);
13.    printf("The constant floating-point marks is: %f\n", marks);
14.    printf("The constant character grade is: %c\n", grade);
15.    printf("The constant string name is: %s\n",name);
16.    return 0;
17. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

const2.c

```
#include<stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    const int value = 4;
    const float marks = 98.98;
    const char grade = 'A';
    const char name[30] = "DataFlair";

    printf("The constant int value is: %d\n",value);
    printf("The constant floating-point marks is: %f\n", marks);
    printf("The constant character grade is: %c\n", grade);
    printf("The constant string name is: %s\n",name);
    return 0;
}
```

Output-

```
File Edit View Search Terminal Help
```

```
dataflair@admin4-H110M-H:~/Desktop$ gcc const2.c -o const2
```

```
dataflair@admin4-H110M-H:~/Desktop$ ./const2
```

```
Welcome to DataFlair tutorials!
```

```
The constant int value is: 4
```

```
The constant floating-point marks is: 98.980003
```

```
The constant character grade is: A
```

```
The constant string name is: DataFlair
```

```
dataflair@admin4-H110M-H:~/Desktop$ █
```

How to use a Constant in C++?

Here is a code in C++ that illustrates the use of some constants:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. const int value = 4;
10. const float marks = 98.98;
11. const char grade = 'A';
12. const char name[30] = "DataFlair";
```

```
13.  
14. cout<<"The constant int value is: "<< value <<endl;  
15. cout<<"The constant floating-point marks is: "<< marks <<endl;  
16. cout<<"The constant character grade is: "<< grade <<endl;  
17. cout<<"The constant string name is: "<< name <<endl;  
18.  
19. return 0;  
20. }
```

Code-

The screenshot shows a terminal window with a dark background. At the top, it displays the user's name and system information: **dataflair@asus-System-Product-Name:**. Below this is a menu bar with options: File, Edit, View, Search, Terminal, Help. Underneath the menu bar, it says **GNU nano 2.9.3** and **const2.cpp**. The main area of the terminal contains the C++ code for the program. The code includes #include <iostream>, using namespace std;, and defines constants for int, float, char, and string types. It then uses cout to print a welcome message and the values of these constants. The output of the program is shown below the code, starting with "Welcome to DataFlair tutorials!".

```
File Edit View Search Terminal Help  
GNU nano 2.9.3 const2.cpp  
#include <iostream>  
using namespace std;  
int main()  
{  
  
cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;  
  
const int value = 4;  
const float marks = 98.98;  
const char grade = 'A';  
const char name[30] = "DataFlair";  
  
cout<<"The constant int value is: "<< value <<endl;  
cout<<"The constant floating-point marks is: "<< marks <<endl;  
cout<<"The constant character grade is: "<< grade <<endl;  
cout<<"The constant string name is: "<< name <<endl;  
  
return 0;  
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ const2.cpp -o const2
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./const2
```

```
Welcome to DataFlair tutorials!
```

```
The constant int value is: 4
```

```
The constant floating-point marks is: 98.98
```

```
The constant character grade is: A
```

```
The constant string name is: DataFlair
```

```
dataflair@asus-System-Product-Name:~/Desktop$ □
```

5. Summary

In this tutorial, we discussed the difference between variables and constants in C and C++. Then, we further carried on our discussion by shedding light on how to declare or define constants or literals and the types of constants available in C in detail. After completing this tutorial, you have gained command over C/C++ constants and literals.

Learn Data Types in C and C++ with Example in Just 4 mins.

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 25, 2019

A programming language cannot work without a wide range of data types as each type has its own significance and utility to perform various tasks. Here, reasons are mentioned why we require different data types in C and C++ Programming:

- At the time of the variable declaration, it becomes convenient for the user to distinguish which type of data variable stores.
- It makes it clear for the user to identify the return value of the function based on the data type.
- If parameters are passed to the function, it becomes easy for the user to give input according to the given format.

This is just the beginning, at the end of this article, you will be an expert in Data Types in C and C++.

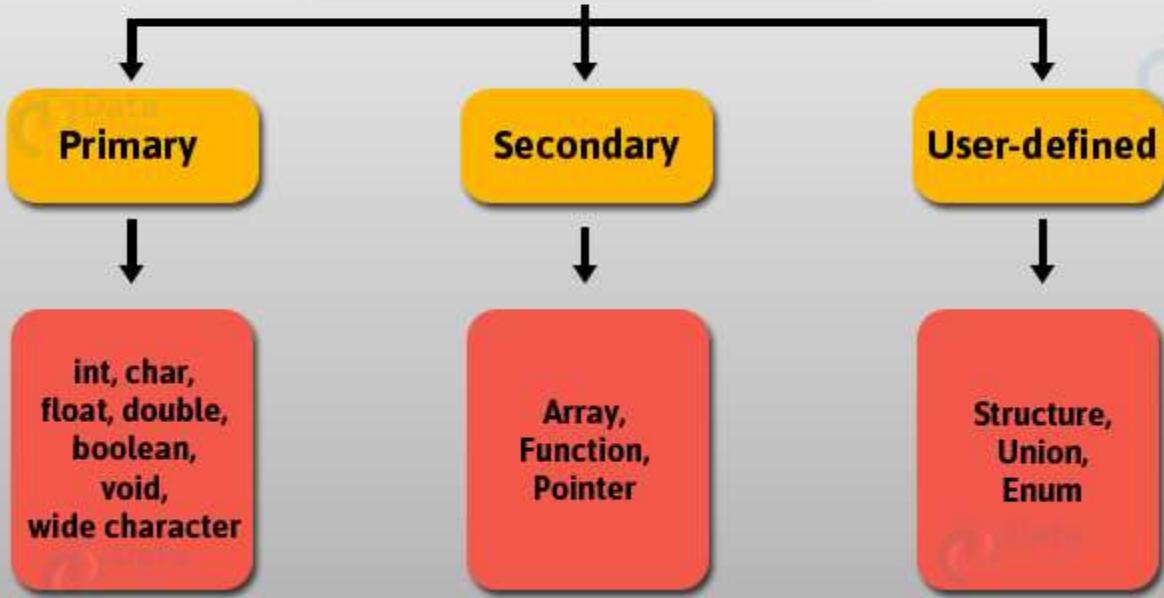
1. What is Data Type in C/C++?

Data types in C and C++ refer to the *characteristics of data stored into a variable*. For instance, while working with mathematical problems, in order to simplify things for us, we look for a specific type of data, let's say, we want to find the factorial of a number. We know that only for whole numbers, the factorial of that number exists which is also a whole number. In order to eliminate all scopes of errors and reduce run-time of the program, we would preferably assign such a data type to the input and output such that it only covers the range of whole numbers. Voila! This is how you begin to think to increase your program efficiency and well utilize the [features offered by C](#).

Clearly, from the above discussion, you might have inferred that memory occupied by different data types would be different. Therefore, a different amount of space in the computer memory would be allocated for them and hence the run time of the program would be reduced, increasing the efficiency of the program.

2. Types of Data Types in C and C++

According to the [conventional classification](#), these are data types in C language-



2.1 Primary Data Types in C and C++

Primary (Fundamental) data types in C programming includes the 4 most basic data types, that is:

- **int:** It is responsible for storing integers. The memory it occupies depends on the compiler (32 or 64 bit). In general, int data type occupies 4 bytes of memory when working with a 32-bit compiler.
- **float:** It is responsible for storing fractions or digits up to 7 decimal places. It is usually referred to as a single-precision floating-point type. It occupies 4 bytes of memory
- **char:** It can be used to store a set of all characters which may include alphabets, numbers and special characters. It occupies 1 byte of memory being the smallest addressable unit of a machine containing a fundamental character set.
- **double:** It is responsible for storing fractions or digits up to 15-16 decimal places. It is usually referred to as a double-precision floating-point type.
- **void (Null) data type:** It indicates zero or no return value. It is generally used to assign the null value while declaring a function.

Learn the Samurai Technique to implement [Arrays in C](#)

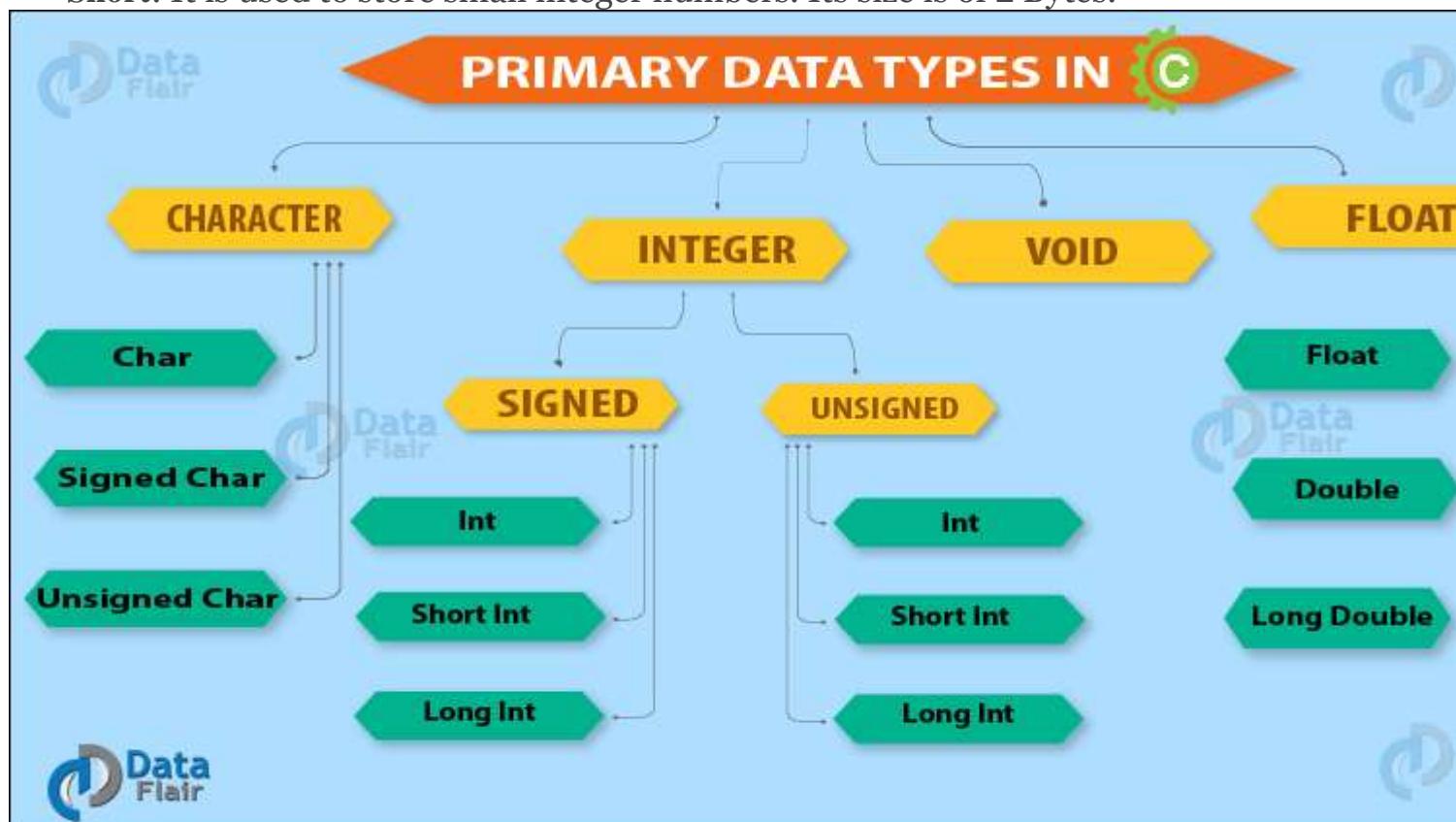
In C++, in addition to the primary data types available in C, there are few more data types available in the C++ programming language.

They are:

- **bool:** It refers to a boolean/logical value. It can either be true or false.
- **wchar_t:** It refers to a wide character whose size is either 2 or 4 bytes. It is similar to the char data type but the only difference is the space occupied in the computer memory.
- **string:** Instead of declaring an array of characters to enter a string data type, C++ gives you the provision to declare the “string” data type.

Along with [data types](#), comes modifiers. Modifiers basically alter the function of a data type and make it more specific by the inclusion of additional provisions. These include:

- **Signed:** It is used to store zero, positive or negative values.
- **Unsigned:** It can store only zero or positive values.
- **Long:** It is used to store large integer numbers. The size of the long type is 8 bytes.
- **Short:** It is used to store small integer numbers. Its size is of 2 Bytes.



Here is a table that would specify the range of various data primary data types along with their modifiers. It is in accordance with a 32-bit compiler.

Data Type	Memory (In Bytes)	Format Specifiers	Range
int	4	%d	-2,147,483,648 to 2,147,483,647
short int	2	%hd	-32,768 to 32,767
unsigned int	4	%u	0 to 4,294,967,295
unsigned short int	2	%hu	0 to 65,535
long int	4	%ld	-2,147,483,648 to 2,147,483,647
unsigned long int	4	%lu	0 to 4,294,967,295
long long int	8	%lld	-(263) to (263)-1

unsigned long long int	8	%llu	0 to 18,446,744,073,709,551,615
char	1	%c	-128 to 127
Signed char	1	%c	-128 to 127
Unsigned char	1	%c	0 to 255
float	4	%f	-
double	8	%lf	-
long double	12	%Lf	-

In order to compute the size of a variable, we can use the function `sizeof()` operator
Its output is of the form of an unsigned integer.

Take a break for a while and learn [Variables in C with examples](#)

Example of Primary Data Types in C

Let's understand with an example of Data types in C-

```

1. #include <stdio.h>
2. int main()
3. {
4.     int number1 = 400;
5.     short int number2 = 500;
6.     unsigned short int number3 = 600;
7.     long int number4 = 700;
8.     unsigned long int number5 = 800;
9.     unsigned long long int number6 = 900;
10.    char character1 ='A';
11.    signed char character2 ='B';
12.    unsigned char character3 ='C';
13.    float digit1 =20.00;
14.    double digit2 = 3.14159;
15.    long double digit3 = 1.414213;
16.
17.    printf("Welcome to DataFlair tutorials!\n\n");
18.
19. //Print statements to show the size of various data types
20.
21. printf("The size of int data type %d is: %lu bytes.\n", number1,sizeof(number1));
22. printf("The size of short int data type %d is: %lu bytes.\n", number2,sizeof(number2));
23. printf("The size of unsigned short int data type %d is: %lu bytes.\n", number3,sizeof(number3));
24. printf("The size of long int data type %ld is: %lu bytes.\n", number4,sizeof(number4));
25. printf("The size of unsigned long int data type %ld is: %lu bytes.\n", number5,sizeof(number5));
26. printf("The size of unsigned long long int data type %lld is: %lu bytes.\n", number6,sizeof(number6));
27. printf("The size of char %c is: %lu byte.\n", character1,sizeof(character1));
28. printf("The size of signed char %c is: %lu byte.\n", character2,sizeof(character2));
29. printf("The size of unsigned char %c is: %lu byte.\n", character3,sizeof(character3));
30. printf("The size of float data type %f is: %ld bytes.\n", digit1,sizeof(digit1));
31. printf("The size of double data type %lf is: %ld bytes.\n", digit2,sizeof(digit2));
32. printf("The size of long double data type %Lf is: %ld bytes.\n", digit3,sizeof(digit3));
33. return 0;
34. }
```

The above example on your screen looks like-

File Edit View Search Terminal Help

GNU nano 2.9.3

datatype.c

```
#include <stdio.h>
int main()
{
    int number1 = 400;
    short int number2 = 500;
    unsigned short int number3 = 600;
    long int number4 = 700;
    unsigned long int number5 = 800;
    unsigned long long int number6 = 900;
    char character1 ='A';
    signed char character2 ='B';
    unsigned char character3 ='C';
    float digit1 =20.00;
    double digit2 = 3.14159;
    long double digit3 = 1.414213;

    printf("Welcome to DataFlair tutorials!\n\n");

    //Print statements to show the size of various data types
    printf("The size of int data type %d is: %lu bytes.\n", number1,sizeof(number1));
    printf("The size of short int data type %d is: %lu bytes.\n", number2,sizeof(number2));
    printf("The size of unsigned short int data type %d is: %lu bytes.\n", number3,sizeof(number3));
    printf("The size of long int data type %ld is: %lu bytes.\n", number4,sizeof(number4));
    printf("The size of unsigned long int data type %ld is: %lu bytes.\n", number5,sizeof(number5));
    printf("The size of unsigned long long int data type %lld is: %lu bytes.\n", number6,sizeof(number6));
    printf("The size of char %c is: %lu byte.\n", character1,sizeof(character1));
    printf("The size of signed char %c is: %lu byte.\n", character2,sizeof(character2));
    printf("The size of unsigned char %c is: %lu byte.\n", character3,sizeof(character3));
    printf("The size of float data type %f is: %ld bytes.\n", digit1,sizeof(digit1));
    printf("The size of double data type %lf is: %ld bytes.\n", digit2,sizeof(digit2));
    printf("The size of long double data type %Lf is: %ld bytes.\n", digit3,sizeof(digit3));

    return 0;
}
```

And, the output you will get-

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc datatypes.c -o test1
dataflair@admin4-H110M-H:~/Desktop$ ./test1
Welcome to DataFlair tutorials!

The size of int data type 400 is: 4 bytes.
The size of short int data type 500 is: 2 bytes.
The size of unsigned short int data type 600 is: 2 bytes.
The size of long int data type 700 is: 8 bytes.
The size of unsigned long int data type 800 is: 8 bytes.
The size of unsigned long long int data type 900 is: 8 bytes.
The size of char A is: 1 byte.
The size of signed char B is: 1 byte.
The size of unsigned char C is: 1 byte.
The size of float data type 20.000000 is: 4 bytes.
The size of double data type 3.141590 is: 8 bytes.
The size of long double data type 1.414213 is: 16 bytes.
dataflair@admin4-H110M-H:~/Desktop$ □
```

Still, can't get the example, learn the [Syntax of C Language](#).

Example of Primary Data Types in C++

Here is an illustrated C++ code which will help you to find the memory occupied by various data types:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
```

```

9. int number1 = 400;
10. short int number2 = 500;
11. unsigned short int number3 = 600;
12. long int number4 = 700;
13. unsigned long int number5 = 800;
14. unsigned long long int number6 = 900;
15. char character1 ='A';
16. signed char character2 ='B';
17. unsigned char character3 ='C';
18. float digit1 =20.00;
19. double digit2 = 3.14159;
20. long double digit3 = 1.414213;
21. string word = "DataFlair";// 9 characters
22. bool flag = 0;
23.
24. //Print statements to show the size of various data types
25.
26. cout<<"The size of int data type "<< number1 << " is: " << sizeof(number1) << " bytes."<<endl;
27. cout<<"The size of short int data type "<< number2 << " is: " << sizeof(number2) << " bytes."<<endl;
28. cout<<"The size of unsigned short data type "<< number3 << " is: " << sizeof(number3) << " bytes."<<endl;
29. cout<<"The size of long int data type "<< number4 << " is: " << sizeof(number4) << " bytes."<<endl;
30. cout<<"The size of unsigned long int data type "<< number5 << " is: " << sizeof(number5) << " bytes."<<endl;
31. cout<<"The size of long long int data type "<< number6 << " is: " << sizeof(number6) << " bytes."<<endl;
32. cout<<"The size of char data type "<< character1 << " is: " << sizeof(character1) << " bytes."<<endl;
33. cout<<"The size of signed char data type "<< character2 << " is: " << sizeof(character2) << " bytes."<<endl;
34. cout<<"The size of unsigned char data type "<< character3 << " is: " << sizeof(character3) << " bytes."<<endl;
35. cout<<"The size of float data type "<< digit1 << " is: " << sizeof(digit1) << " bytes."<<endl;
36. cout<<"The size of double data type "<< digit2 << " is: " << sizeof(digit2) << " bytes."<<endl;
37. cout<<"The size of long double data type "<< digit3 << " is: " << sizeof(digit3) << " bytes."<<endl;
38. cout<<"The size of string data type "<< word << " is: " << sizeof(word) << "bytes. "<<endl;
39. cout<<"The size of bool data type "<< word << " is: " << sizeof(word) << "bytes. "<<endl;
40.
41. return 0;
42. }

```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

datatype.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int number1 = 400;
    short int number2 = 500;
    unsigned short int number3 = 600;
    long int number4 = 700;
    unsigned long int number5 = 800;
    unsigned long long int number6 = 900;
    char character1 ='A';
    signed char character2 ='B';
    unsigned char character3 ='C';
    float digit1 =20.00;
    double digit2 = 3.14159;
    long double digit3 = 1.414213;
    string word = "DataFlair";// 9 characters

    //Print statements to show the size of various data types

    cout<<"The size of int data type "<< number1 << " is: " << sizeof(number1) << " bytes."<<endl;
    cout<<"The size of short int data type "<< number2 << " is: " << sizeof(number2) << " bytes."<<endl;
    cout<<"The size of unsigned short data type "<< number3 << " is: " << sizeof(number3) << " by<<endl;
    cout<<"The size of long int data type "<< number4 << " is: " << sizeof(number4) << " bytes."<<endl;
    cout<<"The size of unsigned long int data type "<< number5 << " is: " << sizeof(number5) << endl;
    cout<<"The size of long long int data type "<< number6 << " is: " << sizeof(number6) << " by<<endl;
    cout<<"The size of char data type "<< character1 << " is: " << sizeof(character1) << " bytes."<<endl;
    cout<<"The size of signed char data type "<< character2 << " is: " << sizeof(character2) << endl;
    cout<<"The size of unsigned char data type "<< character3 << " is: " << sizeof(character3) << endl;
    cout<<"The size of float data type "<< digit1 << " is: " << sizeof(digit1) << " bytes."<<endl;
    cout<<"The size of double data type "<< digit2 << " is: " << sizeof(digit2) << " bytes."<<endl;
    cout<<"The size of long double data type "<< digit3 << " is: " << sizeof(digit3) << " bytes."<<endl;
    cout<<"The size of string data type "<< word << " is: " << sizeof(word) << " bytes."<<endl;

    return 0;
}
```

Output-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

dataflair@asus-System-Product-Name:~/Desktop\$ g++ datatypes.cpp -o datatypes

dataflair@asus-System-Product-Name:~/Desktop\$./datatypes

Welcome to DataFlair tutorials!

```
The size of int data type 400 is: 4 bytes.  
The size of short int data type 500 is: 2 bytes.  
The size of unsigned short data type 600 is: 2 bytes.  
The size of long int data type 700 is: 8 bytes.  
The size of unsigned long int data type 800 is: 8 bytes.  
The size of long long int data type 900 is: 8 bytes.  
The size of char data type A is: 1 bytes.  
The size of signed char data type B is: 1 bytes.  
The size of unsigned char data type C is: 1 bytes.  
The size of float data type 20 is: 4 bytes.  
The size of double data type 3.14159 is: 8 bytes.  
The size of long double data type 1.41421 is: 16 bytes.  
The size of string data type DataFlair is: 32 bytes.
```

dataflair@asus-System-Product-Name:~/Desktop\$ □

2.2 Secondary (Derived) Data Types in C and C++

As the name itself suggests, they are derived from the fundamental data types in the form of a group to collect a cluster of data used as a single unit. These include:

- **Arrays:** A collection of data items of similar data types, which is accessed using a common name.

The basic syntax of declaring an array is:

1. `return_type array_name[size];`

For instance: **float marks[5];**

- **Pointers:** [*Pointers in C/C++*](#) are basically variables that are used to store the memory address of another variable.
- **Functions:** It is a group of statements that are written to perform a specific task. Functions are either user-defined or built-in library functions.

2.3 User-defined Data Types in C and C++

- **Structures** – It is a user-defined data type in which a collection of different data types can be made and accessed through an object.
- **Union**– A special kind of data type which gives us the provision to store different data types in the same memory location.
- **Enumeration** – It refers to the arithmetic data types used to define variables to specify only certain discrete integer values throughout the entire program.

Key takeaway: Arrays and [*structures in C*](#) are collectively called aggregates.

Format Specifiers C

It is important to note that format specifiers is an exclusive feature only available in C, not in C++.

Often referred to as format string, it is an exclusive feature of the C language. It is associated with data types as it defines the type of data to be taken as input or printed when using the I/O statements.

Here is a table that would help you to explore a wide range of format specifiers used for various purposes.

Format Specifiers	Data Type	Elucidation
%d, %i	int short long unsigned short	Used for integers
%c	char signed char unsigned char	Used for characters with any type modifier
%f	float	Used for decimal values
%e, %E, %g, %G	float double	Used for scientific notation of decimal values
%hi	short	Used for signed short integers
%hu	unsigned short	Used for unsigned short integers
%l, %ld, %li	long	Used for signed integers
%lf	double	Used for floating-point
%Lf	long double	Used for floating-point

%lu	unsigned int unsigned long	Used for unsigned integers
%lli, %lld	long long	Used for signed long integers
%llu	unsigned long long	Used for unsigned long long integers
%s	char *	Used for a string
%p	void *	Used when finding the address of the pointer to void *
%o	int short long unsigned short unsigned int	Used for the octal representation of Integer.
%u	unsigned int unsigned long	Used for unsigned integers
%x, %X	int short long unsigned short unsigned int	Used for the hexadecimal representation of Unsigned Integer
%%	-	Used to print % character
%n	-	Used to print nothing

Summary

Data types provide a backbone to the programming language, as it helps the user to differentiate between the type of data variable stores at the time of the variable declaration itself. Additionally, while writing the logic of the program, it becomes clear for the user to identify the return value of the function based on the data type. Not only this, by specifying the data type of the parameters passed to the function, it becomes easy for the user to give input according to the given format.

Loops in C and C++ | The Survival Guide for every Beginner

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 25, 2019

In this tutorial, we will begin our discussion by understanding the meaning of the term **Loops** followed by “Iteration” or “Looping”. It is important to note that the terms “Iteration” and “Looping” can be used interchangeably. Then, we will lay emphasis on how and why is looping done in C and C++. Further, we will discuss the significance of loops in C and C++. Then, we will move towards discussing its types in detail followed by control statements.

We will end this tutorial by analyzing the choice of looping.

It is important to have a great command over this topic to ease your way out in C and C++ programming as it plays an integral role while working with repetitive statements.

We all know how cumbersome it is to give a separate print statement each time we wish to display something, right? We can easily resolve this issue by covering every topic in detail to master the concept of looping.

1. What are Loops in C and C++?

Loops are nothing but a segment of code we use to repeat a specific block of code. It is important to note that ‘loop’ is a concept and ‘looping’ refers to a process to implement loops in C and C++.

Iteration or looping refers to the specified sequence in which a pattern or a block of code is repeated until the condition becomes no longer true. As long as the condition remains to be true, the loop is executed. It is terminated when the condition is false or no longer satisfied.

Take a break and explore the [Applications of C](#)

Every loop has two parts:

1.1 Body of the loop

It consists of the content, like the print statement or a fragment of code which we want to run multiple times.

1.2 Control Statement

These statements control the execution of the loop and are important for loop termination.

Since we now got an idea as to what are loops in C/C++, let's acknowledge its importance by thinking about its significance.

- It is of the utmost importance when the programmer wants to create a database. Creating a database requires taking multiple inputs several times, storing it, displaying it and performing modifications.
- Most of the programs which require mathematical calculations are simplified using loops.
- It saves time and effort.

This is a diagrammatic representation of how looping works in C and C++.

2. Types of Loops in C and C++

The C and C++ language offer you the facility of looping in various formats. There are basically 2 types of loops:

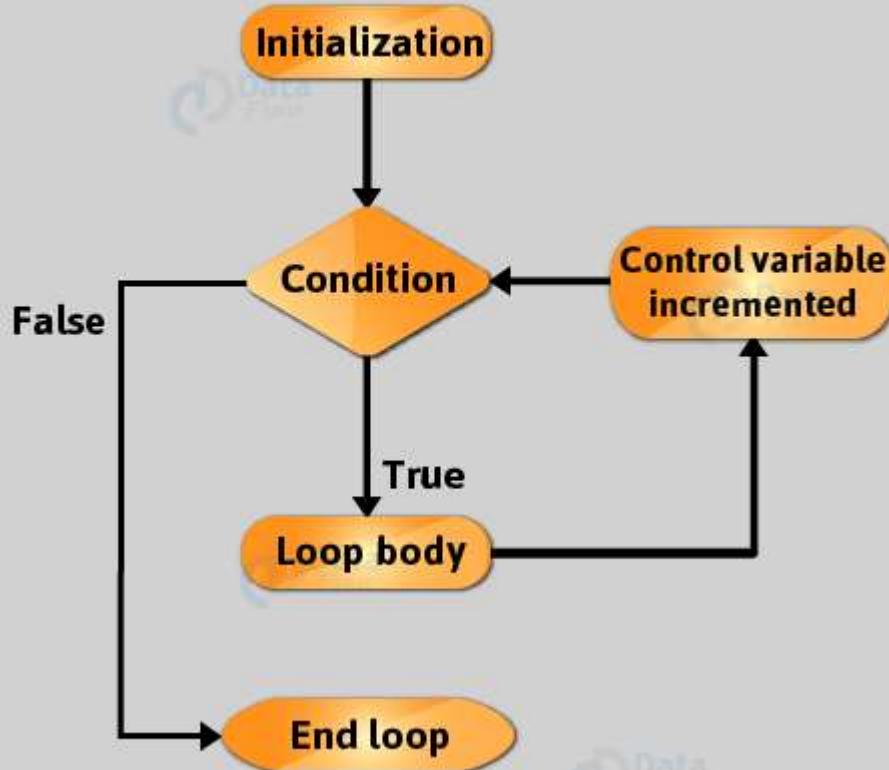
Do you know the major features of C++?

2.1 Entry-controlled loops

Before we execute the loop, we need to check the given conditions beforehand. For this loop to work, the test condition should necessarily be true. For example, for and while loops are entry-controlled.

Here is a diagrammatic representation of an entry-controlled loop which will help you better understand its working:

Entry-Controlled Loops



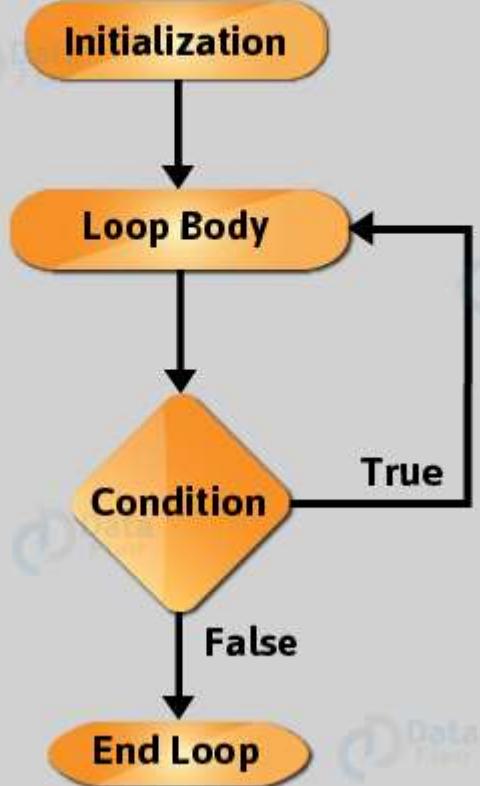
2.2 Exit-controlled loops

In contrast to entry controlled loops, this type of looping offers the benefit to the user to execute the loop at least once before checking the condition. In this case, to execute the loop, the test condition might not necessarily be true.

For example, the do-while loop is exit-controlled. The C and C++ language offer this benefit, unlike Python.

Here is a diagrammatic representation of an exit-controlled loop which will help you better understand its working:

Exit-Controlled Loop



Before we discuss the various types of loops in detail, let us understand that the control statements should correctly be specified, otherwise the loop might never terminate, which is formally referred to as, the infinite loop. It is a bad programming practice and hence should be avoided in every possible way.

The following steps prevent your program from going into an infinite (endless) loop:

- When the termination conditions are not clearly mentioned.
- The control statements are missing.
- The conditions are never met.

Loops in C/C++

1

While Loop

Do-while Loop

2

3

For Loop

Nested For Loop

4

3. While Loop in C and C++

It is the easiest and most basic type of looping which exists in the C and C++ language.
It is an entry controlled loop.

As soon as the loop is executed for the first time, the flow of control goes back to the beginning of the loop again, checking if the subsequent conditions hold true or not.

After loop termination, the flow of control goes back to the immediate statement after the loop.

Before we move forward, let's revise basic [Syntax of C](#)

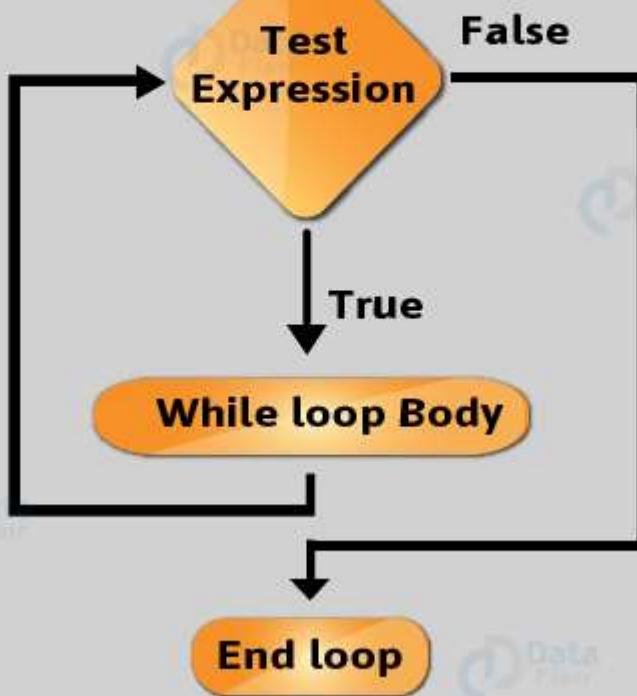
Basic Syntax

```
while(condition)
{
    statement(s)
}
```

Key takeaway: If the loop isn't multiline, which means, it does not contain more than one line, it needn't be enclosed within curly braces. *It is applicable for while, for and do-while loops in C.*

Here is a diagrammatic representation of the while loop:

while loop



Let us consider a very simple program to print a certain statement 10 times using the while loop:

Example of While loop in C

Here is a code in C that illustrates the basic syntax and working of the while loop:

```
1. #include <stdio.h>
2. int main()
3. {
4.
5.     int no_of_executions = 1; //Initializing the number of executions to 1
6.
7.     while ( no_of_executions <= 10 )
8.     {
9.         printf("Welcome to DataFlair tutorials!\n"); //Statement to be displayed 10 times
10.        no_of_executions++; //Incrementing the number of executions
11.    }
12.    return 0;
13. }
```

Code on Screen

File Edit View Search Terminal Help

GNU nano 2.9.3

while.c

```
#include <stdio.h>
int main()
{
    int no_of_executions = 1;                                //Initializing the number of executions

    while ( no_of_executions <= 10 )
    {
        printf("Welcome to DataFlair tutorials!\n");      //Statement to be displayed 10 times
        no_of_executions++;                                //Incrementing the number of executions
    }
    return 0;
}
```

Output

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~$ cd Desktop
dataflair@admin4-H110M-H:~/Desktop$ gcc while.c -o while
dataflair@admin4-H110M-H:~/Desktop$ ./while
Welcome to DataFlair tutorials!
dataflair@admin4-H110M-H:~/Desktop$ 
```

To run and execute the above program, [Install C on Linux with GCC Compiler](#)

Example of While loop in C++

Here is a code in C++ that illustrates the basic syntax and working of the while loop:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7.     int no_of_executions = 1; // Initializing the number of executions to 1
8.
9.     while ( no_of_executions <= 10 )
10.    {
11.        cout<<"Welcome to DataFlair tutorials!"<<endl; // Statement to be displayed 10 times
```

```
12. no_of_executions++; // Incrementing the number of executions
13. }
14.
15. return 0;
16. }
```

Code-

The screenshot shows a terminal window titled "dataflair@asus-System-Product-Name:~". The window has a dark background and light-colored text. At the top, there's a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". Below the menu, it says "GNU nano 2.9.3" on the left and "while2.cpp" on the right. The main area contains the following C++ code:

```
#include <iostream>
using namespace std;

int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int no_of_executions = 1; // Initializing the number of executions to 1
    while(no_of_executions <= 10) // Condition
    {
        cout << no_of_executions << endl;
        no_of_executions++; // Incrementing the number of executions
    }
    return 0;
}
```

Output-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ while.cpp -o while  
dataflair@asus-System-Product-Name:~/Desktop$ ./while
```

```
Welcome to DataFlair tutorials!  
Welcome to DataFlair tutorials!
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Elucidation

Let us understand every statement step by step:

- **Initialization:** Since we want to display the text within the print statement 10 times, therefore, we start with 1. If we initialize **no_of_executions** to 0, the loop will display the message 11 times as indexing starts from 0.
- **Condition:** Here, while is a reserved keyword which indicates the use of while loop. The condition for the loop to be terminated is mentioned within simple brackets. The loop will be executed 11 times. The message will be displayed 10 times, the other one time the loop would check if the condition is true, which is not the case here, so the loop will terminate.

- **Incrementation:** For the flow of control to be passed onto the next condition, that is, for **no_of_executions** to change its value to 2, we use increment operator.

In order to realize the contribution of looping in mathematical aspects, let us now discuss the syntax and working of while loop with the help of a simple example by printing numbers from 1 to 10.

Here is a code in C that illustrates the working of the while loop when playing with numbers:

```
1. #include<stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     int no_of_executions = 1; // Initializing the number of executions to 1
8.     while(no_of_executions <= 10) // Condition
9.     {
10.         printf("%d\n",no_of_executions);
11.         no_of_executions++; // Incrementing the number of executions
12.     }
13.     return 0;
14. }
```

Code on Screen

File Edit View Search Terminal Help

GNU nano 2.9.3

dowhile.c

```
#include<stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int number = 1;                                // Initializing the number to 1
    do                                                 // Starting of the do-while loop
    {
        printf("%d\n",number*number);
        number++;                                    // Incrementing the number from 1 to 10
    }
    while(number <= 10);
    return 0;
}
```

Output

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc while2.c -o while2
dataflair@admin4-H110M-H:~/Desktop$ ./while2
Welcome to DataFlair tutorials!

1
2
3
4
5
6
7
8
9
10
dataflair@admin4-H110M-H:~/Desktop$ █
```

Elucidation

As mentioned earlier, we initialize, specify the condition and increment the variable in order to use the while loop. Now, let us understand how to display numbers from 1 to 10 using the while loop:

- We initialized the **no_of_executions** to 1 as we want to start printing the numbers from 1. If we wanted to begin with 5 and end with 15, we would initialize **no_of_executions** to 5 and changed the condition to **no_of_executions <=15**.
- The loop will be executed till the number is incremented and a point is achieved where the **no_of_executions** becomes 11 and the loop terminates.

Haven't you studied [Variables in C](#)?

4. Do-While Loop in C and C++

The do-while loop is pretty similar to while loop with an additional feature. It is an exit-controlled loop that gives the user the provision to execute the loop at least one.

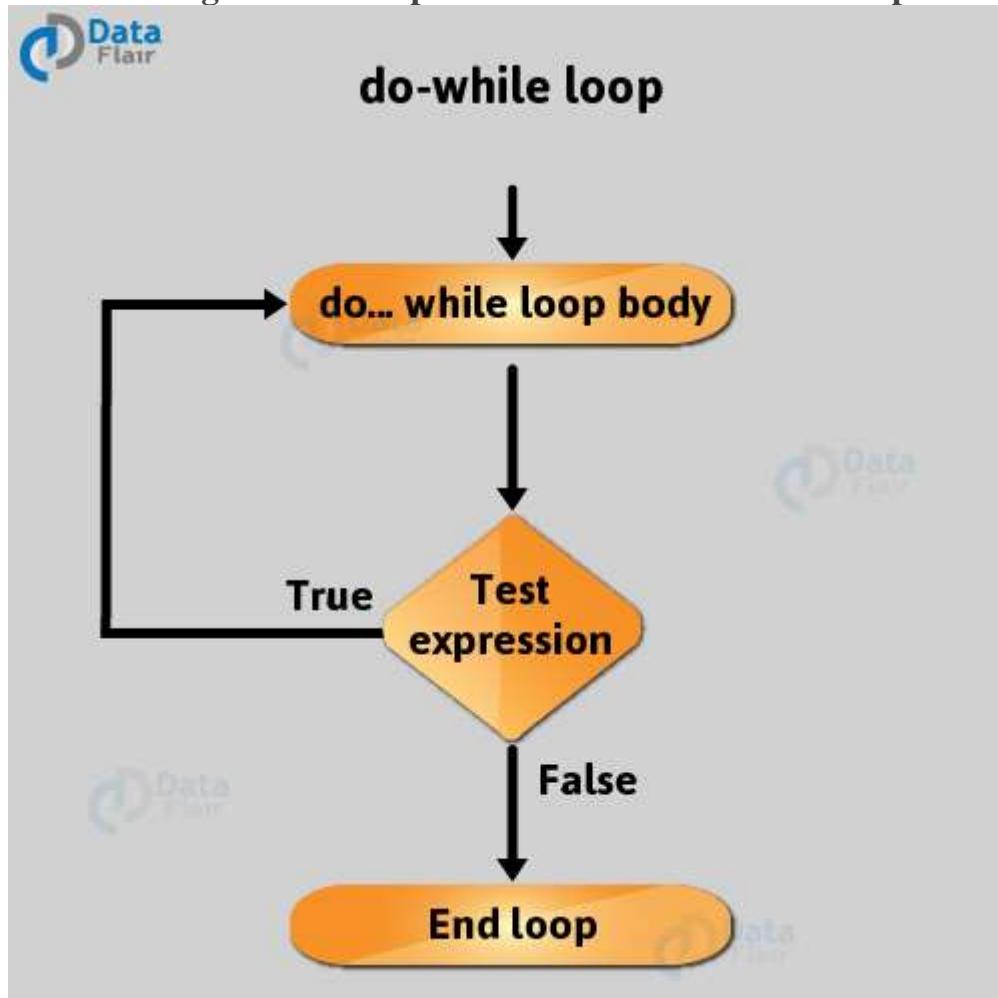
It safe to say that the do-while loop is better than while loop in all aspects. Here, the flow of control goes something like this: First, the body of the loop is executed, then the condition is checked which is in contrast to the while loop.

Basic Syntax-

```
do
{
    statement(s)
} while(condition)
```

In the while loop, the keyword while is written at the beginning of the loop, but here, it is written at the end of the loop.

Here is a diagrammatic representation of the do while loop:



Let us consider a simple program to print the squares of first 10 natural numbers using the do while loop:

Here is a code in C that illustrates the basic syntax and working of the do while loop:

```
1. #include<stdio.h>
```

```
2. int main()
3. {
4.
5. printf("Welcome to DataFlair tutorials!\n\n");
6.
7. int number = 1; // Initializing the number to 1
8. do // Starting of the do-while loop
9. {
10. printf("%d\n",number*number);
11. number++; // Incrementing the number from 1 to 10
12. }
13. while(number <= 10);
14. return 0;
15. }
```

Code on Screen

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: dowhile.c
- Code content:

```
#include<stdio.h>
int main()
{
printf("Welcome to DataFlair tutorials!\n\n");
int number = 1; // Initializing the number to 1
do // Starting of the do-while loop
{
printf("%d\n",number*number);
number++; // Incrementing the number from 1 to 10
}
while(number <= 10);
return 0;
}
```

Output

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc dowhile.c -o dowhile
dataflair@admin4-H110M-H:~/Desktop$ ./dowhile
Welcome to DataFlair tutorials!

1
4
9
16
25
36
49
64
81
100
dataflair@admin4-H110M-H:~/Desktop$ 
```

Here is a code in C++ that illustrates the basic syntax and working of the do while loop:

```
1. #include<iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int number = 1; // Initializing the number to 1
10. do // Starting of the do-while loop
11. {
12. cout<< number*number <<endl;
13. number++; // Incrementing the number from 1 to 10
14. }
15. while(number <= 10);
16. return 0;
17. }
```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

dowhile.cpp

```
#include<iostream>
using namespace std;

int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");

    int number = 1; // Initializing the number to 1
    do // Starting of the do-while loop
    {
        cout<< number*number << endl;
        number++; // Incrementing the number from 1 to 10
    }
    while(number <= 10);
    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ dowhile.cpp -o dowhile
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./dowhile
```

```
Welcome to DataFlair tutorials!
```

```
1  
4  
9  
16  
25  
36  
49  
64  
81  
100
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Elucidation

1. We initialized the number to 1 so that it prints the squares of numbers starting from 1.
2. In the do-while loop, we use the keyword do at the starting of the loop.
3. In the print statement, we performed the multiplication of the variable with itself to find its square for simplicity. To find cubes or other higher powers to which the number is to be raised by, we use math functions that have a wide range of inbuilt functions to perform mathematical operations.
4. We increment the number from 1, now it has a value 2 which will follow the same process and further keep on incrementing till the specific condition is met.

Build your basic knowledge with [Functions in C](#)

5. For Loop in C and C++

It is safe to say that it is the best loop for iteration by programmers since it is very easy to implement and has a well-built structure. It is an entry controlled loop.

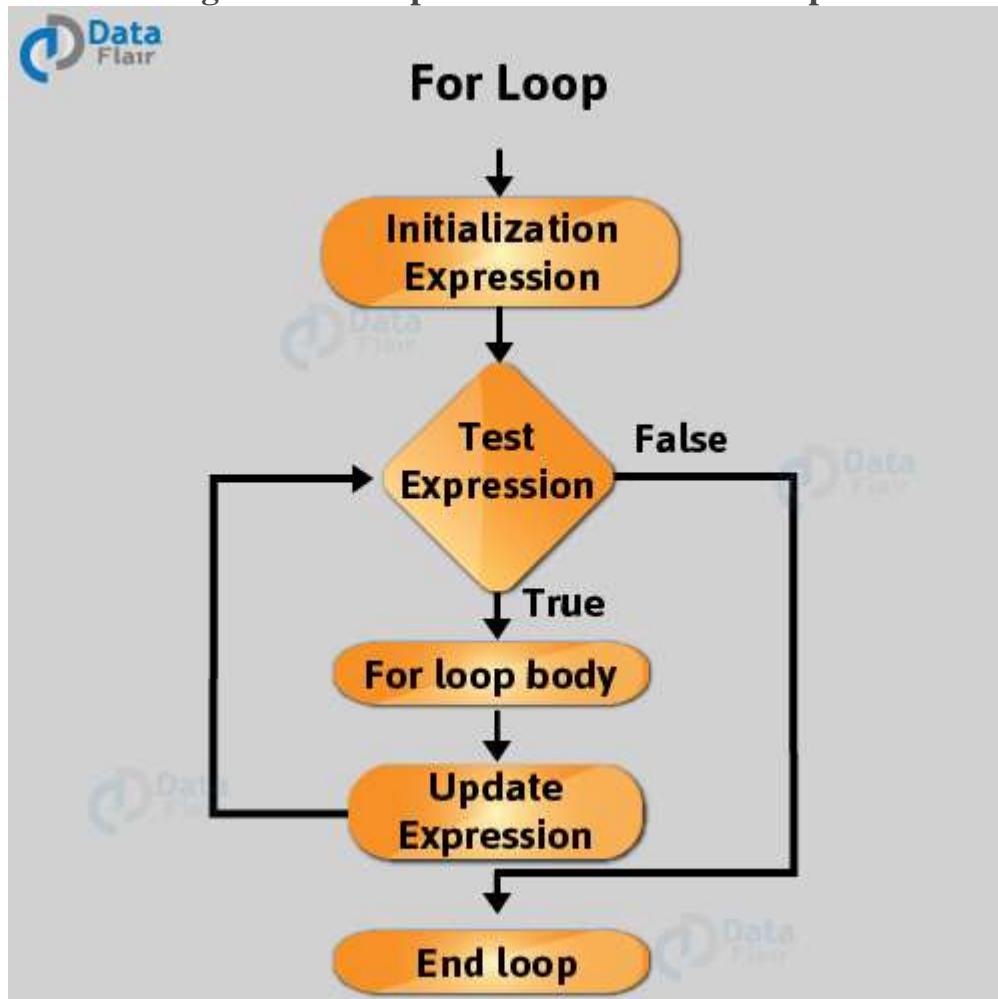
Basic Syntax:

```
for( initialization ; condition ; increment / decrement )  
{  
statement(s)  
}
```

The for loop takes into account three parameters:

- Initialization
- Condition
- Increment or decrement

Here is a diagrammatic representation of the for loop:



Let us consider a simple example to display all the even numbers from 1 to 10 using for loop. In order to keep things simple, we use the concept that all numbers from 1 to 10 are multiples of 2

Example of for loop in C

Here is a code in C that illustrates the working of the for loop to display even numbers from 1 to 10:

```
1. #include<stdio.h>
2. int main()
3. {
4.
5. printf("Welcome to DataFlair tutorials!\n\n");
6.
7. int number;
8. for(number=1;number<=5;number++) //Use of for loop to print even numbers from 1 to 10
9. {
10. printf("%d\n",2*number); // Printing all the even numbers from 1 to 10
11.
12. }
13. return 0;
14. }
```

Code on Screen

File Edit View Search Terminal Help

GNU nano 2.9.3

for.c

```
#include<stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int number;
    for(number=1;number<=5;number++)      //Use of for loop to print even numbers from 1
    {
        printf("%d\n",2*number);        // Printing all the even numbers from 1 to 10
    }
    return 0;
}
```

Output

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc for.c -o for
dataflair@admin4-H110M-H:~/Desktop$ ./for
Welcome to DataFlair tutorials!

2
4
6
8
10
dataflair@admin4-H110M-H:~/Desktop$ 
```

Example of for loop in C++

Here is a code in C++ that illustrates the working of the for loop to display even numbers from 1 to 10:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int number;
10. for(number = 1; number <= 5; number++) //Use of for loop to print even numbers from 1 to 10
11. {
12. cout<< 2*number <<endl; // Printing all the even numbers from 1 to 10
13. }
14.
```

```
15. return 0;  
16. }
```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

for.cpp

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
  
cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;  
  
int number;  
for(number = 1; number <= 5; number++) //Use of for loop to print even numbers from 1 to 10  
{  
cout<< 2*number <<endl; // Printing all the even numbers from 1 to 10  
}  
  
return 0;  
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ g++ for.cpp -o for  
dataflair@asus-System-Product-Name:~/Desktop$ ./for  
Welcome to DataFlair tutorials!  
2  
4  
6  
8  
10  
dataflair@asus-System-Product-Name:~/Desktop$
```

To run and execute the above program, [install C++ on Linux with G++ Compiler](#)

Elucidation

1. We declared the variable number in the main function.
2. We initialized the value of the number to 1 and number ≤ 5 since there are 5 even numbers between 1 to 10.
3. We multiplied the variable number by 2 to obtain all the even numbers lying between 1 to 10.

In this way, we can implement the same program using if-else condition and modulus operator which we will discuss in detail in the section.

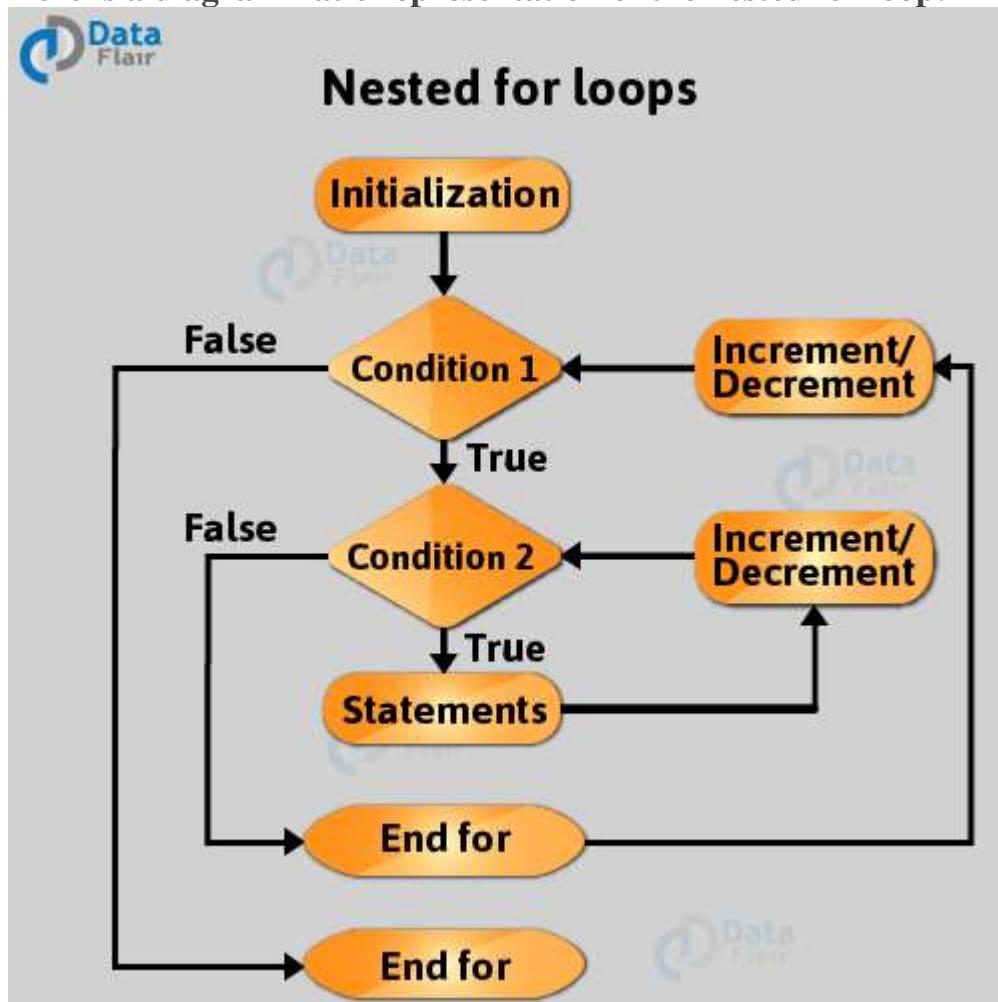
6. Nested For Loop in C and C++

Let's further elaborate our discussion on nested for loop. Before we begin, let's make it clear that nested while and do-while loops exist but aren't popularly used. The term nested refers to a loop within a loop. Each time the outer loop is iterated, the inner loop repeats itself.

Basic Syntax:

```
for( initialization ; condition ; increment / decrement )  
{  
for( initialization ; condition ; increment / decrement )  
{  
statement(s)  
}  
}
```

Here is a diagrammatic representation of the nested for loop:



Let us consider an example of the nested for a loop by displaying the following pattern:

*
* *
* * *

Example of Nested Loop in C

Here is a code in C to display the above pattern:

```
1. #include <stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     int no_of_rows=3;
8.     int iteration1, iteration2;
9.     for( iteration1 = 1; iteration1 <= no_of_rows; iteration1++ )
10.    {
11.        for( iteration2 = 1; iteration2<=iteration1; iteration2++ )
12.        {
13.            printf("* ");
14.        }
15.        printf("\n"); //To display a gap between the lines
16.    }
17.    return 0;
18. }
```

Code on Screen

File Edit View Search Terminal Help

GNU nano 2.9.3

nest.c

```
#include <stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");

    int no_of_rows=3;
    int iteration1, iteration2;
    for( iteration1 = 1; iteration1 <= no_of_rows; iteration1++ )
    {
        for( iteration2 = 1; iteration2<=iteration1; iteration2++ )
        {
            printf("* ");
        }
        printf("\n"); //To display a gap between the lines
    }
    return 0;
}
```

Output

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ cd Desktop
dataflair@admin4-H110M-H:~/Desktop$ gcc nest.c -o nest
dataflair@admin4-H110M-H:~/Desktop$ ./nest
Welcome to DataFlair tutorials!

*
*
*
dataflair@admin4-H110M-H:~/Desktop$ █
```

Example of Nested Loop in C++

Here is a code in C++ to display the above pattern:

```
1. #include<iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int no_of_rows = 3;
10. int iteration1, iteration2;
11. for( iteration1 = 1; iteration1 <= no_of_rows; iteration1++ )
12. {
13. for( iteration2 = 1; iteration2<=iteration1; iteration2++ )
14. {
```

```
15. cout<<"* ";
16. }
17. cout<<endl; //To display a gap between the lines
18. }
19. return 0;
20. }
```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

nest.cpp

```
#include<iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int no_of_rows = 3;
    int iteration1, iteration2;
    for( iteration1 = 1; iteration1 <= no_of_rows; iteration1++ )
    {
        for( iteration2 = 1; iteration2<=iteration1; iteration2++ )
        {
            cout<<"* ";
        }
        cout<<endl; //To display a gap between the lines
    }
    return 0;
}
```

Output -

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ nest.cpp -o nest
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./nest
```

```
Welcome to DataFlair tutorials!
```

```
*
* *
* * *
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Elucidation

1. In the given sample problem, there are 3 rows and hence we initialize the number of rows to 3.
2. We need 2 for loops, one to traverse row-wise and the other to traverse column-wise and hence we initialize iteration1 and iteration2 as 1. We keep iteration2 less than iteration1 because the loop would be executed 3 times.
3. For each iteration of the outer loop, the inner loop gets executed every time. We used '*' in the print statement to display it.

7. Control Statements in C and C++

Control statements are used to specify the flow of control in the program which basically refers to the sequence in which the program gets executed.

We use control statements in C because of the following reasons:

- It is easy to implement and perform repetitive operations.
- We can access any fragment of code in the entire program whenever required.
- It maintains a sequential control of various segments of the code.

Here are a couple of keywords associated with control statements:

7.1 Break Statement in C and C++

The keyword break is used in the selection statement of the switch case. It enables us to terminate the loop immediately as per our requirement.

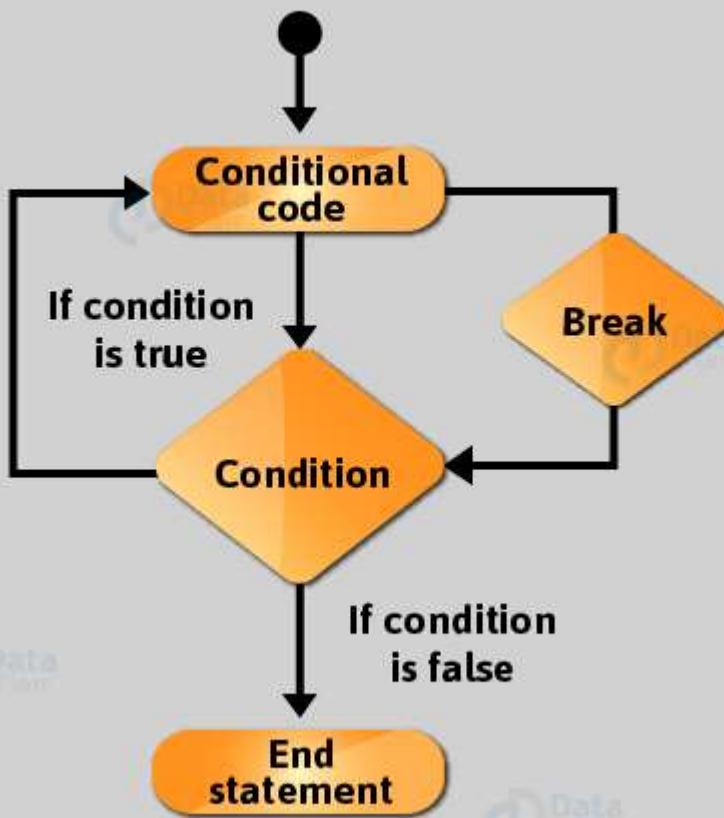
In order to understand the use of break, we must understand the switch case.

Sometimes, we have multiple operations to perform. Merging all the operations into a single program makes our program haphazard.

In order to simplify things, we divide our code into various segments and it depends on the choice of the user which segment he wishes to use. We can achieve this task by using the switch statement.

Here is a diagrammatic representation of break statement:

break statement



A typical example of the switch statement is the construction of a calculator that performs basic mathematical operations like addition, subtraction, multiplication, and division. The user needs to switch cases between these 4 operations.

The role of break comes into play when we need to switch cases.

Example of Break Statement in C and C++

Here is a code in C to create a basic calculator which illustrates the use of break keyword:

```

1. #include<stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     int number1 = 10, number2 = 20;
8.     char operation = '+';
9.     switch(operation)
10.    {
11.        case '+':
12.            printf("Addition of %d and %d is: %d\n", number1 , number2 , number1 + number2 );
13.            break;
14.        case '-':
15.            printf( "Subtraction of %d and %d is: %d\n", number1 , number2 , number1 - number2 );
16.            break;
  
```

```
17. case '*':
18. printf( "Multiplication of %d and %d is: %d\n", number1 , number2 , number1 * number2 );
19. break;
20. case '/':
21. printf( "Division of %d and %d is: %d\n", number1 , number2 , number1 / number2 );
22. break;
23. default:
24. printf( "Invalid choice\n");
25. break;
26. }
27. return 0;
28. }
```

Code on Screen

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name:
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: break.c
- Code content:

```
#include<stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int number1 = 10, number2 = 20;
    char operation = '+';
    switch(operation)
    {
        case '+':
            printf("Addition of %d and %d is: %d\n", number1 , number2 , number1 + number2);
            break;
        case '-':
            printf( "Subtraction of %d and %d is: %d\n", number1 , number2 , number1 - number2);
            break;
        case '*':
            printf( "Multiplication of %d and %d is: %d\n", number1 , number2 , number1 * number2);
            break;
        case '/':
            printf( "Division of %d and %d is: %d\n", number1 , number2 , number1 / number2);
            break;
        default:
            printf( "Invalid choice\n");
            break;
    }
    return 0;
}
```

Output

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc break.c -o break
dataflair@admin4-H110M-H:~/Desktop$ ./break
Welcome to DataFlair tutorials!

Addition of 10 and 20 is: 30
dataflair@admin4-H110M-H:~/Desktop$ 
```

Example of Break Statement in C++

Here is a code in C++ to create a basic calculator which illustrates the use of break keyword:

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     cout<<"Welcome to DataFlair tutorials!"<<endl;
6.
7.     int number1 = 10, number2 = 20;
8.     char operation = '+';
9.     switch(operation)
10.    {
11.        case '+':
12.            cout<< "Addition of "<< number1 << " and " << number2 << " is " << number1 + number2 << endl;
13.            break;
```

```

14. case '+':
15. cout<< "Subtraction of "<< number1 << " and " << number2 << " is " << number1 - number2 << endl;
16. break;
17. case '*':
18. cout<< "Multiplication of "<< number1 << " and " << number2 << " is " << number1 * number2 << endl;
19. break;
20. case '/':
21. cout<< "Division of "<< number1 << " and " << number2 << " is " << number1 / number2 << endl;
22. break;
23. default:
24. cout<< " Invalid choice "<< endl;
25. break;
26. }
27. return 0;
28. }
```

Code-

dataflair@asus-System-Product-Name:

The screenshot shows a terminal window with the following details:

- File Menu:** File, Edit, View, Search, Terminal, Help.
- Toolbar:** GNU nano 2.9.3
- Right Panel:** break.cpp
- Code Area:**

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl;

    int number1 = 10, number2 = 20;
    char operation = '+';
    switch(operation)
    {
        case '+':
            cout<< "Addition of "<< number1 << " and " << number2 << " is " << number1 + number2 << endl;
            break;
        case '-':
            cout<< "Subtraction of "<< number1 << " and " << number2 << " is " << number1 - number2 << endl;
            break;
        case '*':
            cout<< "Multiplication of "<< number1 << " and " << number2 << " is " << number1 * number2 << endl;
            break;
        case '/':
            cout<< "Division of "<< number1 << " and " << number2 << " is " << number1 / number2 << endl;
            break;
        default:
            cout<< " Invalid choice "<< endl;
            break;
    }
    return 0;
}
```
- Output Area:** Shows the command prompt and the output of the program.

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ break.cpp -o break
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./break
```

```
Welcome to DataFlair tutorials!
```

```
Addition of 10 and 20 is 30
```

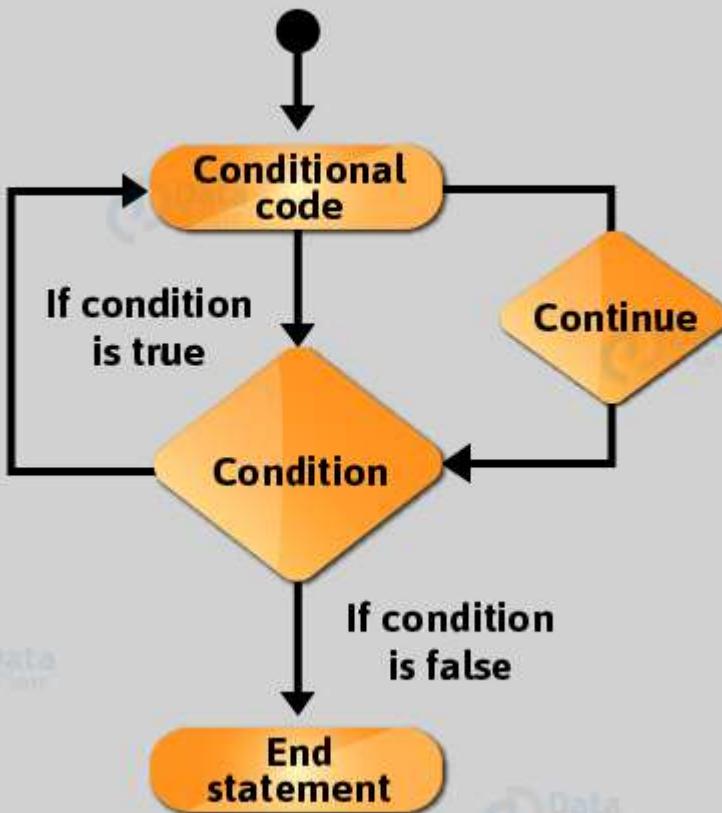
```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

7.2 Continue Statement in C and C++

If the user wishes to skip to the next iteration without exiting the loop, the continue keyword comes in play.

Here is a diagrammatic representation of the continue statement:

continue statement



Let us consider a problem in which we want to display numbers from 1 to 10 but we don't want to include 9, because 7 ate nine!

Example of Continue Statement in C

Here is a code in C which helps us to solve this problem:

```

1. #include <stdio.h>
2. int main()
3. {
4.
5.     printf("Welcome to DataFlair tutorials!\n\n");
6.
7.     int number = 10;
8.     printf( "The numbers are:\n");
9.     for( number = 1 ; number <= 10 ; number ++ )
10.    {
11.        if (number == 9)
12.        {
13.            continue;
14.        }
15.        printf("%d\n", number);
16.    }
17.    return 0;
18. }
```

Code on Screen

File Edit View Search Terminal Help

GNU nano 2.9.3

continue.c

```
#include <stdio.h>
int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    int number = 10;
    printf( "The numbers are:\n");
    for( number = 1 ; number <= 10 ; number ++ )
    {
        if (number == 9)
        {
            continue;
        }
        printf("%d\n", number);
    }
    return 0;
}
```

Output

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc continue.c -o continue
dataflair@admin4-H110M-H:~/Desktop$ ./continue
Welcome to DataFlair tutorials!

The numbers are:
1
2
3
4
5
6
7
8
9
10
dataflair@admin4-H110M-H:~/Desktop$ 
```

Example of Continue Statement in C++

Here is a code in C++ which helps us to solve this problem:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7.     cout<<"Welcome to DataFlair tutorials!"<<endl;
8.
9.     int number = 10;
10.    cout<<"The numbers are:"<<endl;
11.    for( number = 1 ; number <= 10 ; number ++ )
12.    {
13.        if (number == 9)
14.        {
15.            continue;
16.        }
17.        cout<<number<<endl;
```

```
18. }
19. return 0;
20. }
```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

continue.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl;

    int number = 10;
    cout<<"The numbers are:"<<endl;
    for( number = 1 ; number <= 10 ; number ++ )
    {
        if (number == 9)
        {
            continue;
        }
        cout<<number<<endl;
    }
    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

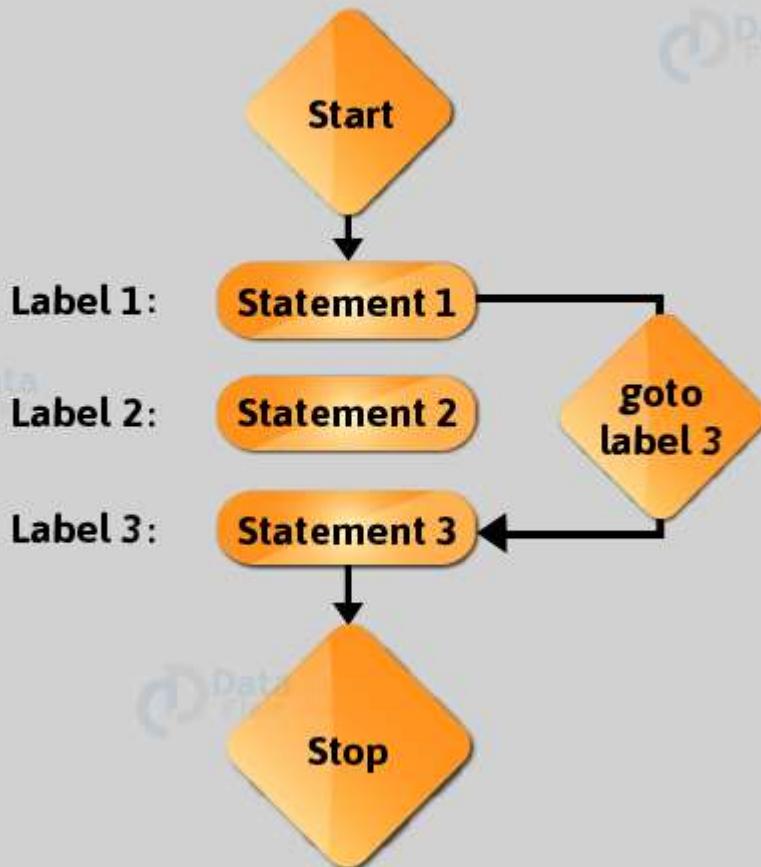
```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ g++ continue.cpp -o continue  
dataflair@asus-System-Product-Name:~/Desktop$ ./continue  
Welcome to DataFlair tutorials!  
The numbers are:  
1  
2  
3  
4  
5  
6  
7  
8  
10  
dataflair@asus-System-Product-Name:~/Desktop$
```

7.3 Goto Statement in C and C++

It is basically a jumping statement, transferring the flow of control from one iteration to the other.

Here is a diagrammatic representation of the goto statement:

goto statement



There are 2 ways in which goto statement can be used:

7.3.1 Down to top

Let us consider a problem where we wish to play the multiples of 3 in the range (1 to 30)

Here is a code in C that illustrates this approach:

```

1. #include <stdio.h>
2. int main()
3. {
4.
5.     printf( "Welcome to DataFlair tutorials!\n\n" );
6.
7.     int number=3;
8.
9.     repeat:
10.    printf("%d\n",number);
11.    number=number+3;
12.
13.    if(number<=30)
14.        goto repeat;
15.    return 0;
16. }
```

Code on Screen

File Edit View Search Terminal Help

GNU nano 2.9.3

goto.c

```
#include <stdio.h>
int main()
{
    printf( "Welcome to DataFlair tutorials!\\n\\n" );
    int number=3;
repeat:
    printf("%d\\n",number);
    number=number+3;
if(number<=30)
    goto repeat;
return 0;
}
```

Output

dataflair@admin4-H110M-H: ~/De

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc goto.c -o goto
dataflair@admin4-H110M-H:~/Desktop$ ./goto
Welcome to DataFlair tutorials!

3
6
9
12
15
18
21
24
27
30
dataflair@admin4-H110M-H:~/Desktop$ □
```

Here is a code in C++ that illustrates this approach:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9. int number = 3;
10.
11. repeat:
12. cout<< number <<endl;;
13. number=number+3;
14.
15. if(number<=30)
16. goto repeat;
17. return 0;
18. }
```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

goto.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
    int number = 3;
repeat:
    cout<< number <<endl;;
    number=number+3;
    if(number<=30)
        goto repeat;
return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ goto.cpp -o goto
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./goto
```

```
Welcome to DataFlair tutorials!
```

```
3  
6  
9  
12  
15  
18  
21  
24  
27  
30
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

7.3.2 Top to down

Let us consider a problem where we wish to print any number which is greater than 10.

Here is a code in C that helps us to solve the above problem:

```
1. #include <stdio.h>  
2. int main()  
3. {  
4.  
5.     printf( "Welcome to DataFlair tutorials!\n\n" );  
6.  
7.     int number = 30;  
8.     if(number<=20)  
9.         goto end;  
10.    printf("The number is : %d\n", number);
```

```
11. end:  
12. printf("Thank you!");  
13. return 0;  
14. }
```

Code on Screen

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

goto2.c

```
#include <stdio.h>  
int main()  
{  
  
printf( "Welcome to DataFlair tutorials!\\n\\n" );  
  
int number = 30;  
if(number<=20)  
    goto end;  
    printf("The number is : %d\\n", number);  
end:  
    printf("Thank you!");  
return 0;  
}
```

Output

dataflair@admin4-H110M-H:~/De

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~/Desktop$ gcc goto2.c -o goto2
dataflair@admin4-H110M-H:~/Desktop$ ./goto2
Welcome to DataFlair tutorials!

The number is : 30
Thank you!dataflair@admin4-H110M-H:~/Desktop$ 
```

Here is a code in C++ that helps us to solve the above problem:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.
7.     cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
8.
9.     int number = 30;
10.    if(number<= 20)
11.        goto end;
12.    cout<<"The number is :"<< number<<endl;
13.    end:
14.    cout<<"Thank you!"<<endl;
15.    return 0;
16. }
```

Code-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

GNU nano 2.9.3

goto2.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    int number = 30;
    if(number<= 20)
        goto end;
    cout<<"The number is :"<< number<<endl;
end:
    cout<<"Thank you!"<<endl;
return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name:
```

```
File Edit View Search Terminal Help
```

```
dataflair@asus-System-Product-Name:~/Desktop$ g++ goto2.cpp -o goto
```

```
dataflair@asus-System-Product-Name:~/Desktop$ ./goto
```

```
Welcome to DataFlair tutorials!
```

```
The number is :30
```

```
Thank you!
```

```
dataflair@asus-System-Product-Name:~/Desktop$ 
```

8. Which Loop to Select?

In order to make the choice of looping, the foremost thing to keep in mind is to decide whether the given problem at hand requires a pre-condition, for which we may use the while loop or the for loop, and, for a post-condition test, we may use the do-while loop.

Keeping this in mind, half of our problem would be solved.

If you are a good programmer, you would prefer to use for loop as it is better than any other loop.

9. Summary

From this tutorial, we inferred that iteration plays a major role when it comes to programming. Programming without the concepts of iteration is like reading a language without the knowledge of grammar.

We discussed by shedding light on what looping actually is, its need, classification of the types of loops in C and C++, various control statements and the choice of looping.

Function in C and C++ – You Wish You Knew Beforehand

BY [DATAFLAIR TEAM](#) · UPDATED · JUNE 27, 2019

Function in C and C++ is a set of statements that we use in order to take various inputs and perform some calculations or computations to obtain the output. This tutorial is specially dedicated to functions in the C/C++ programming language, that covers its syntax, types, significance, different ways to call a function, and types of function arguments. Let's start our journey with an example.



Suppose you want to compute the area of a right-angled triangle for a given set of 10 triangles. A novice at programming who does not have the knowledge of functions would probably say:

Define 10 separate variables to store the area of each triangle and then display it.

Sounds pretty inconvenient, right? What if the number of triangles whose area to be computed is increased to 30? It is not practically feasible to follow the traditional method. This is where the role of functions comes into play.

The above-mentioned problem can easily be resolved by defining a function to compute the area of a triangle taking 2 parameters into consideration, that is, base and height. Then, we can define a single variable that would return its area. The parameters of the 30 triangles in hand simply need to be passed to the function and we get their respective areas without much hassle.

Don't Forget to check [variables in C and C++ language](#)

Function in C and C++

In layman language, a function is anything that is done to achieve a specific purpose.

In programming, a function refers to the collection of statements that together performs a specific task. Functions give you the provision to divide your code into fragments to reduce code complexity, which you might have already inferred after pondering on the introductory problem statement.

Key takeaway: Every program in C and C++ has at least one function, that is, the **main()** function.

Syntax of Function

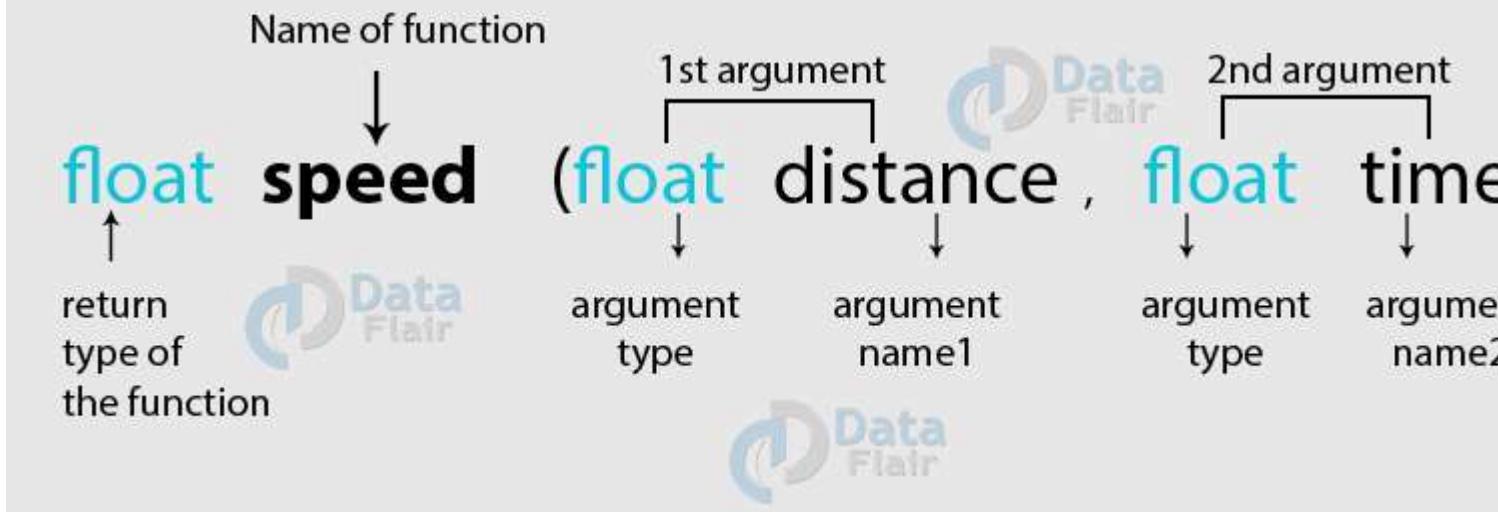
The basic syntax of a function in C/C++ has 2 parts, namely:

1. Function Declaration

return_type function_name(parameter list);

- The function declaration informs the C/C++ compiler about the return type and number of arguments/parameters the function generates.
- It is not necessary to initially declare a function. The function can directly be defined.

FUNCTION DECLARATION



2. Function Definition

Any function in C and C++ is defined in a similar way the main function is. Before the function name, we need to specify its return type. It depends on the wish of the programmer if he wants to pass any arguments to the function or not. The entire body of the function is enclosed within curly braces “{ }” and a set of statements is written that tells the function what to do.

1. return_type **function_name** (parameter list)
2. {
3. **statement(s)**
4. }

Types of Function in C and C++

Functions in C and C++ are broadly classified into two categories, according to the ones already available in the C/C++ compiler and the ones which can be defined and executed in run time. They are:

1. Standard Library Functions

- These functions are already pre-defined by the C/C++ compiler and hence cannot be modified. These functions act as pillars to the C and C++.
- They can be used to perform input and output operations using the **scanf()/cin** and **printf()/cout** function respectively, mathematical operations like finding the square root of a number or the power of a number using **sqrt()** or **pow()** function respectively, provided that the required header files are included.

Here is a program that illustrates the use of *standard library functions*, considering a mathematical function, sqrt() to compute the square root of a positive integer:

```
1. #include <stdio.h>
2. #include <math.h>
3.
4. int main()
5. {
6.
7.     printf("Welcome to DataFlair tutorials!\n\n");
8.     double number = 3, square_root;
9.
10.    square_root = sqrt(number);
11.    printf("The square root of %lf is: %lf", number, square_root);
12.    return 0;
13. }
```

Code on Screen

The screenshot shows a terminal window with a dark background. At the top, the title bar displays "dataflair@asus-System-Product-Name: ~". Below the title bar is a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal shows the file "fun1.c" being edited in the "GNU nano 2.9.3" editor. The code in the file is identical to the one shown in the previous code block. To the right of the code, the terminal prompt ends with a colon. Below the terminal window, there is a large black space.

```
File Edit View Search Terminal Help
GNU nano 2.9.3 fun1.c

#include <stdio.h>
#include <math.h>

int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");
    double number = 3, square_root;

    square_root = sqrt(number);
    printf("The square root of %lf is: %lf", number, square_root);
    return 0;
}
```

Output-

```
File Edit View Search Terminal Help
```

```
dataflair@admin4-H110M-H:~/Desktop$ gcc fun1.c -o fun1 -lm
```

```
dataflair@admin4-H110M-H:~/Desktop$ ./fun1
```

```
Welcome to DataFlair tutorials!
```

```
The square root of 3.000000 is: 1.732051dataflair@admin4-H110M-H:~/Desktop$ □
```

It's the right time to uncover the concept of [Recursion in C/C++](#)

Example of Standard Library Function in C++

Here is a program in C++ that illustrates the use of standard library functions, considering a mathematical function, `sqrt()` to compute the square root of a positive integer:

```
1. #include <iostream>
2. #include <math.h>
3.
4. using namespace std;
5.
6. int main()
7. {
```

```
8.  
9. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;  
10. double number = 3, square_root;  
11.  
12. square_root = sqrt(number);  
13. cout<<"The square root of " << number << " is: " << square_root<<endl;  
14.  
15. return 0;  
16. }
```

Code-

The screenshot shows a terminal window with the following details:

- Terminal title: dataflair@asus-System-Product-Name:
- File menu: File Edit View Search Terminal Help
- Version: GNU nano 2.9.3
- File name: fun1.cpp
- Code content:

```
#include <iostream>
#include <math.h>

using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
    double number = 3, square_root;

    square_root = sqrt(number);
    cout<<"The square root of " << number << " is: " << square_root<<endl;

    return 0;
}
```
- Output area (bottom):

Output-

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ g++ fun1.cpp -o fun1
^[[A^[[Adataflair@asus-System-Product-Name:~/Desktop$ ./fun1
Welcome to DataFlair tutorials!

The square root of 3 is: 1.73205
dataflair@asus-System-Product-Name:~/Desktop$ ]]
```

2. User-defined Functions

As discussed earlier, the C and C++ language give the programmer the provision to declare and define his own functions, according to his own convenience. Another benefit it offers is that the user-defined function can be added to the [standard library](#) for standard use.

Syntax of user defined function in C

```
#include <stdio.h>
void function_name()
{
    statement(s) // Body of the function
}
int main()
{
    statement(s)
    function_name(); // Function calling
    statement(s)
}
```

Example of

Get a cheat sheet for [Loops in C/C++](#)

Example of User-defined Function in C

Here is a code in C which illustrates the use of a **user-function in C** to compute the area of a right-angled triangle:

```
1. #include<stdio.h>
2. #include<math.h>
3.
4. double area_of_triangle(double,double); // function declaration
5.
6. int main()
7. {
8.
9.     printf("Welcome to DataFlair tutorials!\n\n");
10.
11.    double base, height, area;
12.    printf("Enter the base and height of the triangle respectively: ");
13.    scanf("%lf%lf", &base, &height);
14.    area = area_of_triangle(base, height);
15.    printf("The area of the triangle is: %.2lf\n", area);
16.    return 0;
17. }
18.
19. double area_of_triangle(double base, double height)
20. {
21.
22.     double area;
23.     area = 0.5*base*height;
24.     return area;
25. }
```

Code on Screen-

File Edit View Search Terminal Help

GNU nano 2.9.3

fun2.c

```
#include<stdio.h>
#include<math.h>

double area_of_triangle(double, double); // function declaration

int main()
{
    printf("Welcome to DataFlair tutorials!\n\n");

    double base, height, area;
    printf("Enter the base and height of the triangle respectively: ");
    scanf("%lf%lf", &base, &height);
    area = area_of_triangle(base, height);
    printf("The area of the triangle is: %.2lf\n", area);
    return 0;
}

double area_of_triangle(double base, double height)
{
    double area;
    area = 0.5*base*height;
    return area;
}
```

Output-

```
File Edit View Search Terminal Help
dataflair@admin4-H110M-H:~$ cd Desktop
dataflair@admin4-H110M-H:~/Desktop$ gcc fun2.c -o fun2
dataflair@admin4-H110M-H:~/Desktop$ ./fun2
Welcome to DataFlair tutorials!

Enter the base and height of the triangle respectively: 2 4
The area of the triangle is: 4.00
dataflair@admin4-H110M-H:~/Desktop$ 
```

Syntax of User-defined Function in C++

```
#include <iostream>
using namespace std;
void function_name()
{
    statement(s) // Body of the function
}
int main()
{
    statement(s)
```

```
function_name(); // Function calling  
statement(s)  
}
```

Example of User-defined Function in C++

Here is a code in C++ which illustrates the use of a user-function to compute the area of a right-angled triangle:

```
1. #include <iostream>  
2. #include <math.h>  
3. using namespace std;  
4.  
5. double area_of_triangle(double,double); // function declaration  
6.  
7. int main()  
8. {  
9. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;  
10.  
11. double base, height, area;  
12. cout<<"Enter the base and height of the triangle respectively: "  
13. cin>>base;  
14. cin>>height;  
15. area = area_of_triangle(base, height);  
16. cout<<"The area of the triangle is: "<< area <<endl;  
17. return 0;  
18. }  
19.  
20. double area_of_triangle(double base, double height)  
21. {  
22.  
23. double area;  
24. area = 0.5*base*height;  
25. return area;  
26. }
```

Code–

File Edit View Search Terminal Help

GNU nano 2.9.3

fun2.cpp

```
#include <iostream>
#include <math.h>
using namespace std;

double area_of_triangle(double, double); // function declaration

int main()
{
cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

double base, height, area;
cout<<"Enter the base and height of the triangle respectively: ";
cin>>base;
cin>>height;
area = area_of_triangle(base, height);
cout<<"The area of the triangle is: "<< area << endl;
return 0;
}

double area_of_triangle(double base, double height)
{
    double area;
    area = 0.5*base*height;
    return area;
}
```

Output-

dataflair@asus-System-Product-Name:

File Edit View Search Terminal Help

dataflair@asus-System-Product-Name:~/Desktop\$ g++ fun2.cpp -o fun2

dataflair@asus-System-Product-Name:~/Desktop\$./fun2

Welcome to DataFlair tutorials!

Enter the base and height of the triangle respectively: 2 4

The area of the triangle is: 4

dataflair@asus-System-Product-Name:~/Desktop\$ █

Key takeaway:

- Program execution starts from the **main()** function.
- As soon as the function is called in the main function, the flow of control goes over to the function and the function gets executed.
- After the function has been executed, the flow of control returns back to the next statement in the main() function.

In order to acquaint you with the syntax of functions, let's go back to the introductory problem.

For simplicity sake, let's consider only one set of a triangle.

Do you know what are the [reasons behind the popularity of C?](#)

Importance of Functions in C and C++

So far we saw a couple of codes for function in C and C++ which helped us gain an insight as to why functions are an integral part of a programming language.

To sum it up, here are the *salient features of functions in C/C++* which make it so appealing:

1. **Simple:** It is easy to comprehend, define and apply. It basically increases the readability of the code by dividing the code into smaller fragments.
2. **Control:** The flow of control from one part of the program to the other is made easier.
3. **Reusability:** The reusability of the code increases.
4. **Error-finding:** Debugging the program becomes easy as the entire program is divided into small fragments.
5. **Less-Redundant:** Redundancy is decreased making the program efficient and faster.
6. **Time-saving:** It saves a considerable amount of time.

Function Call Methods

In order to understand *what a function call is*, we need to understand the difference between actual parameters and formal parameters.

ACTUAL PARAMETER	FORMAL PARAMETER
We use while calling the function.	Use it in the function header.
We pass the actual values to the function definition through the function call.	We use it to receive the values that we pass to the function through function calls.
The values may be variable (global or local) or constant	The values are simply local variables

In C and C++, there are basically 2 methods to call a function:

1. **Call by value:** The actual parameters get copied to the formal parameters but both the parameters are created in different memory locations. Any changes that are made inside the functions are not reflected in other functions.
2. **Call by reference:** Instead of the actual value, the address of the value is passed to the function and both the parameters will be created in the same memory location. If changes are made inside the functions, it would be reflected in other functions.

Types of Function Arguments

The C and C++ gives the programmer the provision to define and use a function with or without arguments or with or without return types.

But it is important to note that in C/C++, functions cannot return arrays or other functions.

Considering the possibilities in hand, we can say that there are basically 4 types of function arguments are available. They are namely:

- Without arguments and without return value
- With arguments and without return value

- Without arguments and with a return value
- With arguments and with a return value

A Samurai Technique to Learn [Arrays in C and C++](#)

Math Functions in C and C++

The C/C++ Standard library offers a variety of in-built mathematical functions. Instead of deriving the logic of certain mathematical problems on our own, we can always choose the easier way out. C/C++ gives the programmer the provision to directly use some of the pre-defined functions accessible through the <math.h> header file.

Here are some of the functions available in the C Standard Library:

1. **pow (parameter1 , parameter2)** – The pow function calculates the power of a number to which it is raised. There are essentially 2 parameters, the first one in which we have to put the base value and the second one in which the exponent value.
2. **sqrt (parameter)** – The sqrt function computes the square root of a number entered as a parameter.
3. **abs (parameter)** – The abs function computes the absolute value of an integer. In other words, if a negative integer is entered, it becomes positive.
4. **ceil (parameter)** – The ceil function rounds off the value of a given number entered as a parameter. It returns an integral value either greater than or equal to that number.
5. **floor (parameter)** – The floor function also rounds off the value of a given number entered as a parameter. It returns an integral value either lesser than or equal to that number.
6. **fabs (parameter)** – Just like the abs function, the fabs function converts the given number into a positive number but the difference is that the fabs function works with floating-point values as well.
7. **log (parameter)** – The log function computes the logarithmic value of the given number with respect to the base taken as constant value ‘e’.
8. **log10 (parameter)** – The log10 function computes the logarithmic value of the given number with respect to the base taken as 10.
9. **fmod (parameter1, parameter2)** – The fmod function computes the remainder when the given value in the first parameter is divided by the given value in the second parameter.
10. **modf (parameter1, *parameter2)** – The modf function returns the fractional part of the given value (The decimal part of a number) entered as the first parameter.
11. **exp (parameter)** – The exp function computes the value of e to the power the value of the parameter entered, that is, eparameter.

12. **sin (parameter)** – The sin function computes the sine of a given number entered as the parameter.
13. **cos (parameter)** – The cos function computes the cosine of a given number entered as the parameter.
14. **tan (parameter)** – The tan function computes the tangent of a given number entered as the parameter.
15. **asin (parameter)** – The asin function computes the arc sine of a given number entered as the parameter.
16. **acos (parameter)** – The acos function computes the arc cosine of a given number entered as the parameter.
17. **atan (parameter)** – The atan function computes the arc tangent of a given number entered as the parameter.
18. **sinh (parameter)** – The sinh function computes the hyperbolic sine of the value entered as the parameter.
19. **cosh (parameter)** – The cosh function computes the hyperbolic cosine of the value entered as the parameter.
20. **tanh (parameter)** – The tanh function computes the hyperbolic tangent of the value entered as the parameter.

Learn [Virtual Function in C++](#) with Real-time Example

Key takeaway: All the trigonometric values should be in radian.

Example of Math Function in C

Here is a code in C that illustrates the use of some of the basic mathematical functions available in the C programming language.

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <math.h>
4.
5. int main()
6. {
7.
8.     printf("Welcome to DataFlair tutorials!\n\n");
9.
10.    printf("%f\n",ceil(1.8));
11.    printf("%f\n",ceil(5.2));
12.
13.    printf("%f\n",floor(4.9));
14.    printf("%f\n",floor(3.3));
15.
16.    printf("%f\n",sqrt(144));
17.    printf("%f\n",sqrt(50));
18.
19.    printf("%f\n",pow(2,10));
20.    printf("%f\n",pow(4,4));
21.
22.    printf("%d\n",abs(-18));
23.    printf("%d\n",abs(30));
```

```
25. return 0;  
26. }
```

Code-

dataflair@asus-System-Product-Name:

```
File Edit View Search Terminal Help  
GNU nano 2.9.3 mfun.c  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>  
  
int main()  
{  
  
printf("Welcome to DataFlair tutorials!\n\n");  
  
printf("%f\n", ceil(1.8));  
printf("%f\n", ceil(5.2));  
  
printf("%f\n", floor(4.9));  
printf("%f\n", floor(3.3));  
  
printf("%f\n", sqrt(144));  
printf("%f\n", sqrt(50));  
  
printf("%f\n", pow(2,10));  
printf("%f\n", pow(4,4));  
  
printf("%d\n", abs(-18));  
printf("%d\n", abs(30));  
  
return 0;  
}
```

Output-

dataflair@asus-System-Product-Name

```
File Edit View Search Terminal Help
dataflair@asus-System-Product-Name:~/Desktop$ gcc mfun.c -o mfun
dataflair@asus-System-Product-Name:~/Desktop$ ./mfun
Welcome to DataFlair tutorials!

2.000000
6.000000
4.000000
3.000000
12.000000
7.071068
1024.000000
256.000000
18
30
dataflair@asus-System-Product-Name:~/Desktop$ 
```

Since C++ is the superset of C, it is pretty obvious that C++ supports more functions than C. Here are some of the additional mathematical function included in <math.h> Library in C++:

1. **hypot (parameter1 , parameter2)** – The hypot() function computes the hypotenuse of a right-angled triangle by taking the other two known sides of the triangle as parameters.
2. **atan2 (parameter1 , parameter2)** – The atan2 function computes the tangent of parameter1/parameter2

Example of Math Function in C++

Here is a code in C++ that illustrates the use of some of the basic mathematical functions:

```
1. #include <iostream>
2. #include <stdlib.h>
3. #include<math.h>
4. using namespace std;
5.
6. int main()
7. {
```

```
8.  
9. cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;  
10.  
11. double a = 49;  
12. double b = 16;  
13. double c = 2;  
14. double d = -5;  
15. double x = 0.785398; // Value in radians of 45 degrees  
16. double f = 3;  
17. double g = 5;  
18.  
19. // Basic math functions  
20. cout << "The square root value of " << a << " is: " << sqrt(a) << endl;  
21. cout << b << " raised " << " to the power " << c << " is: " << pow(b, c) << endl;  
22. cout << "The absolute value of " << d << " is: " << abs(d) << endl;  
23. // Trigonometric functions  
24. cout << "The sine of " << x << " is: " << sin(x) << endl;  
25. cout << "The cosine of " << x << " is: " << cos(x) << endl;  
26. cout << "The tangent of " << x << " is: " << tan(x) << endl;  
27. cout << "The atan2 of " << b << " and " << c << " is: " << atan2(b,c) << endl;  
28. // Pythagoras  
29. cout<< "The hypotenuse of the right angled triangle of sides " << f << " and " << " g " << " is: " << hypot(f,g)<<endl;  
30.  
31. return 0;  
32. }
```

Code-

File Edit View Search Terminal Help

GNU nano 2.9.3

mfun.cpp

```
#include <iostream>
#include <stdlib.h>
#include<math.h>
using namespace std;

int main()
{
    cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;

    double a = 49;
    double b = 16;
    double c = 2;
    double d = -5;
    double x = 0.785398; // Value in radians of 45 degrees
    double f = 3;
    double g = 5;

    // Basic math functions
    cout << "The square root value of "<< a << " is: " << sqrt(a) << endl;
    cout << b << " raised " << " to the power " << c << " is: " << pow(b, c) << endl;
    cout << "The absolute value of " << d << " is: " << abs(d) << endl;
    // Trigonometric functions
    cout << "The sine of " << x << " is: " << sin(x) << endl;
    cout << "The cosine of " << x << " is: " << cos(x) << endl;
    cout << "The tangent of " << x << " is: " << tan(x) << endl;
    cout << "The atan2 of " << b << " and " << c << " is: " << atan2(b,c) << endl;
    // Pythagorean
    cout<< "The hypotenuse of the right angles triangle of sides "<< f << " and " << " g " << " is: " << endl;

    return 0;
}
```

Output-

```
dataflair@asus-System-Product-Name
```

```
File Edit View Search Terminal Help  
dataflair@asus-System-Product-Name:~/Desktop$ g++ mfun.cpp -o mfun  
dataflair@asus-System-Product-Name:~/Desktop$ ./mfun  
Welcome to DataFlair tutorials!  
  
The square root value of 49 is: 7  
16 raised to the power 2 is: 256  
The absolute value of -5 is: 5  
The sine of 0.785398 is: 0.707107  
The cosine of 0.785398 is: 0.707107  
The tangent of 0.785398 is: 1  
The atan2 of 16 and 2 is: 1.44644  
The hypotenuse of the right angles triangle of sides 3 and 4 is: 5.83095  
dataflair@asus-System-Product-Name:~/Desktop$
```

Summary

Now, you know the importance of functions in C and C++ and why we declare them. After this discussion, it is safe to say that C/C++ without functions is a sportsman without a ribcage. Hence it is crucial to understand the concept of functions to become a good programmer and become pioneers in the field of programming by developing your own functions.

