Computer Programming Tutorial

Computer programming is the act of writing computer programs, which are a sequence of instructions written using a Computer Programming Language to perform a specified task by the computer.

Computer Programming is fun and easy to learn provided you adopt a proper approach. This tutorial attempts to cover the basics of computer programming using a simple and practical approach for the benefit of novice learners.

Audience

This tutorial has been prepared for the beginners who are willing to learn computer programming but they are unable to learn it due to lack of proper guidance. We are confident that after completing this tutorial, you will be at a level where you can code in C Programming language and will have a basic understanding of Java and Python programming languages as well from where you can continue further.

If you are completely new to Computer Programming, then we recommend you to read this tutorial twice or even thrice. First reading will not give you much idea, but during your second reading, you will start grasping most of the concepts and you will enjoy writing computer programs.

Prerequisites

We do not expect much from you as prerequisites, however, we assume that you have some amount of exposure to computers and its peripherals like keyboard, mouse, screen, printer, etc.

Introduction to Computer Program

Before getting into computer programming, let us first understand computer programs and what they do.

A computer program is a sequence of instructions written using a Computer Programming Language to perform a specified task by the computer.

The two important terms that we have used in the above definition are −

- Sequence of instructions
- Computer Programming Language

To understand these terms, consider a situation when someone asks you about how to go to a nearby KFC. What exactly do you do to tell him the way to go to KFC?

You will use Human Language to tell the way to go to KFC, something as follows −

First go straight, after half kilometer, take left from the red light and then drive around one kilometer and you will find KFC at the right.

Here, you have used English Language to give several steps to be taken to reach KFC. If they are followed in the following sequence, then you will reach KFC −

```
1.      Go straight
2.      Drive half kilometer
3.      Take left
4.      Drive around one kilometer
5.      Search for KFC at your right side
```

Now, try to map the situation with a computer program. The above sequence of instructions is actually a **Human Program** written in **English Language**, which instructs on how to reach KFC from a given starting point. This same sequence could have been given in Spanish, Hindi, Arabic, or any other human language, provided the person seeking direction knows any of these languages.

Now, let's go back and try to understand a computer program, which is a sequence of instructions written in a Computer Language to perform a specified task by the computer. Following is a simple program written in **Python** programming Language −

```
print "Hello, World!"
```

The above computer program instructs the computer to print "Hello, World!" on the computer screen.

- A computer program is also called a **computer software**, which can range from two lines to millions of lines of instructions.
- Computer program instructions are also called program source code and **computer programming** is also called **program coding**.
- A computer without a computer program is just a dump box; it is programs that make computers active.

As we have developed so many languages to communicate among ourselves, computer scientists have developed several computer-programming languages to provide instructions to the computer (i.e., to write computer programs). We will see several computer programming languages in the subsequent chapters.

Introduction to Computer Programming

If you understood what a **computer program** is, then we will say: *the act of writing computer programs is called computer programming.*

As we mentioned earlier, there are hundreds of programming languages, which can be used to write computer programs and following are a few of them −

- Java
- C

- C++
- Python
- PHP
- Perl
- Ruby

Uses of Computer Programs

Today computer programs are being used in almost every field, household, agriculture, medical, entertainment, defense, communication, etc. Listed below are a few applications of computer programs −

- MS Word, MS Excel, Adobe Photoshop, Internet Explorer, Chrome, etc., are examples of computer programs.
- Computer programs are being used to develop graphics and special effects in movie making.
- Computer programs are being used to perform Ultrasounds, X-Rays, and other medical examinations.
- Computer programs are being used in our mobile phones for SMS, Chat, and voice communication.

Computer Programmer

Someone who can write computer programs or in other words, someone who can do computer programming is called a Computer Programmer.

Based on computer programming language expertise, we can name a computer programmers as follows −

- C Programmer
- C++ Programmer
- Java Programmer
- Python Programmer
- PHP Programmer
- Perl Programmer
- Ruby Programmer

Algorithm

From programming point of view, an **algorithm** is a step-by-step procedure to resolve any problem. An algorithm is an effective method expressed as a finite set of well-defined instructions.

Thus, a computer programmer lists down all the steps required to resolve a problem before writing the actual code. Following is a simple example of an algorithm to find out the largest number from a given list of numbers −

```
1. Get a list of numbers L₁, L₂, L₃....Lₙ
2. Assume L₁ is the largest, Largest = L₁
3. Take next number Lᵢ from the list and do the following
```

```
4. If Largest is less than Lᵢ
5. Largest = Lᵢ
6. If Lᵢ is last number from the list then
7. Print value stored in Largest and come out
8. Else repeat same process starting from step 3
```

The above algorithm has been written in a crude way to help beginners understand the concept. You will come across more standardized ways of writing computer algorithms as you move on to advanced levels of computer programming.

Computer Programming - Basics

We assume you are well aware of English Language, which is a well-known **Human Interface Language**. English has a predefined grammar, which needs to be followed to write English statements in a correct way. Likewise, most of the Human Interface Languages (Hindi, English, Spanish, French, etc.) are made of several elements like verbs, nouns, adjectives, adverbs, propositions, and conjunctions, etc.

Similar to Human Interface Languages, Computer Programming Languages are also made of several elements. We will take you through the basics of those elements and make you comfortable to use them in various programming languages. These basic elements include −

- Programming Environment
- Basic Syntax
- Data Types
- Variables
- Keywords
- Basic Operators
- Decision Making
- Loops
- Numbers
- Characters
- Arrays
- Strings
- Functions
- File I/O

We will explain all these elements in subsequent chapters with examples using different programming languages. First, we will try to understand the meaning of all these terms in general and then, we will see how these terms can be used in different programming languages.

This tutorial has been designed to give you an idea about the following most popular programming languages −

- C Programming
- Java Programming

- Python Programming

A major part of the tutorial has been explained by taking C as programming language and then we have shown how similar concepts work in Java and Python. So after completion of this tutorial, you will be quite familiar with these popular programming languages.

Computer Programming - Environment

Though Environment Setup is not an element of any Programming Language, it is the first step to be followed before setting on to write a program.

When we say Environment Setup, it simply implies a base on top of which we can do our programming. Thus, we need to have the required software setup, i.e., installation on our PC which will be used to write computer programs, compile, and execute them. For example, if you need to browse Internet, then you need the following setup on your machine −

- A working Internet connection to connect to the Internet
- A Web browser such as Internet Explorer, Chrome, Safari, etc.

If you are a PC user, then you will recognize the following screenshot, which we have taken from the Internet Explorer while browsing tutorialspoint.com.

Similarly, you will need the following setup to start with programming using any programming language.

- A text editor to create computer programs.
- A compiler to compile the programs into binary format.
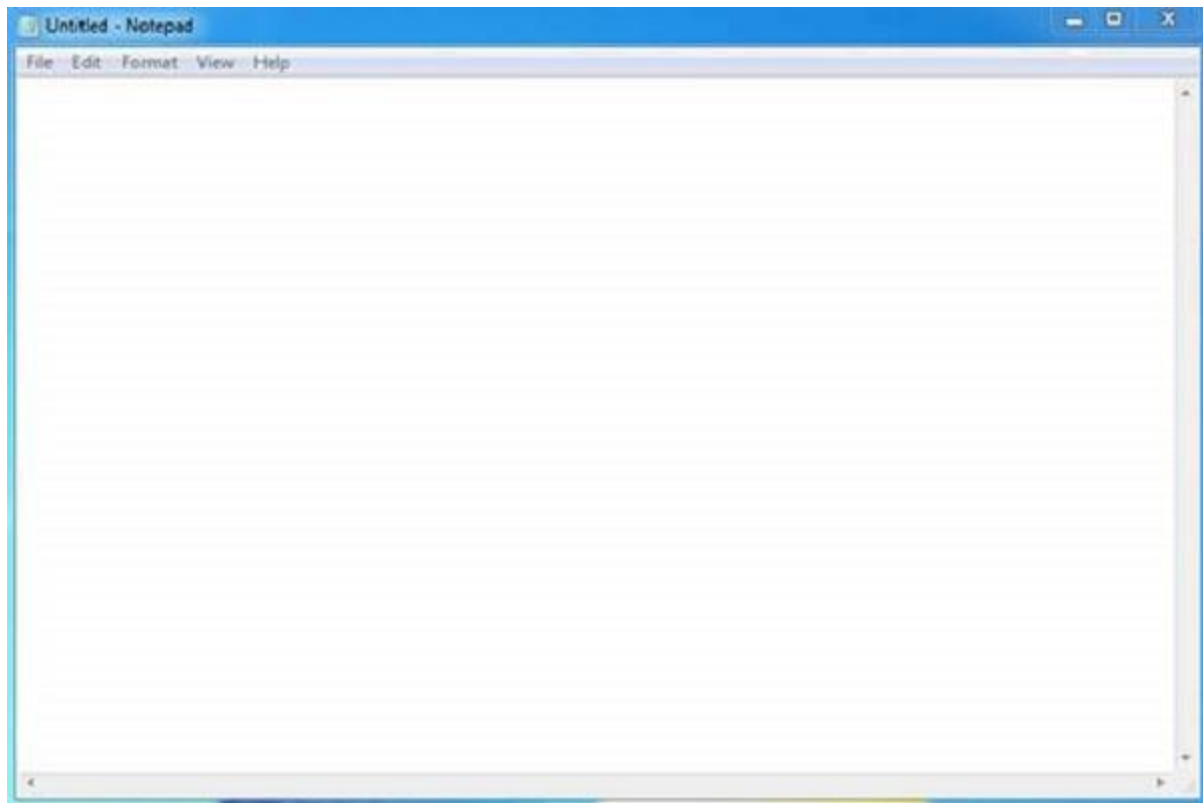- An interpreter to execute the programs directly.

In case you don't have sufficient exposure to computers, you will not be able to set up either of these software. So, we suggest you take the help from any technical person around you to set up the programming environment on your machine from where you can start. But for you, it is important to understand what these items are.

Text Editor

A text editor is a software that is used to write computer programs. Your Windows machine must have a Notepad, which can be used to type programs. You can launch it by following these steps −

```
Start Icon → All Programs → Accessories → Notepad → Mouse Click on Notepad
```

It will launch Notepad with the following window −

You can use this software to type your computer program and save it in a file at any location. You can download and install other good editors like **Notepad++**, which is freely available.

If you are a Mac user, then you will have **TextEdit** or you can install some other commercial editor like **BBEdit** to start with.
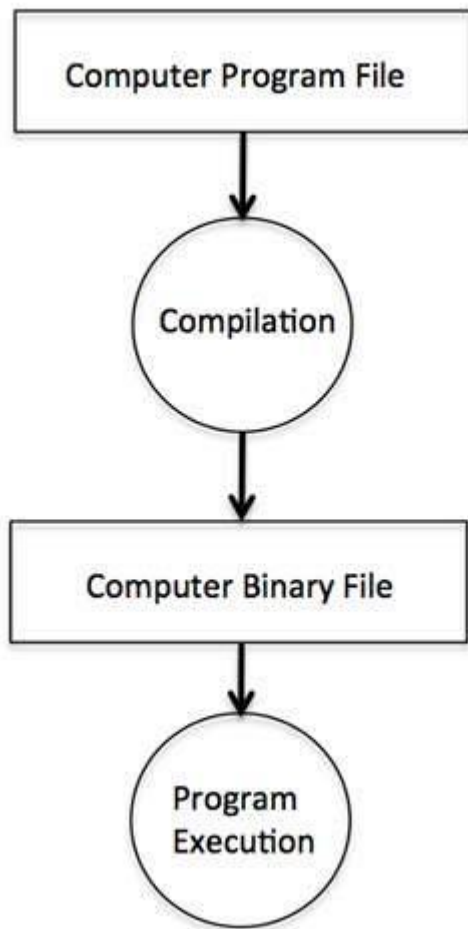
Compiler?

You write your computer program using your favorite programming language and save it in a text file called the program file.

Now let us try to get a little more detail on how the computer understands a program written by you using a programming language. Actually, the computer cannot understand your program directly given in the text format, so we need to convert this program in a binary format, which can be understood by the computer.

The conversion from text program to binary file is done by another software called Compiler and this process of conversion from text formatted program to binary format file is called program compilation. Finally, you can execute binary file to perform the programmed task.

We are not going into the details of a compiler and the different phases of compilation.

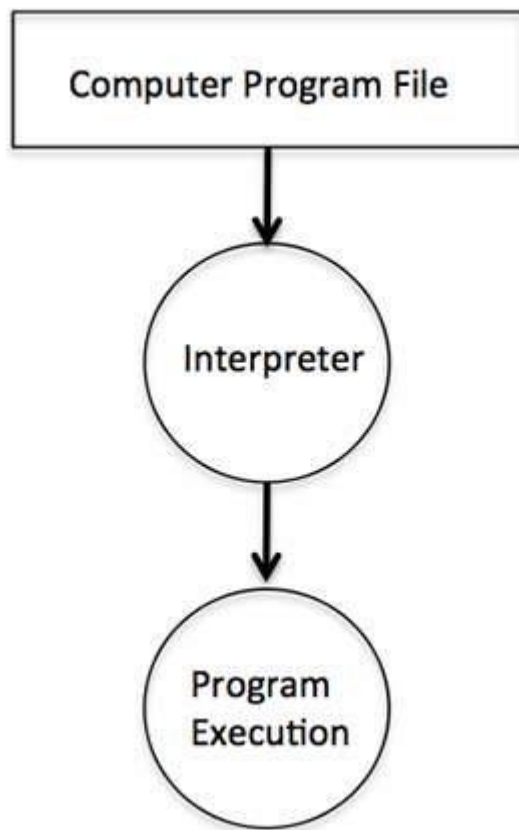The following flow diagram gives an illustration of the process −

So, if you are going to write your program in any such language, which needs compilation like C, C++, Java and Pascal, etc., then you will need to install their compilers before you start programming.

Interpreter

We just discussed about compilers and the compilation process. Compilers are required in case you are going to write your program in a programming language that needs to be compiled into binary format before its execution.

There are other programming languages such as Python, PHP, and Perl, which do not need any compilation into binary format, rather an interpreter can be used to read such programs line by line and execute them directly without any further conversion.

So, if you are going to write your programs in PHP, Python, Perl, Ruby, etc., then you will need to install their interpreters before you start programming.

Online Compilation

If you are not able to set up any editor, compiler, or interpreter on your machine, then *tutorialspoint.com* provides a facility to compile and run almost all the programs online with an ease of a single click.

So do not worry and let's proceed further to have a thrilling experience to become a computer programmer in simple and easy steps.

Computer Programming - Basic Syntax

Let's start with a little coding, which will really make you a computer programmer. We are going to write a single-line computer program to write **Hello, World!** on your screen. Let's see how it can be written using different programming languages.

Hello World Program in C

Try the following example using our online compiler option available at
www.compileonline.com.

For most of the examples given in this tutorial, you will find a **Try it** option in our website code sections at the top right corner that will take you to the online compiler.

Try to change the content inside printf(), i.e., type anything in place of **Hello World!** and then check its result. It just prints whatever you keep inside the two double quotes.

```c
#include <stdio.h>

main() {
   /* printf() function to write Hello, World! */
   printf( "Hello, World!" );
}
```

This little Hello World program will help us understand various basic concepts related to C Programming.

Program Entry Point

For now, just forget about the **#include <stdio.h>** statement, but keep a note that you have to put this statement at the top of a C program.

Every C program starts with main(), which is called the main function, and then it is followed by a left curly brace. The rest of the program instruction is written in between and finally a right curly brace ends the program.

The coding part inside these two curly braces is called the program body. The left curly brace can be in the same line as main(){ or in the next line like it has been mentioned in the above program.

Functions

Functions are small units of programs and they are used to carry out a specific task. For example, the above program makes use of two functions: **main**() and **printf**(). Here, the function main() provides the entry point for the program execution and the other function printf() is being used to print an information on the computer screen.

You can write your own functions which we will see in a separate chapter, but C programming itself provides various built-in functions like main(), printf(), etc., which we can use in our programs based on our requirement.

Some of the programming languages use the word **sub-routine** instead of function, but their functionality is more or less the same.

Comments

A C program can have statements enclosed inside **/*.....*/**. Such statements are called comments and these comments are used to make the programs user friendly and easy to understand. The good thing about comments is that they are completely ignored by compilers and interpreters. So you can use whatever language you want to write your comments.

Whitespaces

When we write a program using any programming language, we use various printable characters to prepare programming statements. These printable characters are **a, b, c,......z, A, B, C,.....Z, 1, 2, 3,...... 0, !, @, #, $, %, ^, &, *, (,), -, _, +, =, \, |, {, }, [,], :, ;, <, >, ?, /, \, ~. `. ", '**. Hope I'm not missing any printable characters from your keyboard.

Apart from these characters, there are some characters which we use very frequently but they are invisible in your program and these characters are spaces, tabs (\t), new lines(\n). These characters are called **whitespaces**.

These three important whitespace characters are common in all the programming languages and they remain invisible in your text document −

Whitespace	Explanation	Representation
New Line	To create a new line	\n
Tab	To create a tab.	\t
Space	To create a space.	empty space

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it. Whitespace is the term used in C to describe blanks, tabs, newline characters, and comments. So you can write **printf("Hello, World!");** as shown below. Here all the created spaces around "Hello, World!" are useless and the compiler will ignore them at the time of compilation.

```
#include <stdio.h>

main() {

   /* printf() function to write Hello, World! */

   printf(    "Hello, World!"       );

}
```

If we make all these whitespace characters visible, then the above program will look like this and you will not be able to compile it −

```
#include <stdio.h>\n
\n
main()\n
```

```
{
\n
\t/* printf() function to write Hello, World! */
\n
\tprintf(\t"Hello, World!"\t);\n
\n
}\n
```

Semicolons

Every individual statement in a C Program must be ended with a semicolon (**;**), for example, if you want to write "Hello, World!" twice, then it will be written as follows −

```
#include <stdio.h>

main() {
   /* printf() function to write Hello, World! */
   printf( "Hello, World!\n" );
   printf( "Hello, World!" );
}
```

This program will produce the following result −

```
Hello, World!
Hello, World!
```

Here, we are using a new line character **\n** in the first printf() function to create a new line. Let us see what happens if we do not use this new line character −

```
#include <stdio.h>

main() {
   /* printf() function to write Hello, World! */
   printf( "Hello, World!" );
   printf( "Hello, World!" );
}
```

This program will produce the following result −

```
Hello, World! Hello, World!
```

We will learn identifiers and keywords in next few chapters.

Program Explanation

Let us understand how the above C program works. First of all, the above program is converted into a binary format using C compiler. So let's put this code in test.c file and compile it as follows −

```
$gcc test.c -o demo
```

If there is any grammatical error (Syntax errors in computer terminologies), then we fix it before converting it into binary format. If everything goes fine, then it produces a binary file called **demo**. Finally, we execute the produced binary demo as follows −

```
$./demo
```

which produces the following result −

```
Hello, World!
```

Here, when we execute the binary **a.out** file, the computer enters inside the program starting from main() and encounters a printf() statement. Keep a note that the line inside /*....*/ is a comment and it is filtered at the time of compilation. So printf() function instructs the computer to print the given line at the computer screen. Finally, it encounters a right curly brace which indicates the end of main() function and exits the program.

Syntax Error

If you do not follow the rules defined by the programing language, then at the time of compilation, you will get syntax errors and the program will not be compiled. From syntax point of view, even a single dot or comma or a single semicolon matters and you should take care of such small syntax as well. In the following example, we have skipped a semicolon, let's try to compile the program −

```c
#include <stdio.h>

main() {

   printf("Hello, World!")

}
```

This program will produce the following result −

```
main.c: In function 'main':
main.c:7:1: error: expected ';' before '}' token
 }
 ^
```

So the bottom-line is that if you are not following proper syntax defined by the programming language in your program, then you will get syntax errors. Before attempting another compilation, you will need to fix them and then proceed.

Hello World Program in Java

Following is the equivalent program written in Java. This program will also produce the same result **Hello, World!**.

```java
public class HelloWorld {
   public static void main(String []args) {
      /* println() function to write Hello, World! */
      System.out.println("Hello, World!");
   }
}
```

Following is the equivalent program written in Python. This program will also produce the same result **Hello, World!**.

```
#  print function to write Hello, World! */
print "Hello, World!"
```

Hope you noted that for C and Java examples, first we are compiling the programs and then executing the produced binaries, but in Python program, we are directly executing it. As we explained in the previous chapter, Python is an interpreted language and it does not need an intermediate step called compilation.

Python does not require a semicolon (;) to terminate a statement, rather a new line always means termination of the statement.

Computer Programming - Data Types

Let's discuss about a very simple but very important concept available in almost all the programming languages which is called **data types**. As its name indicates, a data type represents a type of the data which you can process using your computer program. It can be numeric, alphanumeric, decimal, etc.

Let's keep Computer Programming aside for a while and take an easy example of adding two whole numbers 10 & 20, which can be done simply as follows −

```
10 + 20
```

Let's take another problem where we want to add two decimal numbers 10.50 & 20.50, which will be written as follows −

```
10.50 + 20.50
```

The two examples are straightforward. Now let's take another example where we want to record student information in a notebook. Here we would like to record the following information −

```
Name:
Class:
Section:
Age:
Sex:
```

Now, let's put one student record as per the given requirement −

```
Name: Zara Ali
Class: 6th
Section: J
Age: 13
Sex: F
```

The first example dealt with whole numbers, the second example added two decimal numbers, whereas the third example is dealing with a mix of different data. Let's put it as follows −

- Student name "Zara Ali" is a sequence of characters which is also called a string.
- Student class "6th" has been represented by a mix of whole number and a string of two characters. Such a mix is called alphanumeric.
- Student section has been represented by a single character which is 'J'.
- Student age has been represented by a whole number which is 13.
- Student sex has been represented by a single character which is 'F'.

This way, we realized that in our day-to-day life, we deal with different types of data such as strings, characters, whole numbers (integers), and decimal numbers (floating point numbers).

Similarly, when we write a computer program to process different types of data, we need to specify its type clearly; otherwise the computer does not understand how different operations can be performed on that given data. Different programming languages use different keywords to specify different data types. For example, C and Java programming languages use **int** to specify integer data, whereas **char** specifies a character data type.

Subsequent chapters will show you how to use different data types in different situations. For now, let's check the important data types available in C, Java, and Python and the keywords we will use to specify those data types.

C and Java Data Types

C and Java support almost the same set of data types, though Java supports additional data types. For now, we are taking a few common data types supported by both the programming languages −

Type	Keyword	Value range which can be represented by this data type
Character	char	-128 to 127 or 0 to 255
Number	int	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
Small Number	short	-32,768 to 32,767
Long Number	long	-2,147,483,648 to 2,147,483,647
Decimal Number	float	1.2E-38 to 3.4E+38 till 6 decimal places

These data types are called primitive data types and you can use these data types to build more complex data types, which are called user-defined data type, for example a string will be a sequence of characters.

Python has five standard data types but this programming language does not make use of any keyword to specify a particular data type, rather Python is intelligent enough to understand a given data type automatically.

- Numbers
- String
- List
- Tuple
- Dictionary

Here, Number specifies all types of numbers including decimal numbers and string represents a sequence of characters with a length of 1 or more characters. For now, let's proceed with these two data types and skip List, Tuple, and Dictionary, which are advanced data types in Python.

Computer Programming - Variables

Variables are the names you give to computer memory locations which are used to store values in a computer program.

For example, assume you want to store two values 10 and 20 in your program and at a later stage, you want to use these two values. Let's see how you will do it. Here are the following three simple steps −

- Create variables with appropriate names.
- Store your values in those two variables.
- Retrieve and use the stored values from the variables.

Creating variables

Creating variables is also called **declaring variables** in C programming. Different programming languages have different ways of creating variables inside a program. For example, C programming has the following simple way of creating variables −

```
#include <stdio.h>

main() {
   int a;
   int b;
}
```

The above program creates two variables to reserve two memory locations with names **a** and **b**. We created these variables using **int** keyword to specify variable **data type** which means we want to store integer values in these two variables. Similarly, you can create variables to store **long**, **float**, **char** or any other data type. For example −

```
/* variable to store long value */
long a;
```

```
/* variable to store float value */
float b;
```

You can create variables of similar type by putting them in a single line but separated by comma as follows −

```
#include <stdio.h>

main() {
   int a, b;
}
```

Listed below are the key points about variables that you need to keep in mind −

- A variable name can hold a single type of value. For example, if variable **a** has been defined **int** type, then it can store only integer.
- C programming language requires a variable creation, i.e., declaration before its usage in your program. You cannot use a variable name in your program without creating it, though programming language like Python allows you to use a variable name without creating it.
- You can use a variable name only once inside your program. For example, if a variable **a** has been defined to store an integer value, then you cannot define **a** again to store any other type of value.
- There are programming languages like Python, PHP, Perl, etc., which do not want you to specify data type at the time of creating variables. So you can store integer, float, or long without specifying their data type.
- You can give any name to a variable like **age**, **sex**, **salary**, **year1990** or anything else you like to give, but most of the programming languages allow to use only limited characters in their variables names. For now, we will suggest to use only **a....z, A....Z, 0....9** in your variable names and start their names using alphabets only instead of digits.
- Almost none of the programming languages allow to start their variable names with a digit, so **1990year** will not be a valid variable name whereas **year1990** or **ye1990ar** are valid variable names.

Every programming language provides more rules related to variables and you will learn them when you will go in further detail of that programming language.

Store Values in Variables

You have seen how we created variables in the previous section. Now, let's store some values in those variables −

```
#include <stdio.h>

main() {

   int a;
   int b;

   a = 10;
   b = 20;
```

```
}
```

The above program has two additional statements where we are storing 10 in variable **a** and 20 is being stored in variable **b**. Almost all the programming languages have similar way of storing values in variable where we keep variable name in the left hand side of an equal sign = and whatever value we want to store in the variable, we keep that value in the right hand side.

Now, we have completed two steps, first we created two variables and then we stored required values in those variables. Now variable **a** has value 10 and variable **b** has value 20. In other words we can say, when above program is executed, the memory location named **a** will hold 10 and memory location **b** will hold 20.

Access stored values in variables

If we do not use the stored values in the variables, then there is no point in creating variables and storing values in them. We know that the above program has two variables **a** and **b** and they store the values 10 and 20, respectively. So let's try to print the values stored in these two variables. Following is a C program, which prints the values stored in its variables −

```c
#include <stdio.h>

main() {

   int a;
   int b;

   a = 10;
   b = 20;

   printf( "Value of a = %d\n", a );
   printf( "Value of b = %d\n", b );

}
```

When the above program is executed, it produces the following result −

```
Value of a = 10
Value of b = 20
```

You must have seen **printf()** function in the previous chapter where we had used it to print "Hello, World!". This time, we are using it to print the values of variables. We are making use of **%d**, which will be replaced with the values of the given variable in printf() statements. We can print both the values using a single printf() statement as follows −

```c
#include <stdio.h>

main() {

   int a;
   int b;

   a = 10;
   b = 20;
```

```
   printf( "Value of a = %d and value of b = %d\n", a, b );
}
```

When the above program is executed, it produces the following result −

```
Value of a = 10 and value of b = 20
```

If you want to use **float** variable in C programming, then you will have to use **%f** instead of **%d**, and if you want to print a character value, then you will have to use **%c**. Similarly, different data types can be printed using different % and characters.

Variables in Java

Following is the equivalent program written in Java programming language. This program will create two variables **a** and **b** and very similar to C programming, it will assign 10 and 20 in these variables and finally print the values of the two variables in two ways −

```
public class DemoJava {
   public static void main(String []args) {

      int a;
      int b;

      a = 10;
      b = 20;

      System.out.println("Value of a = " + a);
      System.out.println("Value of b = " + b);

      System.out.println("Value of a = " + a + " and value of b = " + b);
   }
}
```

Variables in Python

Following is the equivalent program written in Python. This program will create two variables **a** and **b** and at the same time, assign 10 and 20 in those variables.

Python does not want you to specify the data type at the time of variable creation and there is no need to create variables in advance.

```
a = 10
b = 20

print "Value of a = ", a
print "Value of b = ", b

print "Value of a = ", a, " and value of b = ", b
```

You can use the following syntax in C and Java programming to declare variables and assign values at the same time −

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

```
main() {
   int a = 10;
   int b = 20;

   printf( "Value of a = %d and value of b = %d\n", a, b );
}
```

Computer Programming - Keywords

So far, we have covered two important concepts called variables and their data types. We discussed how to use **int**, **long**, and **float** to specify different data types. We also learnt how to name the variables to store different values.

Though this chapter is not required separately because reserved keywords are a part of basic programming syntax, we kept it separate to explain it right after data types and variables to make it easy to understand.

Like int, long, and float, there are many other keywords supported by C programming language which we will use for different purpose. Different programming languages provide different set of reserved keywords, but there is one important & common rule in all the programming languages that we cannot use a reserved keyword to name our variables, which means we cannot name our variable like **int** or **float** rather these keywords can only be used to specify a variable data type.

For example, if you will try to use any reserved keyword for the purpose of variable name, then you will get a syntax error.

```
#include <stdio.h>

main() {

   int float;

   float = 10;

   printf( "Value of float = %d\n", float);
}
```

When you compile the above program, it produces the following error −

```
main.c: In function 'main':
main.c:5:8: error: two or more data types in declaration specifiers
    int float;
......
```

Let's now give a proper name to our integer variable, then the above program should compile and execute successfully −

```
#include <stdio.h>

main() {
   int count;
```

```
   count = 10;

   printf( "Value of count = %d\n", count);
}
```

C Programming Reserved Keywords

Here is a table having almost all the keywords supported by C Programming language −

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

Java Programming Reserved Keywords

Here is a table having almost all the keywords supported by Java Programming language −

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw

throws	transient	try	void
volatile	while		

Here is a table having almost all the keywords supported by Python Programming language −

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

We know you cannot memorize all these keywords, but we have listed them down for your reference purpose and to explain the concept of **reserved keywords**. So just be careful while giving a name to your variable, you should not use any reserved keyword for that programming language.

Computer Programming - Operators

An operator in a programming language is a symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce final result. This chapter will explain the concept of **operators** and it will take you through the important arithmetic and relational operators available in C, Java, and Python.

Arithmetic Operators

Computer programs are widely used for mathematical calculations. We can write a computer program which can do simple calculation like adding two numbers $(2 + 3)$ and we can also write a program, which can solve a complex equation like $P(x) = x^4 + 7x^3 - 5x + 9$. If you have been even a poor student, you must be aware that in first expression 2 and 3 are operands and + is an operator. Similar concepts exist in Computer Programming.

Take a look at the following two examples −

```
2 + 3
```

```
P(x) = x⁴ + 7x³ − 5x + 9.
```

These two statements are called arithmetic expressions in a programming language and **plus**, **minus** used in these expressions are called arithmetic operators and the values used in these expressions like 2, 3 and x, etc., are called operands. In their simplest form, such expressions produce numerical results.

Similarly, a programming language provides various arithmetic operators. The following table lists down a few of the important arithmetic operators available in C programming language. Assume variable A holds 10 and variable B holds 20, then −

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	This gives remainder of an integer division	B % A will give 0

Following is a simple example of C Programming to understand the above mathematical operators −

```c
#include <stdio.h>

main() {

   int a, b, c;

   a = 10;
   b = 20;

   c = a + b;
   printf( "Value of c = %d\n", c);

   c = a - b;
   printf( "Value of c = %d\n", c);

   c = a * b;
   printf( "Value of c = %d\n", c);

   c = b / a;
   printf( "Value of c = %d\n", c);

   c = b % a;
   printf( "Value of c = %d\n", c);
}
```

When the above program is executed, it produces the following result −

```
Value of c = 30
Value of c = -10
Value of c = 200
Value of c = 2
Value of c = 0
```

Relational Operators

Consider a situation where we create two variables and assign them some values as follows −

```
A = 20
B = 10
```

Here, it is obvious that variable A is greater than B in values. So, we need the help of some symbols to write such expressions which are called relational expressions. If we use C programming language, then it will be written as follows −

```
(A > B)
```

Here, we used a symbol > and it is called a relational operator and in their simplest form, they produce Boolean results which means the result will be either true or false. Similarly, a programming language provides various relational operators. The following table lists down a few of the important relational operators available in C programming language. Assume variable **A** holds 10 and variable **B** holds 20, then −

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Here, we will show you one example of C Programming which makes use of **if conditional statement**. Though this statement will be discussed later in a separate chapter, but in short, we use **if statement** to check a condition and if the condition is true, then the body of **if statement** is executed, otherwise the body of **if statement** is skipped.

```c
#include <stdio.h>

main() {

   int a, b;

   a = 10;
   b = 20;

   /* Here we check whether a is equal to 10 or not */
   if( a == 10 ) {

      /* if a is equal to 10 then this body will be executed */
      printf( "a is equal to 10\n");
   }

   /* Here we check whether b is equal to 10 or not */
   if( b == 10 ) {

      /* if b is equal to 10 then this body will be executed */
      printf( "b is equal to 10\n");
   }

   /* Here we check if a is less b than or not */
   if( a < b ) {

      /* if a is less than b then this body will be executed */
      printf( "a is less than b\n");
   }

   /* Here we check whether a and b are not equal */
   if( a != b ) {

      /* if a is not equal to b then this body will be executed */
      printf( "a is not equal to b\n");
   }
}
```

When the above program is executed, it produces the following result −

```
a is equal to 10
a is less than b
a is not equal to b
```

Logical Operators

Logical operators are very important in any programming language and they help us take decisions based on certain conditions. Suppose we want to combine the result of two conditions, then logical AND and OR logical operators help us in producing the final result.

The following table shows all the logical operators supported by the C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Try the following example to understand all the logical operators available in C programming language −

```c
#include <stdio.h>

main() {

    int a = 1;
    int b = 0;

    if ( a && b ) {

        printf("This will never print because condition is false\n" );
    }
    if ( a || b ) {

        printf("This will be printed print because condition is true\n" );
    }

    if ( !(a && b) ) {

        printf("This will be printed print because condition is true\n" );
    }
}
```

When you compile and execute the above program, it produces the following result −

```
This will be printed print because condition is true
This will be printed print because condition is true
```

Operators in Java

Following is the equivalent program written in Java. C programming and Java provide almost identical set of operators and conditional statements. This program will create two variables **a** and **b**, very similar to C programming, then we assign 10 and 20 in these variables and finally, we will use different arithmetic and relational operators −

You can try to execute the following program to see the output, which must be identical to the result generated by the above example.

```java
public class DemoJava {

    public static void main(String []args) {
```

```
        int a, b, c;

        a = 10;
        b = 20;

        c = a + b;
        System.out.println("Value of c = " + c );

        c = a - b;
        System.out.println("Value of c = " + c );


        c = a * b;
        System.out.println("Value of c = " + c );

        c = b / a;
        System.out.println("Value of c = " + c );

        c = b % a;
        System.out.println("Value of c = " + c );

        if( a == 10 ) {

            System.out.println("a is equal to 10" );
        }

    }
}
```

Operators in Python

Following is the equivalent program written in Python. This program will create two variables **a** and **b** and at the same time, assign 10 and 20 in those variables. Fortunately, C programming and Python programming languages provide almost identical set of operators. This program will create two variables **a** and **b**, very similar to C programming, then we assign 10 and 20 in these variables and finally, we will use different arithmetic and relational operators.

You can try to execute the following program to see the output, which must be identical to the result generated by the above example.

```
a = 10
b = 20

c = a + b
print "Value of c = ", c

c = a - b
print "Value of c = ", c

c = a * b
print "Value of c = ", c

c = a / b
print "Value of c = ", c

c = a % b
print "Value of c = ", c
```

```
if( a == 10 ):
    print "a is equal to 10"
```

Computer Programming - Decision

Decision making is critical to computer programming. There will be many situations when you will be given two or more options and you will have to select an option based on the given conditions. For example, we want to print a remark about a student based on his secured marks. Following is the situation −
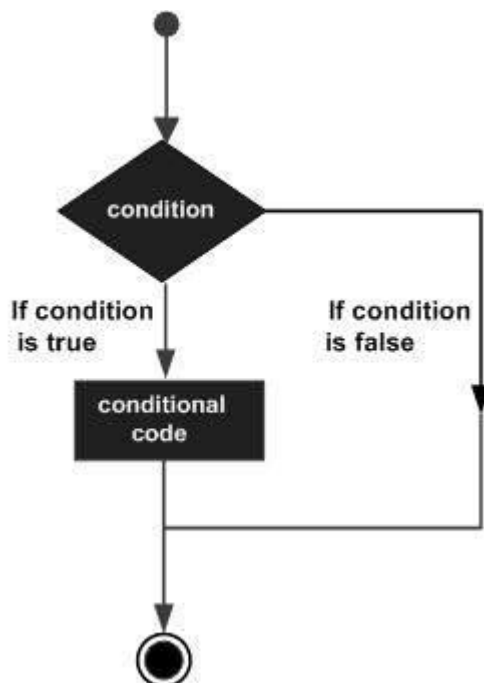
```
Assume given marks are x for a student:

If given marks are more than 95, then
Student is brilliant

If given marks are less than 30, then
Student is poor

If given marks are less than 95 and more than 30, then
Student is average
```

Now, the question is how to write a programming code to handle such situations. Almost all the programming languages provide conditional statements that work based on the following flow diagram −

Let's write a C program with the help of **if conditional statements** to convert the above given situation into a programming code −

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

```
main() {

    int x = 45;

    if( x > 95) {

        printf( "Student is brilliant\n");
    }

    if( x < 30) {

        printf( "Student is poor\n");
    }

    if( x < 95 && x > 30 ) {

        printf( "Student is average\n");
    }
}
```

When the above program is executed, it produces the following result −

```
Student is average
```

The above program uses **if conditional statements**. Here, the first **if statement** checks whether the given condition i.e., variable x is greater than 95 or not and if it finds the condition is true, then the conditional body is entered to execute the given statements. Here we have only one *printf()* statement to print a remark about the student.

Similarly, the second **if statement** works. Finally, the third **if statement** is executed, here we have the following two conditions −

- First condition is **x > 95**
- Second condition is **x < 30**

The computer evaluates both the given conditions and then, the overall result is combined with the help of the binary operator **&&**. If the final result is true, then the conditional statement will be executed, otherwise no statement will be executed.

This tutorial will give you a basic idea on various forms of **if statements** and an introduction to **switch** statements available in C programming language. Different programming languages provide different types of decision-making statements, but the basic concept remains the same as explained in this tutorial.

if...else statement

An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false. The syntax of an **if...else** statement in C programming language is −
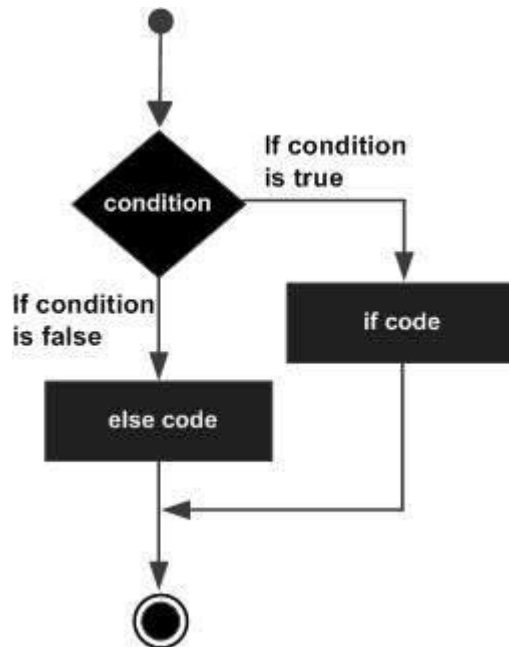
```
if(boolean_expression) {

    /* Statement(s) will execute if the boolean expression is true */
}
```

```
else {

   /* Statement(s) will execute if the boolean expression is false */
}
```

The above syntax can be represented in the form of a flow diagram as shown below −

An **if...else** statement is useful when we have to take a decision out of two options. For example, if a student secures more marks than 95, then the student is brilliant, otherwise no such situation can be coded, as follows −

```
#include <stdio.h>

main() {

   int x = 45;

   if( x > 95) {

      printf( "Student is brilliant\n");
   }else {
      printf( "Student is not brilliant\n");
   }
}
```

When the above program is executed, it produces the following result −

```
Student is not brilliant
```
if...elseif...else statement

An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions.

While using **if, else if, else** statements, there are a few points to keep in mind −

- An **if** can have zero or one **else's** and it must come after an **else if**.
- An **if** can have zero to many **else…if's** and they must come before the **else**.
- Once an **else…if** succeeds, none of the remaining **else…if's** or **else's** will be tested.

The syntax of an **if...else if...else** statement in C programming language is −

```
if(boolean_expression 1) {

    /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2) {

    /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3) {

    /* Executes when the boolean expression 3 is true */
}
else {

    /* Executes when the none of the above condition is true */
}
```

Now with the help of **if...elseif...else** statement, the very first program can be coded as follows −

```
#include <stdio.h>

main() {

    int x = 45;

    if( x > 95) {
        printf( "Student is brilliant\n");
    }else if( x < 30) {
        printf( "Student is poor\n");
    }else if( x < 95 && x > 30 ) {
        printf( "Student is average\n");
    }
}
```

When the above program is executed, it produces the following result −

```
Student is average
```

The Switch Statement

A **switch** statement is an alternative of **if statements** which allows a variable to be tested for equality against a list of values. Each value is called a **case**, and the variable being switched on is checked for each switch case. It has the following syntax −

```
switch(expression){
    case ONE  :
        statement(s);
        break;
    case TWO:
        statement(s);
```
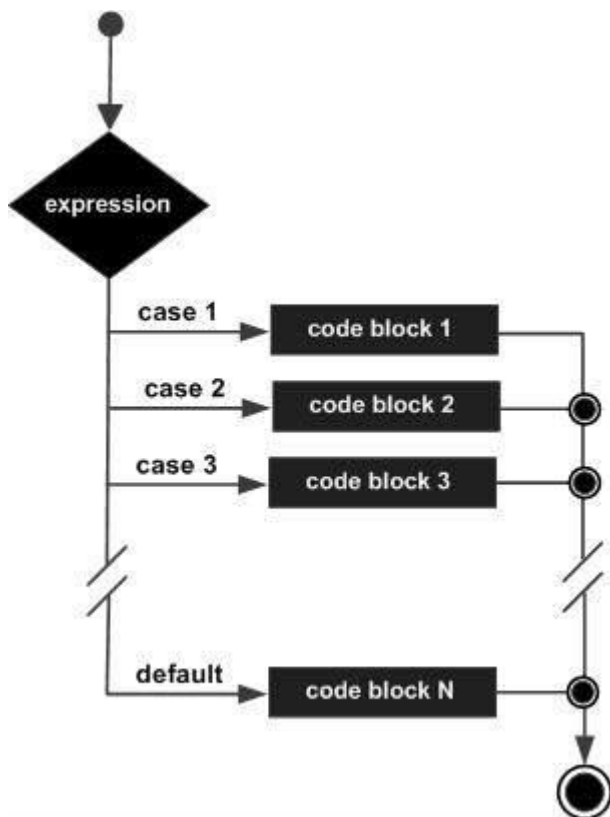
```
      break;
   ......

   default :
      statement(s);
}
```

The **expression** used in a **switch** statement must give an integer value, which will be compared for equality with different cases given. Wherever an expression value matches with a case value, the body of that case will be executed and finally, the switch will be terminated using a **break** statement. If no break statements are provided, then the computer continues executing other statements available below to the matched case. If none of the cases matches, then the default case body is executed.

The above syntax can be represented in the form of a flow diagram as shown below −

Now, let's consider another example where we want to write the equivalent English word for a given number. Then, it can be coded as follows −

```
#include <stdio.h>

main() {

   int x = 2;

   switch( x ){
      case 1 :
         printf( "One\n");
         break;
      case 2 :
```

```
         printf( "Two\n");
         break;
      case 3 :
         printf( "Three\n");
         break;
      case 4 :
         printf( "Four\n");
         break;
      default :
         printf( "None of the above...\n");
   }
}
```

When the above program is executed, it produces the following result −

```
Two
```

Decisions in Java

Following is the equivalent program written in Java which too supports **if**, **if...else**, **if...elseif...else**, and **switch** statements.

You can try to execute the following program to see the output, which must be identical to the result generated by the above C example.

```
public class DemoJava {

   public static void main(String []args) {

      int x = 45;

      if( x > 95) {
         System.out.println( "Student is brilliant");
      }else if( x < 30) {
         System.out.println( "Student is poor");
      }else if( x < 95 && x > 30 ) {
         System.out.println( "Student is average");
      }

   }
}
```

Decisions in Python

Following is the equivalent program written in Python. Python provides **if**, **if...else**, **if...elif...else**, and **switch** statements. Here, you must note that Python does not make use of curly braces for conditional body, instead it simply identifies the body of the block using indentation of the statements.

You can try to execute the following program to see the output −

```
x = 45

if x > 95:
   print "Student is brilliant"
elif x < 30:
   print "Student is poor"
elif x < 95 and x > 30:
```

```
    print "Student is average"

print "The end"
```

When the above program is executed, it produces the following result −

```
Student is average
The end
```

Computer Programming - Loops

Let's consider a situation when you want to print **Hello, World!** five times. Here is a simple C program to do the same −

```
#include <stdio.h>

main() {

    printf( "Hello, World!\n");
    printf( "Hello, World!\n");
    printf( "Hello, World!\n");
    printf( "Hello, World!\n");
    printf( "Hello, World!\n");

}
```

When the above program is executed, it produces the following result −

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

It was simple, but again, let's consider another situation when you want to write **Hello, World!** a thousand times. We can certainly not write printf() statements a thousand times. Almost all the programming languages provide a concept called **loop**, which helps in executing one or more statements up to a desired number of times. All high-level programming languages provide various forms of loops, which can be used to execute one or more statements repeatedly.

Let's write the above C program with the help of a **while loop** and later, we will discuss how this loop works

```
#include <stdio.h>

main() {

    int i = 0;

    while ( i < 5 ) {

        printf( "Hello, World!\n");
```

```
      i = i + 1;
   }
}
```

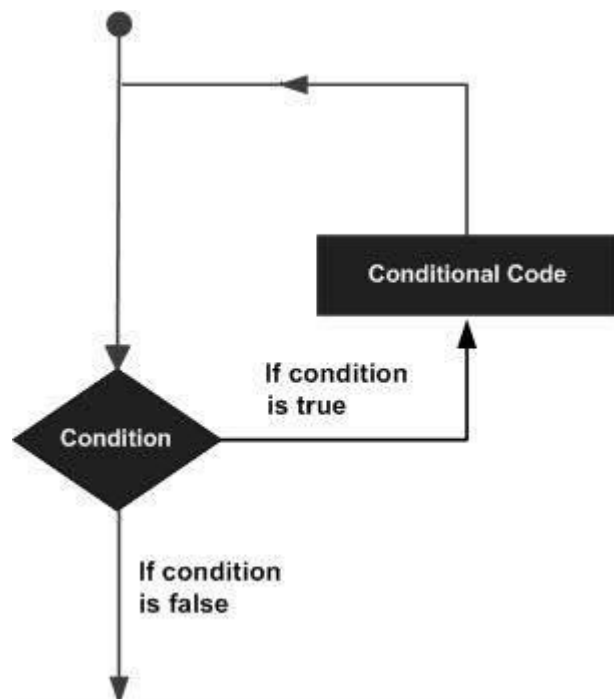When the above program is executed, it produces the following result −

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

The above program makes use of a **while loop**, which is being used to execute a set of programming statements enclosed within {....}. Here, the computer first checks whether the given condition, i.e., variable "a" is less than 5 or not and if it finds the condition is true, then the loop body is entered to execute the given statements. Here, we have the following two statements in the loop body −

- First statement is *printf()* function, which prints Hello World!
- Second statement is *i = i + 1*, which is used to increase the value of variable **i**

After executing all the statements given in the loop body, the computer goes back to while(i < 5) and the given condition, (i < 5), is checked again, and the loop is executed again if the condition holds true. This process repeats till the given condition remains true which means variable "a" has a value less than 5.

To conclude, a loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages −
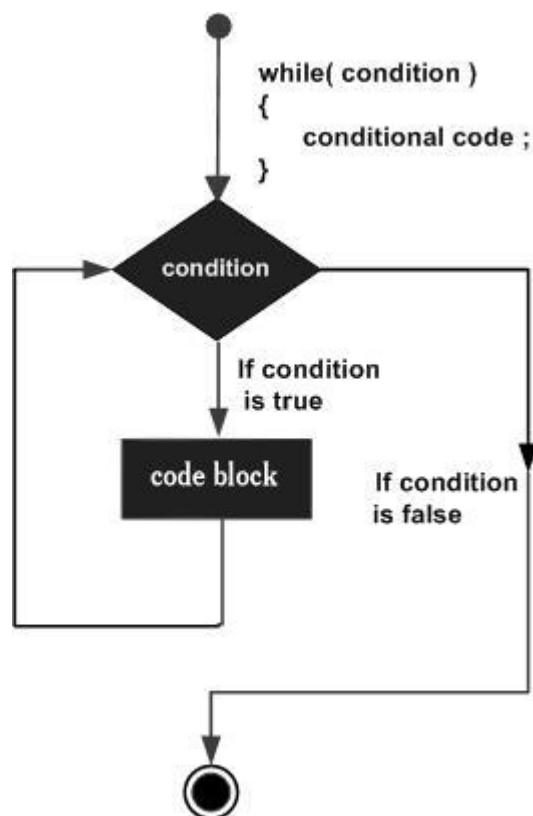
This tutorial has been designed to present programming's basic concepts to non-programmers, so let's discuss the two most important loops available in C programming language. Once you are clear about these two loops, then you can pick-up C programming tutorial or a reference book and check other loops available in C and the way they work.

The while Loop

A **while loop** available in C Programming language has the following syntax −

```
while ( condition ) {

   /*....while loop body ....*/
}
```

The above code can be represented in the form of a flow diagram as shown below −

The following important points are to be noted about a while loop −

- A while loop starts with a keyword **while** followed by a **condition** enclosed in ( ).
- Further to the while() statement, you will have the body of the loop enclosed in curly braces **{...}**.
- A while loop body can have one or more lines of source code to be executed repeatedly.
- If the body of a while loop has just one line, then its optional to use curly braces **{...}**.
- A while loop keeps executing its body till a given **condition** holds true. As soon as the condition becomes false, the while loop comes out and continues executing from the immediate next statement after the while loop body.
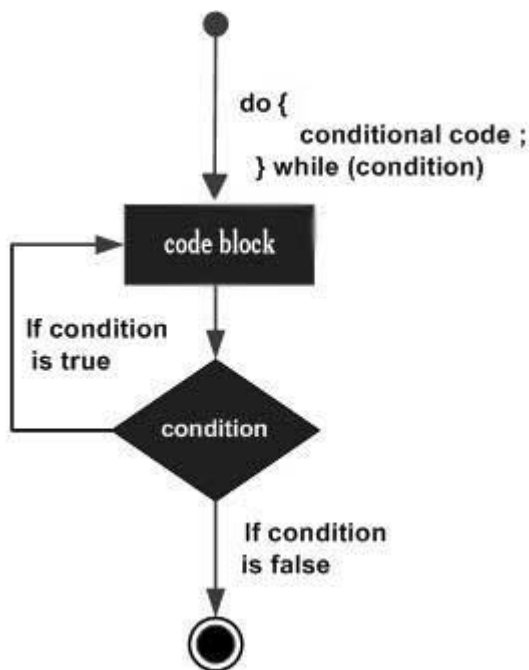
- A condition is usually a relational statement, which is evaluated to either true or false. A value equal to zero is treated as false and any non-zero value works like true.

The do...while Loop

A while loop checks a given condition before it executes any statements given in the body part. C programming provides another form of loop, called **do...while** that allows to execute a loop body before checking a given condition. It has the following syntax −

```
do {

   /*....do...while loop body ....*/
} while ( condition );
```

The above code can be represented in the form of a flow diagram as shown below −

If you will write the above example using **do...while** loop, then **Hello, World** will produce the same result −

```
#include <stdio.h>

main() {

   int i = 0;

   do {

      printf( "Hello, World!\n");
      i = i + 1;
   }while ( i < 5 );
}
```

When the above program is executed, it produces the following result −
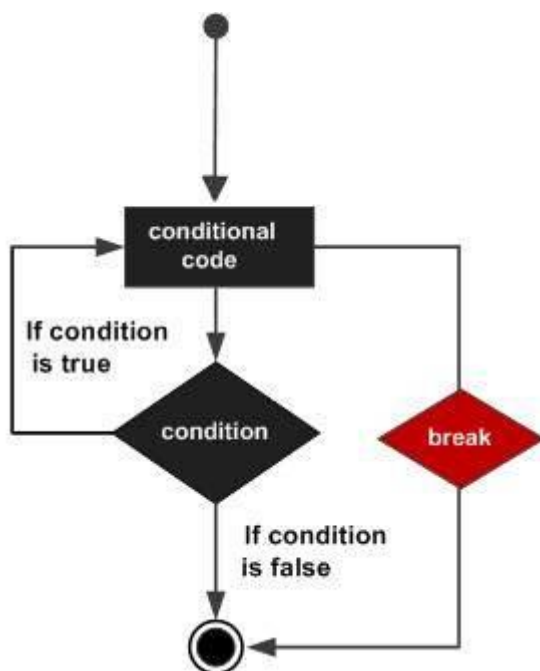
```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```
The break statement

When the **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop. The syntax for a **break** statement in C is as follows −

```
break;
```

A **break** statement can be represented in the form of a flow diagram as shown below −

Following is a variant of the above program, but it will come out after printing Hello World! only three times −

```
#include <stdio.h>

main() {

   int i = 0;

   do {

      printf( "Hello, World!\n");
      i = i + 1;
      if( i == 3 ) {

         break;
      }
   }while ( i < 5 );
}
```

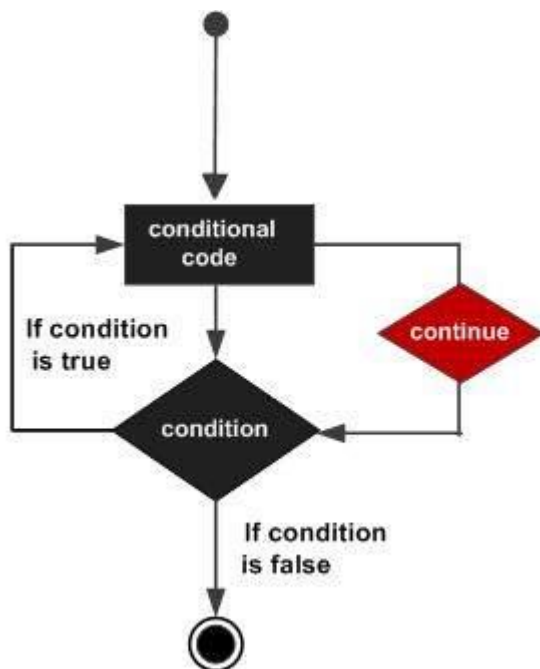When the above program is executed, it produces the following result −

```
Hello, World!
Hello, World!
Hello, World!
```

The continue statement

The **continue** statement in C programming language works somewhat like the **break** statement. Instead of forcing termination, **continue** forces the next iteration of the loop to take place, skipping any code in between. The syntax for a **continue** statement in C is as follows −

```
continue;
```

A **continue** statement can be represented in the form of a flow diagram as shown below −

Following is a variant of the above program, but it will skip printing when the variable has a value equal to 3 −

```
#include <stdio.h>

main() {

   int i = 0;

   do {

      if( i == 3 ) {

         i = i + 1;
         continue;
      }
      printf( "Hello, World!\n");
      i = i + 1;
```

```
    }while ( i < 5 );
}
```

When the above program is executed, it produces the following result −

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

Loops in Java

Following is the equivalent program written in Java that too supports **while** and **do...while** loops. The following program prints **Hello, World!** five times as we did in the case of C Programming −

You can try to execute the following program to see the output, which must be identical to the result generated by the above example.

```
public class DemoJava {

   public static void main(String []args) {

      int i = 0;

      while ( i < 5 ) {

         System.out.println("Hello, World!");
         i = i + 1;
      }

   }
}
```

The **break** and **continue** statements in Java programming work quite the same way as they work in C programming.

Loops in Python

Following is the equivalent program written in Python. Python too supports **while** and **do...while** loops. The following program prints **Hello, World!** five times as we did in case of C Programming. Here you must note that Python does not make use of curly braces for the loop body, instead it simply identifies the body of the loop using indentation of the statements.

You can try to execute the following program to see the output. To show the difference, we have used one more print statement, which will be executed when the loop will be over.

```
i = 0

while (i < 5):
   print "Hello, World!"
   i = i + 1
print "Loop ends"
```

When the above program is executed, it produces the following result −

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Loop ends
```

The **break** and **continue** statements in Python work quite the same way as they do in C programming.

Computer Programming - Numbers

Every programming language provides support for manipulating different types of numbers such as simple whole integers and floating point numbers. C, Java, and Python categorize these numbers in several categories based on their nature.

Let's go back and check the data types chapter, where we listed down the core data types related to numbers −

Type	Keyword	Value range which can be represented by this data type
Number	int	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
Small Number	short	-32,768 to 32,767
Long Number	long	-2,147,483,648 to 2,147,483,647
Decimal Number	float	1.2E-38 to 3.4E+38 till 6 decimal places

These data types are called primitive data types and you can use these data types to build more data types, which are called user-defined data types.

We have seen various mathematical and logical operations on numbers during a discussion on operators. So we know how to add numbers, subtract numbers, divide numbers, etc.

First let's see how to print various types of numbers available in C programming language −

```c
#include <stdio.h>

main() {

   short  s;
   int    i;
   long   l;
   float  f;
   double d;

   s = 10;
```

```
   i = 1000;
   l = 1000000;
   f = 230.47;
   d = 30949.374;

   printf( "s: %d\n", s);
   printf( "i: %d\n", i);
   printf( "l: %ld\n", l);
   printf( "f: %.3f\n", f);
   printf( "d: %.3f\n", d);
}
```

Rest of the coding is very obvious, but we used **%.3f** to print float and double, which indicates the number of digits after the decimal to be printed. When the above program is executed, it produces the following result −

```
s: 10
i: 1000
l: 1000000
f: 230.470
d: 30949.374
```

Math Operations on Numbers

The following table lists down various useful built-in mathematical **functions** available in C programming language which can be used for various important mathematical calculations.

For example, if you want to calculate the square root of a number, for example, 2304, then you have a built-in function available to calculate the square root.

S.N.	Function & Purpose
1	**double cos(double);** This function takes an angle (as a double) and returns the cosine.
2	**double sin(double);** This function takes an angle (as a double) and returns the sine.
3	**double tan(double);** This function takes an angle (as a double) and returns the tangent.
4	**double log(double);** This function takes a number and returns the natural log of that number.
5	**double pow(double, double);** The first is a number you wish to raise and the second is the power you wish to raise it to.
6	**double hypot(double, double);** If you pass this function the length of two sides of a right triangle, it will return the length of the hypotenuse.

7 **double sqrt(double);**

You pass this function a number and it returns its square root.

8 **int abs(int);**

This function returns the absolute value of an integer that is passed to it.

9 **double fabs(double);**

This function returns the absolute value of any decimal number passed to it.

10 **double floor(double);**

Finds the integer which is less than or equal to the argument passed to it.

Following is a simple example to show a few mathematical operations. To utilize these functions, you need to include the math header file **<math.h>** in your program in the same way you included **stdio.h** −

```c
#include <stdio.h>
#include <math.h>

main() {

   short   s;
   int     i;
   long    l;
   float   f;
   double d;

   s = 10;
   i = 1000;
   l = 1000000;
   f = 230.47;
   d = 2.374;

   printf( "sin(s): %f\n", sin(s));
   printf( "abs(i): %f\n", abs(i));
   printf( "floor(f): %f\n", floor(f));
   printf( "sqrt(f): %f\n", sqrt(f));
   printf( "pow(d, 2): %f\n", pow(d, 2));

}
```

When the above program is executed, it produces the following result −

```
sin(s): -0.544021
abs(i): -0.544021
floor(f): 230.000000
sqrt(f): 15.181238
pow(d, 2): 5.635876
```

Besides the above usage, you will use numbers in loop counting, flag representation, true or false values in C programming.

Numbers in Java

Following is the equivalent program written in Java. Java provides almost all the numeric data types available in C programming.

You can try to execute the following program to see the output, which is identical to the result generated by the above C example.

```
public class DemoJava {

   public static void main(String []args) {

      short  s;
      int    i;
      long   l;
      float  f;
      double d;

      s = 10;
      i = 1000;
      l = 1000000L;
      f = 230.47f;
      d = 30949.374;

      System.out.format( "s: %d\n", s);
      System.out.format( "i: %d\n", i);
      System.out.format( "l: %d\n", l);
      System.out.format( "f: %f\n", f);
      System.out.format( "d: %f\n", d);

   }
}
```

When the above program is executed, it produces the following result −

```
s: 10
i: 1000
l: 1000000
f: 230.470001
d: 30949.374000
```

Java also provides a full range of built-in functions for mathematical calculation and you can use them in the same way as you did in C programming.

Numbers in Python

Python is a little different from C and Java; it categorizes numbers in **int**, **long**, **float** and **complex**. Here are some examples of numbers in Python −

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j

-786	0122L		-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L		-90.	-.6545+0J
-0x260	-052318172735L		-32.54e100	3e+26J
0x69	-4721885298529L		70.2-E12	4.53e-7j

Following is the equivalent program written in Python −

```
s = 10
i = 1000
l = 1000000
f = 230.47
d = 30949.374

print "s: ", s
print "i: ", i
print "l: ", l
print "f: ", f
print "d: ", d
```

When the above program is executed, it produces the following result −

```
s:   10
i:   1000
l:   1000000
f:   230.47
d:   30949.374
```

Python also provides a full range of built-in functions for mathematical calculations and you can use them in the same way you have used them in C programming.

Computer Programming - Characters

If it was easy to work with numbers in computer programming, it would be even easier to work with characters. Characters are simple alphabets like a, b, c, d...., A, B, C, D,....., but with an exception. In computer programming, any single digit number like 0, 1, 2,....and special characters like $, %, +, -.... etc., are also treated as characters and to assign them in a character type variable, you simply need to put them inside **single quotes**. For example, the following statement defines a character type variable **ch** and we assign a value 'a' to it −

```
char ch = 'a';
```

Here, **ch** is a variable of character type which can hold a character of the implementation's character set and **'a'** is called a **character literal** or a character constant. Not only a, b, c,.... but when any number like 1, 2, 3.... or any special character like !, @, #, #, $,.... is kept inside

single quotes, then they will be treated as a character literal and can be assigned to a variable of character type, so the following is a valid statement −

```
char ch = '1';
```

A character data type consumes 8 bits of memory which means you can store anything in a character whose ASCII value lies in between -127 to 127, so it can hold any of the 256 different values. A character data type can store any of the characters available on your keyboard including special characters like !, @, #, #, $, %, ^, &, *, (,), _, +, {, }, etc.

Note that you can keep only a single alphabet or a single digit number inside single quotes and more than one alphabets or digits are not allowed inside single quotes. So the following statements are invalid in C programming −

```
char ch1 = 'ab';
char ch2 = '10';
```

Given below is a simple example, which shows how to define, assign, and print characters in C Programming language −

```
#include <stdio.h>

main() {

   char  ch1;
   char  ch2;
   char  ch3;
   char  ch4;

   ch1 = 'a';
   ch2 = '1';
   ch3 = '$';
   ch4 = '+';

   printf( "ch1: %c\n", ch1);
   printf( "ch2: %c\n", ch2);
   printf( "ch3: %c\n", ch3);
   printf( "ch4: %c\n", ch4);
}
```

Here, we used %c to print a character data type. When the above program is executed, it produces the following result −

```
ch1: a
ch2: 1
ch3: $
ch4: +
```

Escape Sequences

Many programming languages support a concept called **Escape Sequence**. When a character is preceded by a backslash (\), it is called an escape sequence and it has a special meaning to the compiler. For example, \n in the following statement is a valid character and it is called a new line character −

```
char ch = '\n';
```

Here, character **n** has been preceded by a backslash (\), it has special meaning which is a new line but keep in mind that backslash (\) has special meaning with a few characters only. The following statement will not convey any meaning in C programming and it will be assumed as an invalid statement −

```
char ch = '\1';
```

The following table lists the escape sequences available in C programming language −

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point.
\f	Inserts a form feed in the text at this point.
\'	Inserts a single quote character in the text at this point.
\"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

The following example shows how the compiler interprets an escape sequence in a print statement −

```
#include <stdio.h>

main() {

    char  ch1;
    char  ch2;
    char  ch3;
    char  ch4;

    ch1 = '\t';
    ch2 = '\n';


    printf( "Test for tabspace %c and a newline %c will start here", ch1,
ch2);
}
```

When the above program is executed, it produces the following result −

```
Test for tabspace     and a newline
 will start here
```

Characters in Java

Following is the equivalent program written in Java. Java handles character data types much in the same way as we have seen in C programming. However, Java provides additional support for character manipulation.

You can try to execute the following program to see the output, which must be identical to the result generated by the above C example.

```
public class DemoJava {

   public static void main(String []args) {

      char  ch1;
      char  ch2;
      char  ch3;
      char  ch4;

      ch1 = 'a';
      ch2 = '1';
      ch3 = '$';
      ch4 = '+';

      System.out.format( "ch1: %c\n", ch1);
      System.out.format( "ch2: %c\n", ch2);
      System.out.format( "ch3: %c\n", ch3);
      System.out.format( "ch4: %c\n", ch4);

   }
}
```

When the above program is executed, it produces the following result −

```
ch1:  a
ch2:  1
ch3:  $
ch4:  +
```

Java also supports escape sequence in the same way you have used them in C programming.

Characters in Python

Python does not support any character data type but all the characters are treated as string, which is a sequence of characters. We will study strings in a separate chapter. You do not need to have any special arrangement while using a single character in Python.

Following is the equivalent program written in Python −

```
ch1 = 'a';
ch2 = '1';
ch3 = '$';
ch4 = '+';

print "ch1: ", ch1
print "ch2: ", ch2
```

```
print "ch3: ", ch3
print "ch4: ", ch4
```

When the above program is executed, it produces the following result −

```
ch1:  a
ch2:  1
ch3:  $
ch4:  +
```

Python supports escape sequences in the same way as you have used them in C programming.

Computer Programming - Arrays

Consider a situation where we need to store five integer numbers. If we use programming's simple variable and data type concepts, then we need five variables of **int** data type and the program will be as follows −

```
#include <stdio.h>

main() {

   int   number1;
   int   number2;
   int   number3;
   int   number4;
   int   number5;

   number1 = 10;
   number2 = 20;
   number3 = 30;
   number4 = 40;
   number5 = 50;

   printf( "number1: %d\n", number1);
   printf( "number2: %d\n", number2);
   printf( "number3: %d\n", number3);
   printf( "number4: %d\n", number4);
   printf( "number5: %d\n", number5);
}
```
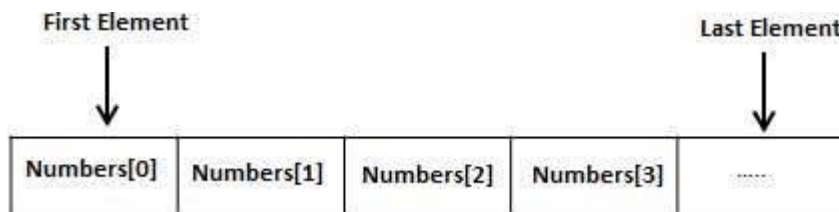
It was simple, because we had to store just five integer numbers. Now let's assume we have to store 5000 integer numbers. Are we going to use 5000 variables?

To handle such situations, almost all the programming languages provide a concept called **array**. An array is a data structure, which can store a fixed-size collection of elements of the same data type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number1, number2, ..., number99, you just declare one array variable **number** of integer type and use number1[0], number1[1], and ..., number1[99] to represent individual variables. Here, 0, 1, 2, ...99 are **index** associated with **var** variable and they are being used to represent individual elements available in the array.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Create Arrays

To create an array variable in C, a programmer specifies the type of the elements and the number of elements to be stored in that array. Given below is a simple syntax to create an array in C programming −

```
type arrayName [ arraySize ];
```

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, now to declare a 10-element array called **number** of type **int**, use this statement −

```
int number[10];
```

Here, *number* is a variable array, which is sufficient to hold up to 10 integer numbers.

Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows −

```
int number[5] = {10, 20, 30, 40, 50};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

```
int number[] = {10, 20, 30, 40, 50};
```

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array −

```
number[4] = 50;
```

The above statement assigns element number 5th in the array with a value of 50. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be the total size of the array minus 1. The following image shows the pictorial representation of the array we discussed above −

Index	0	1	2	3	4
number	10	20	30	40	50

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

```
int var = number[9];
```

The above statement will take the 10th element from the array and assign the value to **var** variable. The following example uses all the above-mentioned three concepts viz. creation, assignment, and accessing arrays −

```c
#include <stdio.h>

int main () {

   int number[10]; /* number is an array of 10 integers */
   int i = 0;

   /* Initialize elements of array n to 0 */
   while( i < 10 ) {

      /* Set element at location i to i + 100 */
      number[ i ] = i + 100;
      i = i + 1;
   }

   /* Output each array element's value */
   i = 0;
   while( i < 10 ) {

      printf("number[%d] = %d\n", i, number[i] );
      i = i + 1;
   }

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
number[0] = 100
number[1] = 101
number[2] = 102
number[3] = 103
number[4] = 104
number[5] = 105
number[6] = 106
number[7] = 107
number[8] = 108
number[9] = 109
```

Arrays in Java

Following is the equivalent program written in Java. Java supports arrays, but there is a little difference in the way they are created in Java using the **new** operator.

You can try to execute the following program to see the output, which must be identical to the result generated by the above C example.

```
public class DemoJava {

   public static void main(String []args) {

      int[] number = new int[10];
      int i = 0;

      while( i < 10 ) {

         number[ i ] = i + 100;
         i = i + 1;
      }

      i = 0;
      while( i < 10 ) {

         System.out.format( "number[%d] = %d\n", i, number[i] );
         i = i + 1;
      }
   }
}
```

When the above program is executed, it produces the following result −

```
number[0] = 100
number[1] = 101
number[2] = 102
number[3] = 103
number[4] = 104
number[5] = 105
number[6] = 106
number[7] = 107
number[8] = 108
number[9] = 109
```

Arrays (Lists) in Python

Python does not have a concept of Array, instead Python provides another data structure called **list**, which provides similar functionality as arrays in any other language.

Following is the equivalent program written in Python −

```
# Following defines an empty list.
number = []
i = 0

while i < 10:
   # Appending elements in the list
   number.append(i + 100)
```

```
   i = i + 1

i = 0
while i < 10:
   # Accessing elements from the list
   print "number[", i,  "] = ", number[ i ]
   i = i + 1
```

When the above program is executed, it produces the following result −

```
number[ 0 ] =  100
number[ 1 ] =  101
number[ 2 ] =  102
number[ 3 ] =  103
number[ 4 ] =  104
number[ 5 ] =  105
number[ 6 ] =  106
number[ 7 ] =  107
number[ 8 ] =  108
number[ 9 ] =  109
```

Computer Programming - Strings

During our discussion about **characters**, we learnt that character data type deals with a single character and you can assign any character from your keyboard to a character type variable.

Now, let's move a little bit ahead and consider a situation where we need to store more than one character in a variable. We have seen that C programming does not allow to store more than one character in a character type variable. So the following statements are invalid in C programming and produce syntax errors −

```
char ch1 = 'ab';
char ch2 = '10';
```

We have also seen how to use the concept of **arrays** to store more than one value of similar data type in a variable. Here is the syntax to store and print five numbers in an array of int type −

```
#include <stdio.h>

main() {

   int number[5] = {10, 20, 30, 40, 50};
   int i = 0;

   while( i < 5 ) {
      printf("number[%d] = %d\n", i, number[i] );
      i = i + 1;
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
number[0] = 10
```

```
number[1] = 20
number[2] = 30
number[3] = 40
number[4] = 50
```

Now, let's define an array of five characters in the same way as we did for numbers and try to print them −

```c
#include <stdio.h>

main() {

   char ch[5] = {'H', 'e', 'l', 'l', 'o'};
   int i = 0;

   while( i < 5 ) {
      printf("ch[%d] = %c\n", i, ch[i] );
      i = i + 1;
   }
}
```

Here, we used %c to print character value. When the above code is compiled and executed, it produces the following result −

```
ch[0] = H
ch[1] = e
ch[2] = l
ch[3] = l
ch[4] = o
```

If you are done with the above example, then I think you understood how strings work in C programming, because **strings in C are represented as arrays of characters**. C programming simplified the assignment and printing of strings. Let's check the same example once again with a simplified syntax −

```c
#include <stdio.h>

main() {

   char ch[5] = "Hello";
   int i = 0;

   /* Print as a complete string */
   printf("String = %s\n", ch);

   /* Print character by character */
   while( i < 5 ) {
      printf("ch[%d] = %c\n", i, ch[i] );
      i = i + 1;
   }
}
```

Here, we used %s to print the full string value using array name **ch**, which is actually the beginning of the memory address holding **ch** variable as shown below −

Although it's not visible from the above examples, a C program internally assigns null character **'\0'** as the last character of every string. It indicates the end of the string and it means if you want to store a 5 character string in an array, then you must define an array size of 6 as a good practice, though C does not complain about it.

If the above code is compiled and executed, it produces the following result −

```
String = Hello
ch[0] = H
ch[1] = e
ch[2] = l
ch[3] = l
ch[4] = o
```

Basic String Concepts

Based on the above discussion, we can conclude the following important points about strings in C programming language −

- Strings in C are represented as arrays of characters.
- We can constitute a string in C programming by assigning character by character into an array of characters.
- We can constitute a string in C programming by assigning a complete string enclosed in double quote.
- We can print a string character by character using an array subscript or a complete string by using an array name without subscript.
- The last character of every string is a null character, i.e., '\0'.
- Most of the programming languages provide built-in functions to manipulate strings, i.e., you can concatenate strings, you can search from a string, you can extract sub-strings from a string, etc. For more, you can check our detailed tutorial on C programming or any other programming language.

Strings in Java

Though you can use character arrays to store strings, but Java is an advanced programming language and its designers tried to provide additional functionality. Java provides strings as a built-in data type like any other data type. It means you can define strings directly instead of defining them as array of characters.

Following is the equivalent program written in Java. Java makes use of the **new** operator to create string variables as shown in the following program.

You can try to execute the following program to see the output −

```
public class DemoJava {

   public static void main(String []args) {

      String str = new String("Hello");

      System.out.println( "String = " + str );
   }
}
```

When the above program is executed, it produces the following result −

```
String = Hello
```

Strings in Python

Creating strings in Python is as simple as assigning a string into a Python variable using single or double quotes.

Given below is a simple program that creates two strings and prints them using print() function −

```
var1 = 'Hello World!'
var2 = "Python Programming"

print "var1 = ", var1
print "var2 = ", var2
```

When the above program is executed, it produces the following result −

```
var1 =  Hello World!
var2 =  Python Programming
```

Python does not support character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. Take a look at the following code segment −

```
var1 = 'Hello World!'
var2 = "Python Programming"

print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

When the above code is executed, it produces the following result −

```
var1[0]:  H
var2[1:5]:  ytho
```

Computer Programming - Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. You have already seen various functions like **printf()** and **main()**. These are called built-in functions provided by the language itself, but we can write our own functions as well and this tutorial will teach you how to write and use those functions in C programming language.

Good thing about functions is that they are famous with several names. Different programming languages name them differently, for example, functions, methods, sub-routines, procedures, etc. If you come across any such terminology, then just imagine about the same concept, which we are going to discuss in this tutorial.

Let's start with a program where we will define two arrays of numbers and then from each array, we will find the biggest number. Given below are the steps to find out the maximum number from a given set of numbers −

```
1. Get a list of numbers L₁, L₂, L₃....Lₙ
2. Assume L₁ is the largest, Set max = L₁
3. Take next number Lᵢ from the list and do the following
4.    If max is less than Lᵢ
5.        Set max = Lᵢ
6.    If Lᵢ is last number from the list then
7.        Print value stored in max and come out
8. Else prepeat same process starting from step 3
```

Let's translate the above program in C programming language −

```c
#include <stdio.h>

main() {

   int set1[5] = {10, 20, 30, 40, 50};
   int set2[5] = {101, 201, 301, 401, 501};
   int i, max;

   /* Process first set of numbers available in set1[] */
   max = set1[0];
   i = 1;
   while( i < 5 ) {

      if( max <  set1[i] ) {
         max = set1[i];
      }
      i = i + 1;
   }

   printf("Max in first set = %d\n", max );

   /* Now process second set of numbers available in set2[] */
   max = set2[0];
   i = 1;
   while( i < 5 ) {
      if( max <  set2[i] ) {
```

```
         max = set2[i];
      }
      i = i + 1;
   }
   printf("Max in second set = %d\n", max );
}
```

When the above code is compiled and executed, it produces the following result −

```
Max in first set = 50
Max in second set = 501
```

If you are clear about the above example, then it will become easy to understand why we need a function. In the above example, there are only two sets of numbers, set1 and set2, but consider a situation where we have 10 or more similar sets of numbers to find out the maximum numbers from each set. In such a situation, we will have to repeat, processing 10 or more times and ultimately, the program will become too large with repeated code. To handle such situation, we write our functions where we try to keep the source code which will be used again and again in our programming.

Now, let's see how to define a function in C programming language and then in the subsequent sections, we will explain how to use them.

Defining a Function

The general form of a function definition in C programming language is as follows −

```
return_type function_name( parameter list ) {

   body of the function

   return [expression];
}
```

A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function −

- **Return Type** − A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameter List** − A parameter is like a placeholder. When a function is invoked, you pass a value as a parameter. This value is referred to as the actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** − The function body contains a collection of statements that defines what the function does.

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform a defined task.

Now, let's write the above example with the help of a function −

```c
#include <stdio.h>

int getMax( int set[] ) {

   int i, max;

   max = set[0];
   i = 1;
   while( i < 5 ) {

      if( max <  set[i] ) {

         max = set[i];
      }
      i = i + 1;
   }

   return max;
}

main() {

   int set1[5] = {10, 20, 30, 40, 50};
   int set2[5] = {101, 201, 301, 401, 501};
   int max;

   /* Process first set of numbers available in set1[] */
   max = getMax(set1);
   printf("Max in first set = %d\n", max );

   /* Now process second set of numbers available in set2[] */
   max = getMax(set2);
   printf("Max in second set = %d\n", max );
}
```

When the above code is compiled and executed, it produces the following result −

```
Max in first set = 50
Max in second set = 501
```

Functions in Java

If you are clear about functions in C programming, then it is easy to understand them in Java as well. Java programming names them as **methods**, but the rest of the concepts remain more or less same.

Following is the equivalent program written in Java. You can try to execute it to see the output −

```java
public class DemoJava {
```

```
    public static void main(String []args) {

        int[] set1 = {10, 20, 30, 40, 50};
        int[] set2 = {101, 201, 301, 401, 501};
        int max;

        /* Process first set of numbers available in set1[] */
        max = getMax(set1);
        System.out.format("Max in first set = %d\n", max );

        /* Now process second set of numbers available in set2[] */
        max = getMax(set2);
        System.out.format("Max in second set = %d\n", max );
    }

    public static int getMax( int set[] ) {

        int i, max;

        max = set[0];
        i = 1;
        while( i < 5 ) {

            if( max <  set[i] ) {
                max = set[i];
            }
            i = i + 1;
        }

        return max;
    }
}
```

When the above program is executed, it produces the following result −

```
Max in first set = 50
Max in second set = 501
```

Functions in Python

Once again, if you know the concept of functions in C and Java programming, then Python is not much different. Given below is the basic syntax of defining a function in Python −

```
def function_name( parameter list ):
   body of the function

   return [expression]
```

Using this syntax of function in Python, the above example can be written as follows −

```
def getMax( set ):
   max = set[0]
   i = 1
   while( i < 5 ):
      if( max <  set[i] ):
         max = set[i]
      i = i + 1
   return max
```

```
set1 = [10, 20, 30, 40, 50]
set2 = [101, 201, 301, 401, 501]

# Process first set of numbers available in set1[]
max = getMax(set1)
print "Max in first set = ", max

# Now process second set of numbers available in set2[]
max = getMax(set2)
print "Max in second set = ", max
```

When the above code is executed, it produces the following result −

```
Max in first set =   50
Max in second set =   501
```

Computer Programming - File I/O

Computer Files

A computer file is used to store data in digital format like plain text, image data, or any other content. Computer files can be organized inside different directories. Files are used to keep digital data, whereas directories are used to keep files.

Computer files can be considered as the digital counterpart of paper documents. While programming, you keep your source code in text files with different extensions, for example, C programming files end with the extension **.c**, Java programming files with **.java**, and Python files with **.py**.

File Input/Output

Usually, you create files using text editors such as notepad, MS Word, MS Excel or MS Powerpoint, etc. However, many times, we need to create files using computer programs as well. We can modify an existing file using a computer program.

File input means data that is written into a file and file output means data that is read from a file. Actually, input and output terms are more related to screen input and output. When we display a result on the screen, it is called output. Similarly, if we provide some input to our program from the command prompt, then it is called input.

For now, it is enough to remember that writing into a file is file input and reading something from a file is file output.

File Operation Modes

Before we start working with any file using a computer program, either we need to create a new file if it does not exist or open an already existing file. In either case, we can open a file in the following modes −

- **Read-Only Mode** − If you are going to just read an existing file and you do not want to write any further content in the file, then you will open the file in read-only mode. Almost all the programming languages provide syntax to open files in read-only mode.
- **Write-Only Mode** − If you are going to write into either an existing file or a newly created file but you do not want to read any written content from that file, then you will open the file in write-only mode. All the programming languages provide syntax to open files in write-only mode.
- **Read & Write Mode** − If you are going to read as well as write into the same file, then you will open file in read & write mode.
- **Append Mode** − When you open a file for writing, it allows you to start writing from the beginning of the file; however it will overwrite existing content, if any. Suppose we don't want to overwrite any existing content, then we open the file in append mode. Append mode is ultimately a write mode, which allows content to be appended at the end of the file. Almost all the programming languages provide syntax to open files in append mode.

In the following sections, we will learn how to open a fresh new file, how to write into it, and later, how to read and append more content into the same file.

Opening Files

You can use the **fopen()** function to create a new file or to open an existing file. This call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream. Here is the prototype, i.e., signature of this function call −

```
FILE *fopen( const char * filename, const char * mode );
```

Here, **filename** is string literal, which you will use to name your file and access **mode** can have one of the following values −

Mode	Description
r	Opens an existing text file for reading purpose.
w	Opens a text file for writing. If it does not exist, then a new file is created. Here, your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here, your program will start appending content in the existing file content.
r+	Opens a text file for reading and writing both.
w+	Opens a text file for both reading and writing. It first truncates the file to zero length, if it exists; otherwise creates the file if it does not exist.
a+	Opens a text file for both reading and writing. It creates a file, if it does not exist. The reading will start from the beginning, but writing can only be appended.

Closing a File

To close a file, use the **fclose( )** function. The prototype of this function is −

```
int fclose( FILE *fp );
```

The **fclose( )** function returns zero on success, or **EOF**, special character, if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file **stdio.h**.

There are various functions provided by C standard library to read and write a file character by character or in the form of a fixed length string. Let us see a few of them in the next section.

Writing a File

Given below is the simplest function to write individual characters to a stream −

```
int fputc( int c, FILE *fp );
```

The function **fputc()** writes the character value of the argument c to the output stream referenced by **fp**. It returns the written character written on success, otherwise **EOF** if there is an error. You can use the following functions to write a null-terminated string to a stream −

```
int fputs( const char *s, FILE *fp );
```

The function **fputs()** writes the string **s** into the file referenced by fp. It returns a non-negative value on success, otherwise **EOF** is returned in case of any error. You can also use the function **int fprintf(FILE *fp,const char *format, ...)** to write a string into a file. Try the following example −

```
#include <stdio.h>

main() {

   FILE *fp;

   fp = fopen("/tmp/test.txt", "w+");
   fprintf(fp, "This is testing for fprintf...\n");
   fputs("This is testing for fputs...\n", fp);
   fclose(fp);
}
```

When the above code is compiled and executed, it creates a new file **test.txt** in **/tmp** directory and writes two lines using two different functions. Let us read this file in the next section.

Reading a File

Given below is the simplest function to read a text file character by character −

```
int fgetc( FILE * fp );
```

The **fgetc()** function reads a character from the input file referenced by **fp**. The return value is the character read; or in case of any error, it returns **EOF**. The following function allows you to read a string from a stream −

```
char *fgets( char *buf, int n, FILE *fp );
```

The function **fgets()** reads up to n - 1 characters from the input stream referenced by **fp**. It copies the read string into the buffer **buf**, appending a **null** character to terminate the string.

If this function encounters a newline character '\n' or EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including the new line character. You can also use **int fscanf(FILE *fp, const char *format, ...)** to read strings from a file, but it stops reading after encountering the first space character.

```
#include <stdio.h>

main() {

   FILE *fp;
   char buff[255];

   fp = fopen("/tmp/test.txt", "r");
   fscanf(fp, "%s", buff);
   printf("1 : %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("2: %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("3: %s\n", buff );
   fclose(fp);

}
```

When the above code is compiled and executed, it reads the file created in the previous section and produces the following result −

```
1 : This
2: is testing for fprintf...

3: This is testing for fputs...
```

Let's analyze what happened here. First, the **fscanf()** method reads **This** because after that, it encountered a space. The second call is for **fgets()**, which reads the remaining line till it encountered end of line. Finally, the last call **fgets()** reads the second line completely.

File I/O in Java

Java provides even richer set of functions to handle File I/O. For more on this topic, we suggest you to check our Java Tutorials.

Here, we will see a simple Java program, which is equivalent to the C program explained above. This program will open a text file, write a few text lines into it, and close the file.

Finally, the same file is opened and then read from an already created file. You can try to execute the following program to see the output −

```java
import java.io.*;

public class DemoJava {

   public static void main(String []args) throws IOException {

      File file = new File("/tmp/java.txt");

      // Create a File
      file.createNewFile();

      //  Creates a FileWriter Object using file object
      FileWriter writer = new FileWriter(file);

      // Writes the content to the file
      writer.write("This is testing for Java write...\n");
      writer.write("This is second line...\n");

      // Flush the memory and close the file
      writer.flush();
      writer.close();

      // Creates a FileReader Object
      FileReader reader = new FileReader(file);
      char [] a = new char[100];

      // Read file content in the array
      reader.read(a);
      System.out.println( a );

      // Close the file
      reader.close();
   }
}
```

When the above program is executed, it produces the following result −

```
This is testing for Java write...
This is second line...
```

File I/O in Python

The following program shows the same functionality to open a new file, write some content into it, and finally, read the same file −

```python
# Create a new file
fo = open("/tmp/python.txt", "w")

# Writes the content to the file
fo.write( "This is testing for Python write...\n");
fo.write( "This is second line...\n");

# Close the file
fo.close()

# Open existing file
```

```
fo = open("/tmp/python.txt", "r")

# Read file content in a variable
str = fo.read(100);
print str

# Close opened file
fo.close()
```

When the above code is executed, it produces the following result −

```
This is testing for Python write...
This is second line...
```

Computer Programming - Summary

We appreciate your patience for going through this tutorial. We have tried to keep it concise but as this subject contains several topics, we have shown a few examples in detail.

If you have not understood any of the concepts, then we recommend to go through the tutorial once again and once you are comfortable with the concepts explained in this tutorial, you can proceed further.

There are many other subjects related to computer programming which we did not cover intentionally to avoid any confusion, but we are sure those concepts will not be difficult for you to understand as long as you make yourself comfortable with the concepts explained in this tutorial.

At tutorialspoint, we have put lots of effort to prepare comprehensive tutorials on C, Java, and Python programming languages and we strongly recommend you to start either of them, after having completed this tutorial.

Kindly share with us your views about this tutorial, mentioning the tutorial name in the subject line at **webmaster@tutorialspoint.com**. If you have any suggestion to improve this tutorial further, then we would definitely like to hear from you.