Easy way to learn C Programming

Dr. X. Joshphin Jasaline Anitha

C-Preface

C- The high-level language that was originally developed by Dennis M. Ritchie is easy to learn and can be compiled on a variety of computer platforms. Most of the state-of-the-art software has been implemented using C. This book aims to make the learning of this universal computer language simple and interesting. The step by step methods that are given in the book is sure to make it reader friendly. Apart from being a ready reference, this book is written with a view to attract more and more students to learn, comprehend and subsequently use it for their projects and research.

I hope this volume will be a valuable reference for computer science & non IT students, researchers, computer programmers and software professionals.

Goals

- **Be clear, readable, and possibly even entertaining**. Many C books are too concise for the average reader. I've tried to give clear, thorough explanations to hold the reader's interest.

- **Be authoritative without being pedantic**. To avoid arbitrarily deciding what to include and what not to include, I've tried to cover all the features of the C language and library. At the same time, I've tried to avoid burdening the reader with unnecessary detail.

- **Be organized for easy learning**. My experience in teaching C underscores the importance of presenting the features of C gradually. I use a spiral approach, in which difficult topics are introduced briefly, then revisited one or more times later in the book with details added each time.

- **Motivate language features**. Instead of just describing each feature of the language and giving a few simple examples of how the feature is used, I've tried to motivate each feature and discuss how it's used in practical situations.

- **Emphasize style**. It's important for every C programmer to develop a consistent style. Rather than dictating what this style should be, though, I usually describe a few possibilities and let the reader choose the one that's most appealing. Knowing alternative styles is a big help when reading other people's programs (which programmers often spend a great deal of time doing).

CONTENTS

CHAPTER - 1

Introduction to C and C Fundamentals

Overview

- Origin Of C
- The Problems Of B
- Features Of C
- Characteristics Of C
- Current Uses Of C
- The C Compilation Model
- Basic Structure Of C
- The C Character Set
- Identifiers And Keywords
- Data Type
- Constants
- Variables
- Escape Sequences
- Expressions
- Statements

Origin of C

DENNIS RITCHIE

C was Invented and first implemented by Dennis Ritchie on a DEC PDP-11 that version used the unix operating system. The first version of Unix was written in the low-level PDP-7 assembler language. C is the result of a development process that started with an older language called B, which was invented by Ken Thompson.B leds to the development of C in 1970's. The C programming language was first described by Brian Kernighan and Dennis Ritchie. In 1983, a committee was established to create an ANSI(American National Standards institute) standard that would define the C language once and for all.The ANSI C standard was finally adopted in December 1989, with the first copies becoming available in early 1990. The standard was also adopted by ISO(International Standards Organisation) and now is refered as ANSI /ISO C standard.

C is a middle level language because it combines the best elements of high level languages with the control and flexibility of assembly language.

Highest level	---------- >	Ada
		Modula-2
		Pascal
		COBOL
		FORTRAN
		BASIC
Middle level	---------- >	Java
		C++
		C
		FORTH
Lowest level	---------- >	Macro-assembler
		Assembler

The Problems of B (BCPL - Basic Combined Programming Language)

The advent of the PDP-11 exposed several inadequacies of B's semantic model. First, its character-handling mechanisms, inherited with few changes from BCPL, were clumsy: using library procedures to spread packed strings into individual cells and then repack, or to access and replace individual characters, began to feel awkward, even silly, on a byte-oriented machine.

Second, although the original PDP-11 did not provide for floating-point Arithmetic. Floating-point operations had been added to BCPL but the mechanism was possible only because on the relevant machines, a single word was large enough to contain a floating-point number; this was not true on the 16-bit PDP-11.

Finally, the B and BCPL model implied overhead in dealing with pointers. Each pointer reference generated a run-time scale conversion from the pointer to the byte address expected by the hardware.

Aside from the problems with the language itself, the B compiler's threaded-code technique yielded programs so much slower than their assembly-language .

In 1971 , the B language was extended by adding a character type and also rewrote its compiler to generate PDP-11 machine instructions instead of threaded code. Thus the transition from B to C was contemporaneous with the creation of a compiler capable of producing programs fast and small enough to compete with assembly language. B called the slightly-extended language NB, for `new B.'

Features of C

- Modularity
- Portability
- Code-Reusability
- Ability to Extend Itself
- Limited Number of Keywords

Characteristics of C

We briefly list some of C's characteristics that define the language and also have lead to its popularity as a programming language. Naturally we will be studying many of these aspects throughout the course.

- Small size
- Extensive use of function calls
- Loose typing -- unlike PASCAL
- Structured language
- Low level (Bitwise) programming readily available
- Pointer implementation - extensive use of pointers for memory, array, structures and functions.

C has now become a widely used professional language for various reasons.

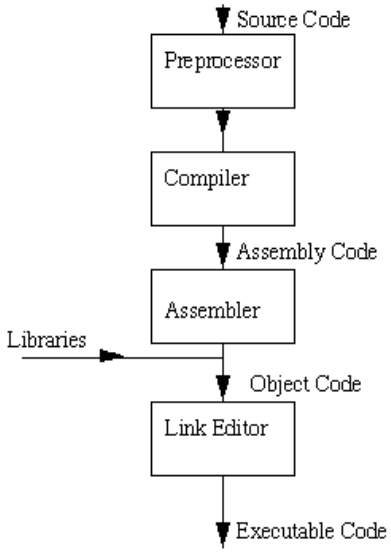
- It has high-level constructs.
- It can handle low-level activities.
- It produces efficient programs.
- It can be compiled on a variety of computers.

Its main drawback is that it has poor error detection which can make it off putting to the beginner. However diligence in this matter can pay off handsomely since having learned the rules of C we can break them. Not many languages allow this. This if done properly and carefully leads to the power of C programming.

Current uses of C

- Operating system
- Interpreters
- Editors
- Compilers
- File utilities
- Performances enhancers
- Real time executives

The C Compilation Model



The Preprocessor

The Preprocessor accepts source code as input and is responsible for

- removing comments
- interpreting special preprocessor directives denoted by #.

For example

- #include -- includes contents of a named file. Files usually called header files.
  (e.g) o #include <math.h> -- standard library maths file.
  o #include <stdio.h> -- standard library I/O file
- #define -- defines a symbolic name or constant. Macro substitution.
  o #define MAX_ARRAY_SIZE 100

C Compiler

The C compiler translates source to assembly code. The source code is received from the preprocessor.

Assembler

The assembler creates object code. On a UNIX system you may see files with a .o suffix (.OBJ on MSDOS) to indicate object code files.

Link Editor

If a source file references library functions or functions defined in other source files the link editor combines these functions (with main()) to create an executable file. External Variable references resolved here also.

BASIC STRUCTURE OF C

```
Documentation Section
Link section
Definition section
Global declaration
Return_type main(parameter list)
        {
                Statement sequence
        }
Subprogram section
Function 1
Function 2
```

Function 3
:
:
Function n

Documentation section - Consists of set of comment lines giving the name of the program, the author and other details regarding the program. Comment line should be given between /*…….*/.

Link section - provides instructions to the compiler to link functions from the system library.

Definition section - Defines all symbolic constants .

Global declaration section - Declares variables that are globally visible, inside all the functions

Main program section - All C program must have one main function. Consists of two parts, Declaration part and Executable part.

- The declaration part declares all variables used in the executable part.
- There is at least one statement in the executable part.
- These two parts appear between the opening and closing braces.
- The program execution begins at opening braces and end at closing braces. The closing brace of the main function section is at the logical end of the program.
- All statements in the declaration part and executable parts end with a semicolon.

Subprogram section - Contains all the user-defined functions that are called in the main function. User defined functions are placed immediately after the main function, although they may appear in any order.

Sample program

```
/* Print the given sentence*/
#include<stdio.h>
main()
{
printf("Welcome to JSC");
}
```

Output:

Welcome to JSC

The C Character Set

C uses the following as building blocks to form basic program elements such as constants, variables, operators, expressions etc.

- Uppercase letters A to Z.
- Lowercase Letters a to z
- Digits 0 to 9 and
- Certain Special Characters like :
  ! * + \ " < # (= | { >] ' . (blank) %) ` ; } / ^ - [: , ? & _

Special Characters

SYMBOL	NAME	SYMBOL	NAME
,	Comma	&	Ampersand
.	Period	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus Sign
?	Question Mark	+	Plus Sign
'	Aphostrophe	<	Opening Angle(Less than sign)
"	Quotation Marks	>	Closing Angle(Greater than Sign)
!	Exclaimation Mark	(	Left Parenthesis

|	Vertical Bar	)	Right Parenthesis
/	Slash	[	Left Bracket
\	Backslash	]	Right Bracket
~	Tilde	{	Left Brace
_	Underscore	}	Right Brace
$	Dollar Sign	#	Number Sign
%	Percentage Sign		

Identifiers And Keywords

Identifiers are names given to various program elements, such as variable, functions and array identifiers consist of letters and digits, in any order, expect that the first character must be a letter. The underscore character(_) can also be include. An identifier may also begin with an underscore.
The following names are examples of valid identifiers.

X	Y12	sum_1	_temprature
Name	area	tax_rate	TABLE

There are certain reserved words, called keywords, that have standard, predefined meanings in C. The keywords can be used only for intended purpose, they cannot be used as user defined keywords. Give bellow is a list of the standard keywords.

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

Data Types

There are five atomic data types in C: character, integer, floating-point,double floating-point and valueless(char,int,float,double, and void, respectively). The data type supported in a language dictates the type of values which can be processed by the language. C supports several different types of data, each of which may be represented differently within the computers memory.

The data types supported by C may broadly be classified into

- Simple, Primitive or atomic data.
- Compound, structured or derived data.

An atomic data is a fundamental unit of information which cannot be broken down to constituent parts. Whereas a derived data item is made up of one or more simple data items. The list given below classifies the items as simple compound.

SIMPLE	**Integers**	**Characters**	**Floats**	**Doubles**
COMPOUND	**Arrays**	**Structures**	**Unions**	**BitFields**

The basic data types

Type	Typical size in Bits	size in Bytes	Range
char	8	1	-127 to 127
unsigned char	8	1	0 to 255
signed char	8	1	-127 to 127
int	16 or 32	2	-32,767 to 32,767
unsigned int	16 or 32	2	0 to 65,535
signed int	16 or 32	2	-32,767 to 32,767
short int	16	2	-32,767 to 32,767
unsigned short int	16	2	0 to 65,535

long int	32	4	-2,147,483,647 to 2,147,483,647
signed long int	32	4	-2,147,483,647 to 2,147,483,647
unsigned long int	32	4	0 to 4,294,967,295
float	32	4	3.4E-38 to 3.4E+38
double	64	8	1.7E-308 to 1.7E+308
long double	80	10	1.7E-308 to 1.7E+308

Constants

C has four basic type of constants

- integer constants
- floating point constants
- character constants
- string constants

Integer Constanst

An integer constant is an integer valued number. Thus, it consists of a sequence of numbers. Integer constants can be written in three different number system : decimal (base 10), octal (base 8) and hexadecimal (base 16),binary(base 2).

A decimal integer constant can consist of any combination of digits taken from the set 0 through 9. Examples of valid decimal integer constants are : **0 1 743 32767**.

An octal integer constant can consist of digits taken from the set 0 through 7. The first digit must be 0 in order to identify the constant as an octal number. Example of valid octal integer constants are :
01 0743 077777

A hexadecimal integer constant must begin with either 0x or 0X. It can then be followed by any combination of digits taken from the sets 0 through 9 and A through F. Examples of valid hexadecimal integer constants are :**0x 0X1 0X7FFF 0xabcd.**

The magnitude of an integer constant can range from zero to some maximum value. A typical maximum value for most personal computers and many micro computers is 32767 in decimal notation this is equivalent to 77777 octal or 7fff hexadecimal.

An unsigned integer constant can be identified by appending the letter U or u to the end of the constant. Examples are : **50000U (decimal unsigned), 0777777U (octal unsigned), 0X50000U(hexadecimal unsigned).**

Long integer constants can be identified by the letter L or l at the end of the constant. For example, **123456789L(decimal long),0123456L(Octal long).**

An unsigned long integer may be specified by appending the letters UL to the end of the constants. Examples are, **12345678UL(decimal unsigned long), 0XFFFFFUL (hexadecimal unsigned long).**

Floating-point constants

A floating-point constants is decimal number that represents a signed real number. The representation of a floating-point constant an integer portion and a decimal point or an exponent (or both).Example of Valid floating-point constants are,

1.	**0.2**	**827.602**
50000.	**0.000437**	**12.3**
2E-8	**0.006e-3**	**1.6667E+8.12121212e12**

Floating-point constants have a much greater range than integer constants. Typically, the magnitude of a floating-point constant might range from a minimum value of 3.4E-38 to a maximum of 3.4E+38.

Floating-point constants are of the type float, double are long. A floating-point constant without an f,F,l or L suffix is of type of double. If the letter f or F is the suffix, the constant has type float. If the letter l or L is the suffix, it is of the type long double. For example :

```
100L    /* Has type long double */
100F    /* Has type Float */
100D    /* Has type double */
```

The precision of floating-point constants (i.e. the number of significant figures) will vary from one version of C to another.

Character Constants

A character Constant is a single character, enclosed in single quotation marks. Example of character constants are, **'A' 'x' '3' '$'**

Character constants have integer values that are determined by the computer's particular character set. Most computers make use of the ASCII character set, in which each individual character is numerically encoded.

For example

constant	value
'A'	65
'x'	120
'3'	51
'$'	36
' '	32

Escape Sequences

Certain ASCII characters are unprintable, which means they are not displayed on the screen or printer. Those characters perform other functions aside from displaying text. Examples are backspacing, moving to a new line, or ringing a bell. Working with C, you have already worked with \n and \t escape sequences. Escape sequences usually consist of a backslash and a letter or a combination of digits. To represent the creation of nonprinting characters such as a newline character, single quotation mark, or certain other characters such as ",',? And \ escape sequences can be used. An escape sequence is regarded as a single character and is therefore valid as a character constant. Escape sequences are typically used to specify actions such as carriage return and tab movements on terminals and printers. The commonly used escape sequences are listed below.

Escape Sequences	Represents	ASCII Value
\a	Bell(alert)	007
\b	Backspace	008
\f	Form Feed	012
\n	New Line	010
\r	Carriage return	013
\t	Horizontal tab	009
\v	Vertical tab	011
\'	Single quotation mark	039
\"	Double quotation mark	034
\?	Literal quotation mark	063
\\	Backslash	092
\0	Null	000

If a backslash precedes a character that does not appear in the above table, the compiler handles the undefined characters as the character itself and thus the result may be unpredictable.

S	T	R	I	N	G	\ 0

Character Strings

A character string or a string constant consists of any number of consecutive characters (including none) enclosed in double quotations marks. Example of string constants are, **"green","Washington,**

D.C.20005", "$19.95".

Since a string can be of any length the end of the string is marked with the single character '\0' the null character, having an ASCII value of 0. Note that a character constant 'A' and the corresponding single character string constant "A" are not equivalent. Also a character constant has an equivalent integer value. A single character string constant consists of two characters; the specified character followed by the null character(\0). For example, the string "STRING" would be internally stored as.Here, each box represents a memory location containing one character. The string "STRING" would be require 7 elements; six for the string and one for the null character.

Variables

A variable is an identifier used to represent some specified type of information. A variable represents a single data item, that is, a numerical quantity or a character constant etc. The data item must be assigned to the variable at some point in the program. The data item can then be accessed later in the program simply by referring to the variable name. The information represented by the variable can change during the execution of the program by assigning different data items at various places within the program. But the data type associated with the variable cannot change.

Variable Declaration

Before a variable is used in a c program, it must be declared. This involves notifying the compiler of the variable's name and type as follows : **type name**; Where type can be int if the variable is going to hold an integer value, char if this to be a character variable, or float if this is a floating-point variable. The following is an example of variable declaration : **int var_name** ; in the above example, the name of the variable is var_name and it is of type int.

Several variables of the same type can be declared in one statement as well. For example:**int count,flag,state;** Note that each of these variables will be of type int.

Integer-type variables can be declared to be short int or long int for smaller and larger integer quantities respectively. Such integer variables can also be declared by simply writing short an long. For example, **short int a,b,c** and **long int r,s,t**.

An integer variable can also be declared to be unsigned. For example, **unsigned x,y**; on the other hand, an unsigned long integer can be declared as follows, **unsigned long p,q**;

Given below are examples of declaring variables data types.

```
float root1,root2;
char flag,text[80];
double factor;
```

In the above example. 'Text' is a character type array. It can store a character string of maximum length 79 characters. The 80th character is for the delimiter, '\0'. Initial values can be assigned to variables within a type declaration. To do so, the declaration must consist of a data type, followed by a variable name, an equal sign (=) and a constant of the appropriate type. For example :

```
int c = 12;
char star = '$';
float sum = 0.0;
```

on the other hand, a character type array can be initialized within declaration as follows,

```
char text[11]="Villivakkam"
```

Expressions

An expression represents a single data item, such as a number or a character. The expression may consist of a single entity, such as a constant, a variable or a reference too a function. It may also consist of some combination of the above entities interconnected by one or more operators are :

7

```
123.45
'a'
var
var = var + 16
printf("hello \n")
```

Statements

A statement causes the computer to carry out some action. There are two types of statements in the C language ; simple and compound.

Simple statements (expression statements) consist of an expression, followed by a semicolon.
For example.

```
main()
{
        int x,y;
        x=y+3;
}
```

The first line within the body of this program declares two integer variables x and y. The executable portion of the example is made up of several expression are the values 3,y, the expression y+3 and the assignment x=y+3.

A **Compound statement** consist of several individual statements enclosed within a pair braces ({ and }). The individual statements may themselves be expression statement, or compound statements. This collection of statements is treated as a single statement by C compiler. An example of compound statement is shown below :

```
{
        pi = 3.14159;
        circumfrnce = 2 * pi * radius;
        area = pi * radius * radius ;
}
```

The compound statement provides capability for embedding statements within other statements.

Example programs

```
/* To convert a character uppercase to lowercase by initializing a variable */
#include<stdio.h>
main()
{
char c='R';
clrscr();
printf("The character is:%c\n",c);
printf("The character in lowercase is %c\n",c+32);
getch();
}
```

Output:

```
The character is:R
The character in lowercase is r
```

Example

```
/* To initialize the given string */
#include<stdio.h>
main()
{
```

```
char m[11] ="C language";
clrscr();
printf("%s",m);
getch();
}
```

Output:

C language

Example

```
/*Simple calculation  by using initialized integer values  */
#include<stdio.h>
main()
{
int x=150,y=450,z=3,k;
clrscr()  ;
k=(x+y)/z  ;
printf( " k=%d " , k) ;
getch();
}
```

Output:

k=200

Example

```
/* To find the area of a circle by using initialized radius value */
#include<stdio.h>
#define pi 3.14159
main()
{
float radius=2.5,area ;
clrscr();
area=pi*radius*radius;
printf("Area of a circle is %f",area);
getch();
}
```

Output:

Area of a circle is :19.6349375

SOLVED PROGRAMS

Complete the following:

1. C has been developed by _____ in the year _____ while working at _____.
2. Binary equivalent of 762 is _____, its octal equivalent is _____ and its hex equivalent is _____.
3. First character in any variable name must always be an a _____
4. C variables are case _____ (sensitive / insensitive).
5. A character variable can at a time store _____ character(s).

Answers

1. Dennis Ritchie, 1972, American Telegraph & Telecommunication's Bell Laboratories
2. 1011111010, 1372, 2FA 3. Alpha 4. Sensitive 5. One

What will be the output of the following programs:

1.
```
main( )
{
        char ch=291;
        printf("%d %d %c", 32700, ch, ch);
}
```

➲ **Output :**

-32700 35#

2.
```
main( )
{
        int a, b;
        a = -3 - -3;
        b = -3 - -(-3);
        printf ("a= %d b = %d b = %d",a,b);
}
```

➲ **Output :**

a=0 b=6
a = 0 b = -6 b = 1245

3.
```
main( )
{
        int x;
        x=3*4%5;
        printf ("x=%d",x);
}
```

➲ **Output :**

x=2

4.
```
main( )
{
        float a=5, b=2;
        int c;
        c = a %b;
        printf ("%d",c);
}
```

➲ **Output :**

Error message: Illegal use of floating point in function main.

LAB EXERCISE

Exercise # 1

Create and execute the following programs in C Program Editor.

Key in the following program, run it and watch the result.

```
#include <stdio.h>
main()
{
        printf("Welcome to the World! \n");
}
```

Exercise # 2

Write a program that prints your name and address.

Exercise # 3

Write appropriate declarations for each group of variabls and arrays

- Floating-point variabls: root1,root2
  Long intger variables : cust_no
  double precision variables : gross,tax,net

- character variables: first,last
  80-element character array : message

Exercise # 4

Write a program to declare an integer, float, a character variable and also a character array. Initialize the numeric variables to 1 and the character variables to 'a'.

Exercise #5

Identify the type of each of the following variables. Write appropriate declarations and assign the given initial values for each group of variables and array.

A = -8.2 b = 0.005.
c1 = 'w', c2 = '&'
u = 711 (octal), v = ffff (hexadecimal)
EOL = newline character
message = "ERROR"

Exercise # 6

Write a program to define each of the following symbolic constants, as it would appear within a C program.

Constant	Text
Factor	-18
ERROR	0.00001
NAME	"Sharon"
EOLON	'\n'

Exercise # 7

Write a program to display the below statements(Using escape sequence)

i. A Friend in need is a
   Friend in deed
   Tit for tat

ii. Mary had a
   Little lamp,
   Little lamp,
   Little lamp,

Exercise # 8

Study the following program carefully and point out the errors:

```
                mane{ }
(
print( ' I think /n ' )
```

CHAPTER - 2

Data Input/Output and 'C' Operators

Overview

- Introduction
- Single Character Input/Output functions
- The formatted data Output function
- The formatted data Input function
- Conversion Character list
- String Input/Output function
- Operators
- Mixed mode expression and Type Conversion
- Type casting

Introduction

Input/Output functions are used to accept values into variables and printing them after the processing is over. The input and output of data can be done through the standard input/output media or through files. The standard input medium is the **keyboard**, whereas the standard output medium is the **console**.

The C language is accompanied by a collection of **header files** that provide necessary information. Each header file contains information in support of a group of related library functions. These files are included in the program by giving the #include statement at the beginning of the program. The header file required by the standard input/output library functions is called **stdio.h**.

An input/output function can be accessed from anywhere within a program simply by writing the function name, followed by a list of arguments enclosed in parenthesis. The arguments represent data items that are sent to the function.

Some input/output functions return data items whereas others don't. The functions returning data items may appear within expressions, as though each function reference were an ordinary variable; for example **c=getchar().** On the other hand, functions not returning data items may be referenced as though they were separate statements; for example **putchar(..);**

The C library contains two distinct system of routines that handle input/output operations. They are :

- Buffered I/O
- Unbuffered I/O

The buffered I/O is used to read and write ASCII characters whereas the unbuffered I/O is used to read and write binary data.

A **buffer** is a temporary storage area, either in the memory, or on the controller card for the device. In **buffered** I/O, characters typed at the keyboard are collected untill the user presses the RETURN or the ENTER key. The characters are made available to the program, as a block.

Buffered I/O can be further subdivided into Console I/O and buffered File I/O. Console I/O refers to operations that occur at the keyboard and the screen of your computer. Buffered file I/O refers to operations that are performed to read and write data onto a file. In unbuffered I/O, the character which the user enters at the keyboard is made available to the program immediately without being stored temporarily in a buffer location. It is not necessary to press the ENTER key to register the character.

A buffered system is preferred over an unbuferred system mainly for the following reasons:

1. I/O operations on a buffered system is less time consuming, since a block of characters can be transmitted at one time.
2. In the event when mistakes are made and the return key is not pressed, the mistakes can be corrected.

In some cases however, unbuferred I/O is desired in interactive programs. A word processing program is an example.

Problem

1. _____ is the header file that has to be included in a program for accessing standard input and output functions.
2. A buffered system is preferred over an unbuffered system. State True or False.
3. A Buffered I/O is subdivided into _____ and _____.

Single Character Input/Output Functions

The GETCHAR() Function

Using the C Library function - getchar(), one is allowed to input a single character from the terminal. This function is the simplest input mechanism which reads character at a time. It returns a single character that is typed in from the keyboard. The function does not require any argument though a pair of empty parentheses must follow the word getchar. A reference to the getchar function is written as **character variable = getchar();** where character variable refers to some previously declared character variable.

A C program contains the following statements.

```
char c;                 /* Declares c as a character-type variable */
    .
    .
    .
c=getchar();            /* Causes a single character to be entered using the keyboard */
```

To distinguish the end of the input from valid data, the getchar function returns a distinctive value when there is no more input, a value that cannot be confused with any real character. This value is called EOF, which denotes “end of file”. **EOF** is an integer defined in the library file <stdio.h>. Typically EOF will be assigned the value -1. However , this value may vary from one compiler to another.

On machines where characters are unsigned, the value of EOF (-1) cannot be stored properly in the character type variable, c. Therefore c must be declared to be a type big enough to hold EOF in addition to any possible character. Therefore the variable should be declared to be of integer type.

Pressing the keys CONTROL-Z causes <EOT> (End Of Terminal) to be sent to the program [Not that this varies from machine to machine]. This character is interpreted by getchar() as an end of file and getchar() then returns the defined value of EOF.

Given below is a sample program that shows how CTRL+Z can be used to denote EOF.

Example

```
#include<stdio.h>
main()
{
  int c;
  printf("Enter character (CONTROL-Z to end): ");
   c=getchar();
   if(c!=EOF)
     printf("End of file not encountered !");
}
```

Output:

```
Enter character (CONTROL-Z to end):R
End of file not encountered !
```