JAWAHARLAL COLLEGE OF ENGINEERING AND TECHNOLOGY
(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(NBA Accredited)

COURSE MATERIAL

EST 102 PROGRAMMING IN C (LAB MANUAL)

VISION OF THE INSTITUTION

Emerge as a centre of excellence for professional education to produce high quality engineers andentrepreneurs for the development of the region and the Nation

MISSION OF THE INSTITUTION

- To become an ultimate destination for acquiring latest and advanced knowledge in the multidisciplinary domains.
- To provide high quality education in engineering and technology through innovative teaching-learning practices, research and consultancy, embedded with professional ethics.
- To promote intellectual curiosity and thirst for acquiring knowledge through outcome based education.
- To have partnership with industry and reputed institutions to enhance the employability skills of the students and pedagogical pursuits.
- To leverage technologies to solve the real life societal problems through community services.

ABOUT THE DEPARTMENT

- Established in: 2008
- Courses offered: B. Tech in Computer Science and Engineering
- Affiliated to the A P J Abdul Kalam Technological University.

DEPARTMENT VISION

To produce competent professionals with research and innovative skills, by providing them with the most conducive environment for quality academic and research oriented undergraduate education along with moral values committed to build a vibrant nation.

DEPARTMENT MISSION

- Provide a learning environment to develop creativity and problem solving skills in a professional manner.
- Expose to latest technologies and tools used in the field of computer science.
- Provide a platform to explore the industries to understand the work culture and expectation of an organization.
- Enhance Industry Institute Interaction program to develop the entrepreneurship skills.
- Develop research interest among students which will impart a better life for the society and the nation.

PROGRAMME EDUCATIONAL OBJECTIVES

Graduates will be able to

- Provide high-quality knowledge in computer science and engineering required for a computer professional to identify and solve problems in various application domains.
- Persist with the ability in innovative ideas in computer support systems and transmit the knowledge and skills for research and advanced learning.
- Manifest the motivational capabilities, and turn on a social and economic commitment to community services.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration

for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clearinstructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leaderin a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Outcomes: After the completion of the course the student will be able to

CO 1	Analyze a computational problem and develop an algorithm/flowchart to find its solution
CO 2	Develop readable* C programs with branching and looping statements, which uses Arithmetic, Logical, Relational or Bitwise operators.
CO 3	Write readable C programs with arrays, structure or union for storing the data to beprocessed
CO 4	Divide a given computational problem into a number of modules and develop a readable multi-function C program by using recursion if required, to find the solution to the computational problem
CO 5	Write readable C programs which use pointers for array processing and parameter passing
CO 6	Develop readable C programs with files for reading input and storing output

readable* - readability of a program means the following:

1. Logic used is easy to follow
2. Standards to be followed for indentation and formatting
3. Meaningful names are given to variables
4. Concise comments are provided wherever needed

PROGRAM SPECIFIC OUTCOMES (PSO)

The students will be able to

- Use fundamental knowledge of mathematics to solve problems using suitable analysismethods, data structure and algorithms.
- Interpret the basic concepts and methods of computer systems and technicalspecifications to provide accurate solutions.
- Apply theoretical and practical proficiency with a wide area of programming knowledge,design new ideas and innovations towards research.

Prerequisite: NIL

	PO 1	PO 2	PO 3	PO 4	PO 5	PO6	PO 7	PO 8	PO 9	PO1 0	PO1 1	PO1 2
CO 1	✓	✓	✓	✓		✓				✓	✓	✓
CO 2	✓	✓	✓	✓	✓					✓		✓
CO 3	✓	✓	✓	✓	✓					✓		✓
CO 4	✓	✓	✓	✓	✓					✓	✓	✓
CO 5	✓	✓			✓					✓		✓
CO 6	✓	✓			✓					✓		✓

Assessment Pattern

	Continuous Assessment Tests	End Semester

Bloom's Category	Test 1 (Marks)	Test 2 (Marks)	Examination Marks
Remember	15	10	25
Understand	10	15	25
Apply	20	20	40
Analyse	5	5	10

Mark distribution

Total Marks	CIE Marks	ESE Marks	ESE Duration
150	50	100	3 hours

Attendance	: 10 marks
Continuous Assessment Test 1 (for theory, for 2 hrs)	: 20 marks
Continuous Assessment Test 2 (for lab, internal examination, for 2 hrs)	: 20 marks

Internal Examination Pattern: There will be two parts; Part A and Part B. Part A contains 5 questions with 2 questions from each module (2.5 modules x 2 = 5), having 3 marks for each question. Students should answer all questions. Part B also contains 5 questions with 2 questions from each module (2.5 modules x 2 = 5), of which a student should answer any one. The questions should not have sub- divisions and each one carries 7 marks.

End Semester Examination Pattern: There will be two parts; Part A and Part B. Part A contains 10 questions with 2 questions from each module, having 3 marks for each question. Students should answer all questions. Part B contains 2 questions from each module of which a student should answerany one. Each question can have maximum 2 sub-divisions and carry 14 marks.

Sample Course Level Assessment Questions

Course Outcome 1 (CO1): Write an algorithm to check whether largest of 3 natural numbers is prime or
not. Also, draw a flowchart for solving the same problem.

Course Outcome 2 (CO2): Write an easy to read C program to process a set of n natural numbers and to find the largest even number and smallest odd number from the given set of numbers. The program should not use division and modulus operators.

Course Outcome 3(CO3): Write an easy to read C program to process the marks obtained by n students of a class and prepare their rank list based on the sum of the marks obtained. There are 3 subjects for which examinations are conducted and the third subject is an elective where a student is allowed to take any one of the two courses offered.

Course Outcome 4 (CO4): Write an easy to read C program to find the value of a mathematical functionf which is defined as follows. f(n) = n! / (sum of factors of n), if n is not prime and f(n) = n! / (sum of digits of n), if n is prime.

Course Outcome 5 (CO5): Write an easy to read C program to sort a set of n integers and to find the number of unique numbers and the number of repeated numbers in the given set of numbers. Use a function which takes an integer array of n elements, sorts the array using the Bubble Sorting Technique and returns the number of unique numbers and the number of repeated numbers in the given array.
Course Outcome 6 (CO6): Write an easy to read C program to process a text file and to print thePalindrome words into an output file.

C PROGRAMMING LAB (Practical part of EST 102, Programming in C)

Assessment Method: The Academic Assessment for the Programming lab should be done internally by the College. The assessment shall be made on 50 marks and the mark is divided as follows: Practical Records/Outputs - 20 marks (internal by the College), Regular Lab Viva - 5 marks (internal by the College), Final Practical Exam – 25 marks (internal by the College).

The mark obtained out of 50 will be converted into equivalent proportion out of 20 for CIE computation.

LIST OF LAB EXPERIMENTS

1. Familiarization of Hardware Components of a Computer
2. Familiarization of Linux environment – How to do Programming in C with Linux
3. Familiarization of console I/O and operators in C
    i) Display “Hello World”

ii) Read two numbers, add them and display their sum

iii) Read the radius of a circle, calculate its area and display it

iv) Evaluate the arithmetic expression ((a -b / c * d + e) * (f +g)) and display its solution. Read the values of the variables from the user through console.

4. Read 3 integer values and find the largest among them.
5. Read a Natural Number and check whether the number is prime or not
6. Read a Natural Number and check whether the number is Armstrong or not
7. Read n integers, store them in an array and find their sum and average
8. Read n integers, store them in an array and search for an element in thearray using an algorithm for Linear Search
9. Read n integers, store them in an array and sort the elements in the array using Bubble Sortalgorithm
10. Read a string (word), store it in an array and check whether it is a palindrome word or not.
11. Read two strings (each one ending with a $ symbol), store them inarrays and concatenate them without using library functions.
12. Read a string (ending with a $ symbol), store it in an array and count the number of vowels,consonants and spaces in it.
13. Read two input each representing the distances between two points in the Euclidean space,store these in structure variables and add the two distance values.
14. Using structure, read and print data of n employees (*Name, Employee Id and Salary*)
15. Declare a union containing 5 string variables (*Name, House Name, City Name, State and Pin code*) each with a length of C_SIZE (user defined constant). Then, read and display the address of a person using a variable of the union.
16. Find the factorial of a given Natural Number n using recursive and non-recursive functions
17. **R**ead a string (word), store it in an array and obtain its reverse by using a user defined function.
18. Write a menu driven program for performing matrix addition, multiplication and finding the transpose. Use functions to (i) read a matrix, (ii) find the sum of two matrices, (iii) find the product of two matrices, (i) find the transpose of a matrix and (v) display a matrix.
19. Do the following using pointers?
    - **i)** add two numbers
    - **ii)** swap two numbers using a user defined function
20. Input and Print the elements of an array using pointers
21. Compute sum of the elements stored in an array using pointers and user

defined function.

22. Create a file and perform the following
    - iii) Write data to the file
    - iv) Read the data in a given file & display the file content on console
    - v) append new data and display on console
23. Open a text input file and count number of characters, words and lines in it; and store the results

in an output file

Progamming in

Lab Manual

EST 102

COURSE OBJECTIVE

To understand the fundamental concept of programming and use it in problem solving.

COURSE OUTCOME

After the completion of the course the student will be able to

1. Analyze a computational problem and develop an algorithm/flowchart to find its solution.
2. Develop readable* C programs with branching and looping statements, which uses Arithmetic, Logical, Relational or Bitwise operators.
3. Write readable C programs with arrays, structure or union for storing the data to be processed.
4. Divide a given computational problem into a number of modules and develop a readable multi-function C program by using recursion if required, to find the solution to the computational problem.
5. Write readable C programs which use pointers for array processing and parameter passing.
6. Develop readable C programs with files for reading input and storing output.

INDEX

LIST OF LAB EXPERIMENTS

1. Familiarization of Hardware Components of a Computer

Computer hardware refers to the physical parts or components of a computer such as the monitor, mouse, keyboard, computer data storage, hard drive disk (HDD), system unit (graphic cards, sound cards, memory, motherboard and chips), etc. all of which are physical objects that can be touched.

The main hardware components are listed below:-

- Microprocessor
- Motherboard
- RAM
- Hard Disk Drive
- Optical disc drive [CD / DVD Drive]
- Keyboard
- Mouse
- Monitor
- Computer case and SMPS
- Computer Speaker
- Uninterrupted power supply (UPS)

Microprocessor

A microprocessor is an electronic component that is used by a computer to do its work. It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together.

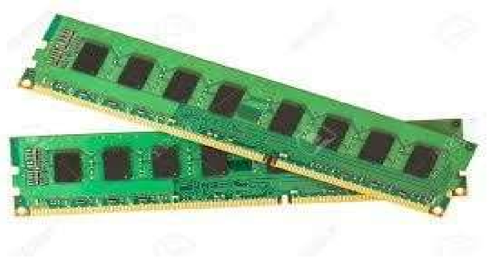
A microprocessor incorporates most or all of the functions of a central processing unit (CPU) on a single integrated circuit (IC).

Motherboard

A motherboard is the central or primary printed circuit board (PCB) making up a complex electronic system, such as a modern computer. It is also known as a main board, baseboard, system board, planar board, or, on Apple computers, logic board, and is sometimes abbreviated casually as mobo.

RAM

Random access memory (usually known by its acronym, RAM) is a type of computer data storage. Today it takes the form of integrated circuits that allow the stored data to be accessed in any order, i.e. at random.

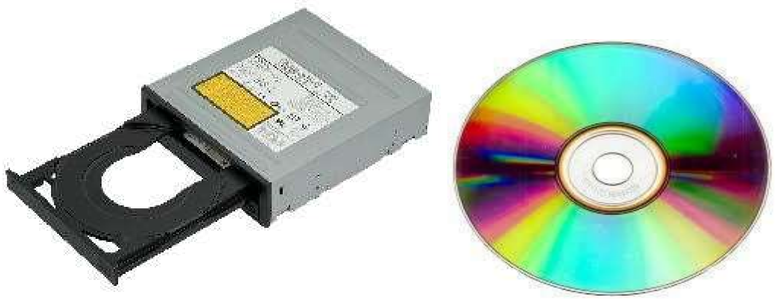


Hard Disk Drive

A hard disk drive (HDD), commonly referred to as a hard drive, hard disk, or fixed disk drive, is a non-volatile storage device which stores digitally encoded data on rapidly rotating platters with magnetic surfaces. Strictly speaking, "drive" refers to a device distinct from its medium, such as a tape drive and its tape, or a floppy disk drive and its floppy disk. Early HDDs had removable media; however, an HDD today is typically a sealed unit.

Optical disc drive [CD / DVD Drive]

An optical disc drive (ODD) is a disk drive that uses laser light or electromagnetic waves near the light spectrum as part of the process of reading and writing data. It is a computer's peripheral device that stores data on optical discs. Some drives can only read from discs, but commonly drives are both readers and recorders. Recorders are sometimes called burners or writers.



Keyboard

A keyboard is an arrangement of buttons, or keys. A keyboard typically has characters engraved or printed on the keys; in most cases, each press of a key corresponds to a single written symbol.

Mouse

A mouse (plural mice, mouse devices, or mouses) is a pointing device that functions by detecting two-dimensional motion relative to its supporting surface. Physically, a mouse consists of a small case, held under one of the user's hands, with one or more buttons.

Monitor

A monitor is a piece of computer hardware that displays the video and graphics information generated by a connected computer through the computer's video card.

Computer case and SMPS

A computer case is the enclosure that contains the main components of a computer. Cases are usually constructed from steel, aluminum, or plastic, although other materials such as wood, plexiglas or fans have also been used in case designs. Cases can come in many different sizes, or form factors.

A switched-mode power supply, switching-mode power supply or SMPS, is an electronic power supply unit (PSU) that incorporates a switching regulator. While a linear regulator maintains the desired output voltage by dissipating excess power in a "pass" power transistor, the SMPS rapidly switches a power transistor between saturation (full on) and cutoff (completely off) with a variable duty cycle whose average is the desired output voltage.

Computer Speaker

Computer speakers, or multimedia speakers, are external speakers, commonly equipped with a low-power internal amplifier. The standard audio connection is 3.5mm (1/8 inch) stereo jacks plug often colour-coded lime green (following the PC 99 standard) for computer sound cards.

Uninterrupted power supply (UPS)

An uninterruptible power supply (UPS), also known as a continuous power supply (CPS) or a battery backup is a device which maintains a continuous supply of electric power to connected equipment by supplying power from a separate source when utility power is not available.

Small UPS systems provide power for a few minutes; enough to power down the computer in an orderly manner, while larger systems have enough battery for several hours.

2. Familiarization of Linux environment – How to do Programming in C with Linux

Student can login into the system using the username and password then take the terminal, for getting command terminal, search for terminal or use shortcut key Ctrl+Alt+t. In terminal each student can login into the lab server using ssh command.

ssh login name@server IP address then press the enter key

Type password then press the enter key

After login 'vi' editor is used for doing programs.

vi program name.c

Press **insert key** to type the program

For saving the program, press **Esc** key then type ***:wq***

For compile the program using cc command: ***cc*** **program name.c**

After successful compilation take the output of the program using ***./a.out*** command.

LINUX COMMANDS

1. ***ls*** :- used to list the files and directories under the current directory.
   Syntax:- ***ls*** press enter key
2. ***rm***:- used to remove a particular file.
   Syntax:- rm < file name>
3. ***mkdir***:- used to create a directory.
   Syntax:- mkdir <directory name>
4. ***cd***:- used to change the directory.
   Syntax:- cd <directory name>
5. ***cd ..*** :-used to leave from a particular directory.
   Syntax:- cd .. press enter key
6. ***rmdir***:- used to remove an empty directory.
   Syntax:- rmdir <empty directory name>

7. ***cp*** :- used copy a file to another file.

   Syntax:- cp <source file name> <destination file name>

8. **mv**:- used to moves files or directories from one place to another.

   Syntax:- mv <source file name> <destination file name>

9. ***man***:- used to displays the whole manual of the command.

   Syntax: man <command name>

3. Introduction to C

C is a general-purpose, high-level language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. The UNIX operating system, the C compiler, and essentially all UNIX applications programs have been written in C. The C has now become a widely used professional language for various reasons.

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

Facts about C

C was invented to write an operating system called UNIX. C is a successor of B language which was introduced around 1970. The language was formalized in 1988 by the American National Standard Institute (ANSI). The UNIX OS was totally written in C by 1973. Today C is the most widely used and popular System Programming Language. Most of the state-of-the-art software have been implemented using C. Today's most popular Linux OS and RBDMS MySQL have been written in C.

Why to use C?

C was initially used for system development work, in particular the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs

- Databases
- Language Interpreters
- Utilities

The largest measure of C's success seems to be based on purely practical considerations:

1. The portability of the compiler
2. The standard library concept
3. A powerful and varied repertoire of operators
4. An elegant syntax
5. Ready access to the hardware when needed
6. And the ease with which applications can be optimized by hand-coding isolated procedures

Structure of a C program

```
Documentation Section
Link Section /Include header file section
Definition Section
Global declaration section
Main() function section
{
              Declaration part
              Executable part
}
Subprogram section
 Function 1
 Function 2
     …
 Function n
(User defined functions)
```

Documentation Section:-

Documentation section consists of a set of comment lines giving the name of the program, the author and other details.

Link section:-

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

Definition Section:-

Definition section defines all symbolic constants.

Global Declaration:-

This section declares some variables that are used in more than one function. This section also declares all the user defined functions.

Main() function section:-

Every C program must contain a main() function. This section contains two parts, declaration and executable parts. Declaration part declares all the variables used in the executable part. There is at least one statement in the executable part. Executable part contains the statements following the declaration of the variables.

Subprogram section:-

This section contains all the user defined functions that are called in the main() function. All sections, except the main () function and link section may be absent when they are not required.

Character set of C language

The character set in C Language can be grouped into the following categories.

1.Letters
2.Digits
3.Special characters
4.White Spaces

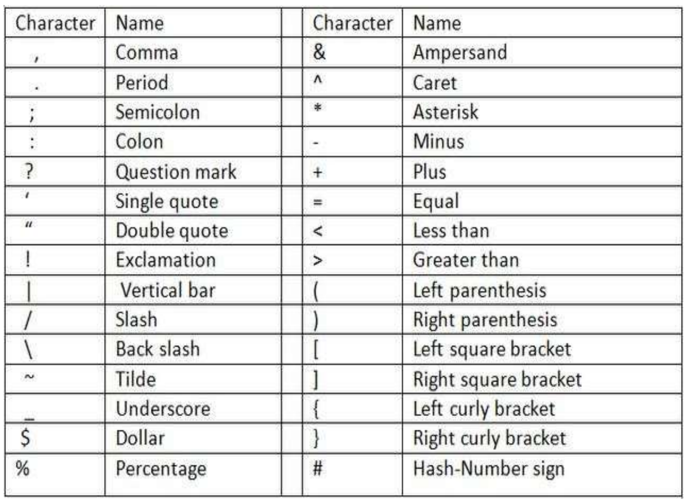
White Spaces are ignored by the compiler until they are a part of string constant. White Space may be used to separate words, but are strictly prohibited while using between characters of keywords or identifiers.

C Character set table

The characters in C are grouped in to 4

1. Letters(A-Z, a-z)
2. Digits (0-9)
3. Special Characters
4. White spaces(used to separate words)

Special Characters



Character	Name		Character	Name
,	Comma		&	Ampersand
.	Period		^	Caret
;	Semicolon		*	Asterisk
:	Colon		-	Minus
?	Question mark		+	Plus
'	Single quote		=	Equal
"	Double quote		<	Less than
!	Exclamation		>	Greater than
|	Vertical bar		(	Left parenthesis
/	Slash		)	Right parenthesis
\	Back slash		[	Left square bracket
~	Tilde		]	Right square bracket
_	Underscore		{	Left curly bracket
$	Dollar		}	Right curly bracket
%	Percentage		#	Hash-Number sign

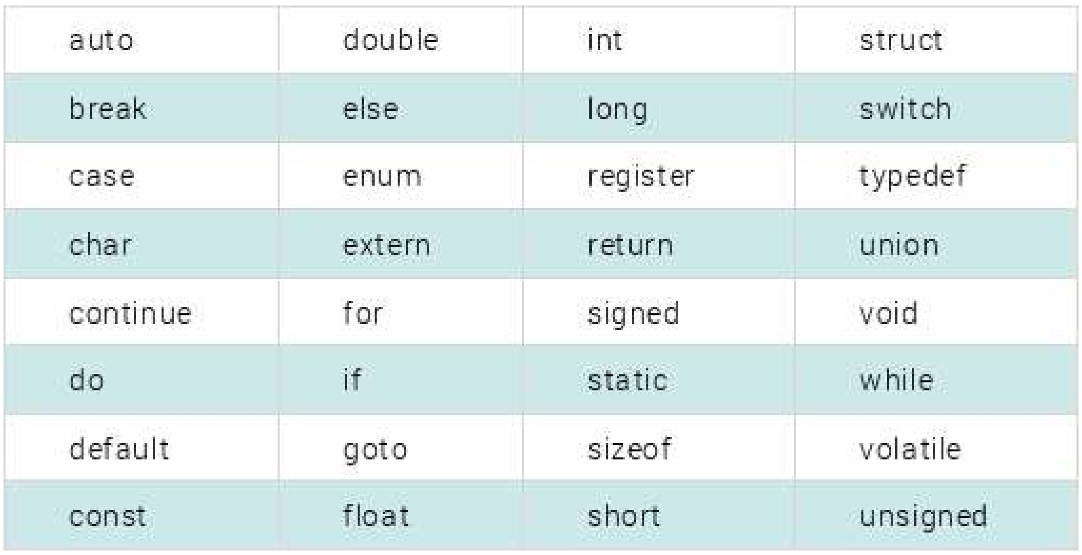
Escape sequences

1. BlankSpace(\b)
2. HorizontalTab(\t)
3. CarriageReturn(\r)
4. NewLine(\n)
5. Form Feed (\f)
6. Vertical Tab (\v)

Keywords and Identifiers

Every word in C language is a keyword or an identifier. Keywords are reserved words that have standard, predefined meanings in C. All keywords must be written in lowercase. Keywords in C language cannot be used as a variable name. They are specifically used by the compiler for its own purpose and they serve as building blocks of a c program.

The following are the 32 Keywords of C language.



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

Identifier refers to the name of user-defined variables, array and functions. A variable should be essentially a sequence of letters and or digits and the variable name should begin with a character.

Both uppercase and lowercase letters are permitted. The underscore character is also permitted in identifiers. The underscore (_) symbol can be used as an identifier.

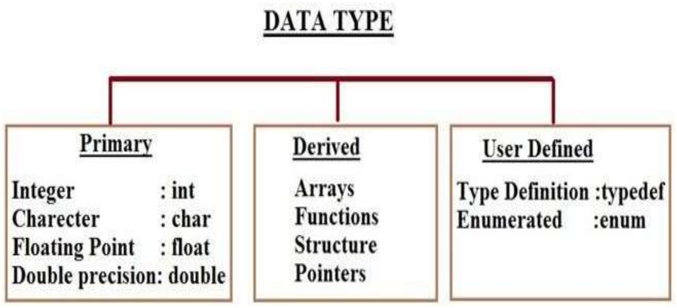
Some examples of identifiers are: tax_rate, _temp, place etc.

Data Types

Data types are used to store various types of data that is processed by program. The definition of a variable will assign storage for the variable and define the type of data that will be held in the location.

C has different data types for different types of data and can be broadly classified as:

1. Primary data types
2. Secondary data types



Primary data types available in c are

Data Type	**Description**	**Memory Requirements**
int	integer quantity	2 bytes or 1 word
float	floating point number	1 word(4 bytes)
char	single character	1 byte
double	double precision number	2 words(8 bytes)

Integer Data Type:-

Integer data types are used to define integer numbers. Integers are whole numbers with a range of values supported by a particular machine. Generally an integer occupies 2 bytes memory space and its value range limited to -32768 to +32767 (that is, -2^{15} to $+2^{15}-1$).

A signed integer use one bit for storing sign and rest 15 bits for number. To control the range of numbers and storage space, C has three classes of integer storage namely short int, int and long int. All three data types have signed and unsigned forms.

A short int requires half the amount of storage than normal integer. Unlike signed integer, unsigned integers are always positive and use all the bits for the magnitude of the number. Therefore the range of an unsigned integer will be from 0 to 65535. The long integers are used to declare a longer range of values and it occupies 4 bytes of storage space.

Syntax: int <variable name>;

Examples are:

```
int num1;
short int num2;
long int num3;
```

Character Type:

Character type variable can hold a single character. As there are singed and unsigned int (either short or long), in the same way there are signed and unsigned chars; both occupy 1 byte each, but having different ranges. Unsigned characters have values between 0 and 255, signed characters have values from –128 to 127.

Syntax: char <variable name>;

char ch = 'a';

Floating Point and Double Types:

The float data type is used to store fractional numbers (real numbers) with 6 digits of precision. Floating point numbers are denoted by the keyword float. When the accuracy of the floating point number is insufficient, we can use the double to define the number. The double is same as float but with longer precision and takes double space (8 bytes) than float. To extend the precision further long double can be used which occupies 10 bytes of memory space.

Syntax: float <variable name>; like

float num1;

double num2;

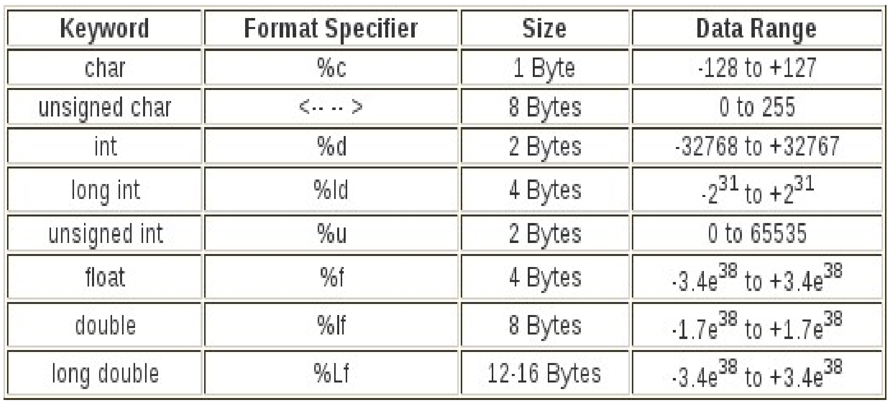
long double num3;

Example: 9.125, 3.1254

Void Type:

The void type has no values therefore it cannot be declared as a variable. The void data type is usually used with function to specify its type. Like in our first C program we declared "main()" as void type because it does not return any value. The concept of returning values will be discussed in detail in the C function hub.

Data Types in C, Size & Range of Data Types



Keyword	Format Specifier	Size	Data Range
char	%c	1 Byte	-128 to +127
unsigned char	<-- -- >	8 Bytes	0 to 255
int	%d	2 Bytes	-32768 to +32767
long int	%ld	4 Bytes	-2^{31} to $+2^{31}$
unsigned int	%u	2 Bytes	0 to 65535
float	%f	4 Bytes	$-3.4e^{38}$ to $+3.4e^{38}$
double	%lf	8 Bytes	$-1.7e^{38}$ to $+1.7e^{38}$
long double	%Lf	12-16 Bytes	$-3.4e^{38}$ to $+3.4e^{38}$

Constants

A constant is a number, character, or a character string that can be used as a value in a program. Use constants to represent floating-point, integer, enumeration, or character values that cannot be modified. Constants are fixed values that do not change during the execution of a program. C has 4 basic types of constants.

Integer constant

Integer constants are a sequence of digits. It can be written in 3 different number systems

(i) Decimal Integer Constant:

- 0 to 9
- E.g: 49, 58, -62, … (40000 cannot come bcoz it is > 32767)

(ii) Octal Integer Constant:

- 0 to 7
- Add "0" before the value.
- Eg.: 045, 056, 067

(iii) Hexadecimal Integer:

- 0 to 9 and A to F
- Add 0x before the value
- E.g: 0x42, 0x56, 0x67

Floating point constants

Floating point constants are a sequence of digits, followed by a decimal point, followed by a sequence of digits, and optionally followed by an exponent. Ex: +867.9, -26.9876, 654.0 .In exponential form, the real constant is represented as two parts. The part lying before the 'e' is the 'mantissa', and the one following 'e' is the 'exponent'. For example, the value 237.95 may be written as 23795E2 in exponential notation. E2 means multiply by 102.

Character constants

Character constants are a single character surrounded by single quotes (`''), or a number--the ordinal value of the corresponding character (usually its ASCII value). Within quotes, the single character may be represented by a letter or by "escape sequences. Example of character constants are '3','c'. Since each character constant represents an integer value, it is also possible to perform arithmetic operations on character constants.

String constants

String constants are a sequence of character constants surrounded by double quotes (`""). The character may be letters, numbers, special characters and blank space. Examples are "hello","65". Each string constant always ends with a special character '\0'. The compiler automatically places a null character (\0) at the end of every string constant.

Variables

A variable is a value that can change any time. It is a memory location used to store a data value. A variable name should be carefully chosen by the programmer so that its use is reflected in a useful way in the entire program. Variable names are case sensitive. Example of variable names are,

Sun
number
Salary
Emp_name
average1

Any variable declared in a program should confirm to the following

1. They must always begin with a letter, although some systems permit underscore as
   first character.
2. The length of a variable must not be more than 8 characters.
3. White space is not allowed
4. A variable should not be a Keyword
5. It should not contain any special characters.

Variable declaration

The declaration of variables should be done in the declaration part of the program. These variables must be declared before they are used in the program. The declaration provides two things:

1) Compiler obtains the variable name
2) It tells the compiler, the data type of the variable being declared and helps in allocating memory.
3) The syntax of declaring a variable is as follows

```
data_type variable_name;
```

Here data_type must be a valid data type and variable_name is an identifier.

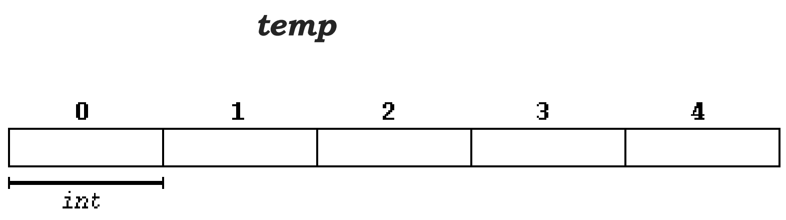
Some declarations are

```
int tax;

float tax_rate,count;
```

Arrays

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.That means that, for example, we can store 5 values of type int in an array without having to declare 5 different variables, each one with a different identifier. Instead of that, using an array we can store 5 different values of the same type, int is an example of a unique identifier. For example, an array to contain 5 integer values of type int called temp could be represented like this:



Where each blank panel represents an element of the array of type integer values. These elements are numbered from 0 to 4 since in arrays the first index is always 0, independently of its length.

Declaring Arrays

An array is declared by specifying its data type, name and the number of elements the array holds between square brackets immediately following the array name.
Here is the syntax:

data_type array_name[size];

For example, to declare an integer array which contains 100 elements we can do as follows:

```
int a[100];
```

There are some rules on array declaration. The data type can be any valid C data types including structure and union. The array name has to follow the rule of variable and the size of array has to be a positive constant integer.
Array elements can be accessed via indexes array_name[index]. Indexes of array starts from 0 not 1 so the highest elements of an array is array_name[size-1].

Initializing Arrays

It is like a variable, an array can be initialized. To initialize an array, provide initializing values which are enclosed within curly braces in the declaration and placed following an equals sign after the array name. Here is an example of initializing an integer array.

```
int a[3]={1,2,3};
```

Array and Pointer

Each array element occupies consecutive memory locations and array name is a pointer that points to the first element. Beside accessing array via index we can use pointer to manipulate array. This program helps you visualize the memory address each array elements and how to access array element using pointer.

Multidimensional Arrays

An array with more than one index value is called a multidimensional array. All the array above is called single-dimensional array. To declare a multidimensional array you can do follow syntax

datatype arrayname[size][size][size]

The number of square brackets specifies the dimension of the array. For example to declare two dimensions integer array we can do as follows:

int matrix[3][3];

Statements in C

The statements of a C program control the flow of program execution. In C, as in other programming languages, several kinds of statements are available to perform loops, to select other statements to be executed, and to transfer control. C statements consist of tokens, expressions, and other statements. A statement that forms a component of another statement is called the "body" of the enclosing statement.

Frequently the statement body is a "compound statement." A compound statement consists of other statements that can include keywords. The compound statement is delimited by braces ({ }). All other C statements end with a semicolon (;). The semicolon is a statement terminator.

The expression statement contains a C expression that can contain the arithmetic or logical operators introduced in Expressions and Assignments. The null statement is an empty statement. Any C statement can begin with an identifying label consisting of a name and a colon.

C Operator Precedence and Associativity

Operator	Description	Associativity
()	Parentheses (function call) (see Note 1)	left-to-right
[]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment/decrement (see Note 2)	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(*type*)	Cast (change *type*)	
*	Dereference	
&	Address	
sizeof	Determine size in bytes	
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
?:	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	

^= |= <<= >>=	Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

Note 1:
Parentheses are also used to group sub-expressions to force a different precedence; such parenthetical expressions can be nested and are then evaluated from inner to outer.
Note 2:
Postfix increment/decrement have high precedence, but the actual increment or decrement of the operand is delayed (to be accomplished sometime before the statement completes execution). So in the statement **y = x * z++;** the current value of **z** is used to evaluate the expression (*i.e.,* **z++** evaluates to **z**) and **z** only incremented after all else is done.

4. Experiments

4.1 Familiarization of console I/O and operators in C

i) Display “Hello World”

ii) Read two numbers add them and display their sum

iii) Read the radius of a circle, calculate its area and display it

iv) Evaluate the arithmetic expression ((a -b / c * d + e) * (f +g)) and display its solution. Read the values of the variables from the user through console.

i) ALGORITHM

```
1. Start
2. Print hello world
3. Stop
```

ii) ALGORITHM

```
1. Start
2. Read a,b
3. sum=a+b
4. Print sum
5. Stop
```

iii) ALGORITHM

```
1. Start
2. Read r
3. area=3.14*r*r
4. Print area
5. Stop
```

iv) ALGORITHM

```
1. Start
```

```
2. Read a,b,c,d,e,f,g
3. s=((a -b / c * d + e) * (f +g))
4. Print s
5. Stop
```

4.2 Read 3 integer values and find the largest among them.

ALGORITHM

```
1. Start
2. Read a,b,c
3. if(a > b && a>c) then print a is largest else goto step 4
4. if(b > a && b>c ) then print b is largest else print c is largest
5. Stop
```

4.3 Read a Natural Number and check whether the number is prime or not

ALGORITHM

```
1. Start
2. Read num
3. Set flag=0, i = 2
4. If (num num/2) then goto step 4.1 else goto
   4.1   if( num % 2 == 0) then set flag=1 else goto step 4.2
   4.2   i=i+1, goto step 4
5. if (flag ==0) then print num is prime else print num is not prime
6. Stop
```

4.4 Read a Natural Number and check whether the number is Armstrong or not

ALGORITHM

```
1. Start
2. sum = 0
3. Read n
```

```
4.  temp=n
5.  If (temp != 0) then goto step 6 else goto step 9
6.  t = temp%10
7.  sum = sum+t*t*t
8.  temp = temp/10 goto step 5
9.  if(n == sum) then print it is an armstrong no else print it is not an
armstrong no
10. Stop
```

4.5 Read n integers store them in an array and find their sum and average

ALGORITHM

```
    1.  Start
    2.  Read the limit n
    3. sum =0,i=0
    4.  if(i<n) then goto step 5 else goto step 8
    5.  Read the number A[i]
    6.  sum = sum + A[i]
    7.  i=i+1, then goto step 4
    8.  avg = sum / n
    9.  Print sum, avg
    10.Stop
```

4.6 Read n integers, store them in an array and search for an element in the array using an algorithm for Linear Search

ALGORITHM

```
  START
    1.  Start
    2.  Read the limit n
    3.  Read the number to be search num
    4.  i=0
    5. if(i<n) then goto step 6 else goto step 8
```

```
6. Read the number A[i]
7. i=i+1, then goto step 5
8. i=0
9. if(i<n) then goto step 10 else goto step 11
10. if(A[i]==num) then print number is present and goto step 11, else
    i=i+1 and goto step 9
10. Stop
```

4.7 Read n integers, store them in an array and sort the elements in the array using Bubble Sort

ALGORITHM

```
1. Start
2. Read the limit of array to n
3. i=0
4. if(i<n) then goto step 5 else goto step 7
5. Read the number A[i]
6. i=i+1, then goto step 4
7. i=0
8. if(i < n) then goto step 9 else goto step 10
    8.1. j=0
    8.2 if (j< n-i-1)goto step 11 else goto step 9
    8.3 if(a[j] > a[j+1])then goto step 8.4 else goto step
    8.4 temp=a[j]
    8.5 a[j]=a[j+1]
    8.6 a[j+1]=temp
    8.7 j=j+1,goto step 8.2
9. i=i+1,goto step 8
10. Print the sorted array
11. Stop
```

4.8 Read a string (word), store it in an array and check whether it is a palindrome word or not.

ALGORITHM

```
1. start
```

```
2. Read the string to str
3. Find length of str.to n.
4. flag=0
5. l=0
6. h=n-1
7. if(l<=n/2) then goto step 8 else goto step 10
8. If (str[l] !=str[h]) then flag=1and goto step 10 else goto step 9
9. l=l+1 and h=h-1,goto step 7
10. if(flag==1) then print string is not a palindrome else print string is a
    palindrome
11.       Stop
```

4.9 Read two strings (each one ending with a $ symbol), store them in arrays and concatenate them without using library functions.

ALGORITHM

```
1. Start
2. Read two strings to a and b
3. Find length of a to n
4. i=0,j=n-1
5. (if b[i]!= '\0') then a[j]=b[i] else goto step 7
6. i=i+1,j=j+1 then goto step 5
7. a[j]='\0'
8. Print a
9. Stop
```

4.10 Read a string (ending with a $ symbol), store it in an array and count the number of vowels, consonants and spaces in it.

ALGORITHM

```
1. Start
2. Read the string to str
3. vowels=0,Consonant=0,space=0,i=0
4. if(str[i]!='\0') then goto step 5 else goto step 9
```

```
5. if (str[i] == 'a' ||str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] ==
   'u' || str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O'
   ||str[i] == 'U') then vowels=vowels+1 else goto step 6
6. if ((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z')) then
   consonant =consonant+1 else goto step 7
7. if (str[i] == ' ') then space=space +1 and goto step 8
8. i=1+1,goto step 4
9. Print vowels,Consonant,space
10.Stop
```

4.11 **Read two input each representing the distances between two points in the Euclidean space, store these in structure variables and add the two distance values.**

Note:

In mathematics, the **Euclidean distance** or **Euclidean** metric is the "ordinary" straight-line **distance** between two points in **Euclidean** space. With this **distance**, **Euclidean** space becomes a metric space. The associated norm is called the **Euclidean** norm.

1. Take the coordinates of two points you want to find the distance between. Call one point Point 1 (x1,y1) and make the other Point 2 (x2,y2).
2. Know the distance formula. ...
3. Find the horizontal and vertical distance between the points
4. Square both values
5. Add the squared values together
6. Take the square root of the equation.

ALGORITHM

```
1. Start
2. Read x1,x2,y1,y2 using structure
3. s= ((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1))
4. distance = square root (s)
5. Print distance
6. Stop
```

4.12 Using structure, read and print data of n employees (Name, Employee Id and Salary)

ALGORITHM

1. Start
2. Declare a structure *employee* with a variable emp and structure members name, e_id and salary
3. read name, e_id and salary using structure variable emp.
4. print name, e_id and salary using structure variable emp.
5. stop

4.13 Declare a union containing 5 string variables (Name, House Name, City Name, State and Pin code) each with a length of C_SIZE (user defined constant). Then, read and display the address of a person using a variable of the union.

ALGORITHM

1. Start
2. Define a variable C_SIZE with a constant value.
3. Declare a structure *union* with a variable emp and union members Name, House_Name, City_Name, State and Pin code
4. Read name using union variable emp
5. Print name using union variable emp
6. Read House_Name, using union variable emp
7. Print House_Name, using union variable emp
8. Read City_Name using union variable emp.
9. Print City_Name using union variable emp.
10. Read State using union variable emp.
11. Print state using union variable emp.
12. Read pin_code using union variable emp.
13. Print pin_code using union variable emp.
14. Stop

4.14 Find the factorial of a given Natural Number n using recursive and non recursive functions

i) Non recursive

ALGORITHM of main()

```
1. Start
2. Read n
3. Call the function fact(n)
4. Stop
```

ALGORITHM of Function fact(x)

```
1. Start
2. z=1
3. If (x != 0) then goto step 2.3 else goto step 2.5
4. z= z*x
5. x=x-1, go to step 2.2
6. Print z
7. Read the number to n
8. Call the function fact(n)
9. Stop
```

ii) Recursive

ALGORITHM for main()

```
1. Start
2. Read n
3. Call the function fact(n)
4. Print Factorial f
5. Stop
```

ALGORITHM of Function fact(x)

```
1. Start
2. If (x==1) then return 1 else goto step 3
3. f=x*fact(x-1)
4. Return f
5. Stop
```

4.15 Read a string (word), store it in an array and obtain its reverse by using a user defined function.

ALGORITHM of main()

```
        1. Start
        2. Call the function reverse()
        5. Stop
```

ALGORITHM of Function reverse

```
        1. Start
        2. Read a word to str
        3. Find length of str to n.
        4. n=n-1
        5. if(n>=0)then Print str[n] else goto step 4
        6. Stop
```

4.16 Write a menu driven program for performing matrix addition, multiplication and finding the transpose. Use functions to (i) read a matrix, (ii) find the sum of two matrices, (iii) find the product of two matrices, (iv) find the transpose of a matrix and (v) display a matrix.

ALGORITHM of main()

```
  1. Start
  2. Declare 2 matrices a,b and their rows and columns(m1, n1, m2, n2) and a choice
     variable op
  3. Read m1, n1
  4. Call the function readmatrix(a,m1,n1)
  5. Read m2, n2
  6. Call the function readmatrix(b,m2,n2)
  7. To print Matrix a, call the function displaymatrix(a,m1,n1);
  8. To print Matrix b, call the function displaymatrix(b,m2,n2);
  9. Read op                    // 1.add 2.multiply 3.transpose 4.exit
  10.if(op==1) then goto step 11 else goto step 12
  11. if(m1==m2 && n1==n2) then call the function addmatrix(a,b,m1,n1) else print
matrix addition is not possible go to step 15
  12. if(op==2) then goto step 13 else goto step 14
```

```
13. if(n1==m2) then call the function **multmatrix(a,b,m1,n1,n2)**, else print "matrix multiplication is not possible" , go to step 15
14. if(op==3) then call the function transpose(a,m1,n1) else print invalid choice.
15.Stop
```

ALGORITHM of read_matrix(matrix[10][10],int row,int col)

```
1. Start
2. i=0,j=0
4. if(i<row)then goto step 5 else goto step 9
5. j=0
6. if(j<col) then read a[i][j],goto step 7 else goto step 8
7. j=j+1,goto step 6
8. i=i+1,goto step 4
9. Stop
```

ALGORITHM of displaymatrix(int a[][100],int m,int n)

```
1. Start
2. i=0,j=0
4. if(i<m)then goto step 5 else goto step 11
5. j=0
6. if(j<n) then print a[i][j],goto step 7 else goto step 9
7. Print a space
8. j=j+1,goto step 6
9. Print a new line
10. i=i+1,goto step 4
11. Stop
```

ALGORITHM of addmatrix(int a[][100],int b[][100],int m,int n)

```
1. Start
2. Declare a matrix c
3. i=0,j=0
4. if(i<m)then goto step 5 else goto step 9
5.  j=0
6. if(j<n) then calculate c[i][j]=a[i][j]+b[i][j],goto step 7 else goto step 8
```

```
7. j=j+1,goto step 6
8. i=i+1,goto step 4
9. Call the function displaymatrix(c,m,n);
10.Stop
```

ALGORITHM of transpose(int a[][100],int m,int n)

```
1. Start
2. Declare a matrix c
3. i=0,j=0
4. if(i<m)then goto step 5 else goto step 9
5. j=0
6. if(j<n) then c[j][i]=a[i][j] ,goto step 7 else goto step 8
7. j=j+1,goto step 6
8. i=i+1,goto step 4
9. Call the function displaymatrix(c,n,m);
10.     Stop
```

ALGORITHM of multmatrix(int a[][100],int b[][100],int m1,int n1,int n2)

```
1. Start
2. Declare a matrix c
3. i=0,j=0
4. if(i<m1)then goto step 5 else goto step 12
5. j=0
6. if(j<n2) then c[j][i]=0 ,goto step 7 else goto step 8
7. k=0
8. if(k<n1) then  c[i][j]=c[i][j]+ a[i][k] * b[k][j],goto step 9 else
   goto step 10
9. k=k+1,goto step 8
10.j=j+1,goto step 6
11.i=i+1,goto step 4
12. Call the function displaymatrix(c,m1,n2)
13.Stop
```

4.17 Do the following using pointers i) add two numbers ii) swap two numbers using a user defined function.

i) Add two numbers

ALGORITHM

```
1. Start
2. Initialize two integer pointers p,q.
3. Read a,b
4. Reference the pointers to variables using '&' operator.//p=&a,q=&b
5. Now, add the values, using * operator//sum=*p+*q
6. Print the sum.
7. Stop
```

ii) Swap two numbers

ALGORITHM

```
1. Start
2. Initialize two integer pointers p,q.
3. Read a,b
4. Reference the pointers to variables using '&' operator.//p=&a,q=&b
5. Now, interchange the values, using * operator//t=*p,*p=*q,*q=t
   Print the swapped values using * operator.//Print *p,*q
6. Stop
```

4.18 Input and Print the elements of an array using pointers

ALGORITHM

```
1. Start
2. Read size of the array to n
3. Initialize one integer pointer *ptr
4. i=0
5. ptr=&a[0]
6. if(i<n) then goto step 7 else goto step 10
7. Read a number to ptr
8. ptr=ptr+1
9. i=i+1,goto step 6
```

```
10.   i=0
11.   ptr=&a[0]
12.   if(i<n) then goto step 13 else goto step 16
13.   Print a number using pointer ptr
14.   ptr=ptr+1
15.   i=i+1,goto step 12
16.   Stop
```

4.19 Compute sum of the elements stored in an array using pointers and user defined functions.

ALGORITHM for main()

```
1. Start
2. Read size of the array to n
3. Initialize one integer pointer *ptr
4. i=0
5. ptr=&a[0]
6. if(i<n) then goto step 7 else goto step 10
7. Read a number to ptr
8. ptr=ptr+1
9. i=i+1,goto step 6
10. Call the function sum(&a,n)
11.Stop
```

ALGORITHM for function sum(int *ptr, int n)

```
1. Start
2. i=0,s=0
3. if(i<n) then goto step 4 else goto step 7
4. s=s+*ptr
5. ptr=ptr+1
6. i=i+1,goto step 3
7. Print s
8. Stop
```

4.20 Create a file and perform the following

i) Write data to the file

ii) Read the data in a given file & display the file content on console

iii) Append new data and display on console

i) Write data to the file

```
1. Start
2. Create a file pointer *fp
3. Open a new file in write mode using file pointer fp
   //fp=fopen(“a.txt”,”w”)
4. if(fp==NULL) then print Error opening file else goto step 5
5. Read ch
6. if (ch!='EOF') then goto step 7 else goto step 8
7. Write ch to file a.txt,goto step 5
8. Close the file
9. Stop
```

ii) Read the data in a given file & display the file content on console

```
1. Start
2. Create a file pointer *fp
3. Open an existing file in read mode using file pointer fp
   //fp=fopen(“a.txt”,”r”)
4. if(fp==NULL) then print Error opening file else goto step 5
5. Read the content from file a.txt//ch=getc(fp)
6. if (ch!='EOF') then goto step 7 else goto step 8
7. Print the content from the file a.txt, goto step 5  //putchar(ch);
8. Close the file
9. Stop
```

iii) Append new data and display on console

```
1. Start
2. Create a file pointer *fp
```

3. Open an existing file in append mode using file pointer fp //fp=fopen(“a.txt”,”a+”) //**a+** : append data in a file and update it, which means it can write at the end and also is able to read the file
4. if(fp==NULL) then print Error opening file else goto step 5
5. Read the content to be append to ch
6. if (ch!='EOF') then goto step 7 else goto step 8
7. Write ch to the end of the file a.txt,goto step 5
8. Close the file
9. Stop

4.21 Open a text input file and count the number of characters, words and lines in it; and store the results in an output file.

1. Start
2. Create a file pointer *fp
3. c=o,w=0,l=0
4. Open an existing file in read mode using file pointer fp //fp=fopen(“a.txt”,”r”)
5. if(fp==NULL) then print Error opening file else goto step 5
6. Read the content from file a.txt to ch//ch=getc(fp)
7. if (ch!='EOF') then goto step 7 else goto step 10
8. if(ch==’\n’) then l=l+1 else goto step 9
9. if(ch==’ ’) then w=w+1 else c=c+1,goto step 6
10. Print c,w,l
11. Close the file
12. Stop

5. REFERENCES

4.1 (i) Display Hello World

PROGRAM

```
#include<stdio.h>
main()
{
printf("Hello World!!");
}
```

OUTPUT

```
Hello World!!
```

(ii) Add Two Numbers

PROGRAM

```
#include<stdio.h>
main()
{
int a,b,c;
printf("Enter First Number:");
scanf("%d",&a);
printf("Enter Second Number:");
scanf("%d",&b);
c=a+b;
printf("Sum of above numbers is:%d",c);
}
```

OUTPUT

```
Enter First Number:34
Enter Second Number:56
Sum of above numbers is:90
```

(iii) Area Of Circle

PROGRAM

```
#include<stdio.h>
# define PI 3.14
main()
{
```

```
float  r,Area;
printf("Enter the radius:");
scanf("%f",&r);
Area=PI*r*r;
printf("Area of Circle:%f",Area);
}
```

OUTPUT

```
Enter the radius:3
Area of Circle:28.260000
```

(iv)Evaluate Arithmetic Expression

```
#include<stdio.h>
main()
{
int a,b,c,d,e,f,g,Ans;
printf("Enter 7 values for the variables:");
scanf("%d%d%d%d%d%d%d",&a,&b,&c,&d,&e,&f,&g);
Ans=((a-b/c*d+e)*(f+g));
printf("Solutin=%d",Ans);
}
```

OUTPUT

```
Enter 7 values for the variables:1 2 3 4 5 6 7
Solutin=78
```

4. 2 Find the Largest of Three Numbers

PROGRAM

```
#include<stdio.h>
main()
{
int n1, n2, n3;
    printf("Enter three different numbers: ");
    scanf("%d%d%d", &n1, &n2, &n3);
    if (n1 >= n2 && n1 >= n3)
        printf("%d is the largest number.", n1);
    if (n2 >= n1 && n2 >= n3)
        printf("%d is the largest number.", n2);
    else
        printf("%d is the largest number.", n3);
}
```

OUTPUT

```
Enter three different numbers:
```

```
34
3
78
78 is the largest number.
```

4.3 PRIME OR NOT

PROGRAM

```
#include <stdio.h>
main()
 {
    int n, i, flag = 0;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    for (i = 2; i <= n / 2; ++i)
        {
        if (n % i == 0)
       {
            flag = 1;
            break;
        }
    }
    if (n == 1)
    {
        printf("1 is neither prime nor composite.");
    }
    else
    {
        if (flag == 0)
            printf("%d is a prime number.", n);
        else
            printf("%d is not a prime number.", n);
    }
```

OUTPUT

```
Enter a positive integer: 7
34 is a prime number.
 ./a.out
Enter a positive integer: 78
78 is not a prime number.
```

4.4 AMSTRONG OR NOT

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

```
{
        int num, S, remainder, result = 0;
        printf("Enter a three-digit integer: ");
        scanf("%d", &num);
        S = num;
        while (S != 0)
        {
            remainder = S % 10;
            result += remainder * remainder * remainder;
            S /= 10;
        }
        if (result == num)
            printf("%d is an Armstrong number.", num);
        else
            printf("%d is not an Armstrong number.", num);

    }
```

```
OUTPUT
Enter a three-digit integer: 153
153 is an Armstrong number.
 ./a.out
Enter a three-digit integer: 346
346 is not an Armstrong number.
```

4.5 ARRAY SUM AND AVERAGE

```
#include <stdio.h>
int main()
        {
        int n, i;
        float num[100], sum = 0.0, avg;
        printf("Enter the numbers of elements: ");
        scanf("%d", &n);
        while (n > 100 || n < 1)
        {
            printf("Error! number should in range of (1 to 100).\n");
            printf("Enter the number again: ");
            scanf("%d", &n);
        }
        for (i = 0; i < n; ++i) {
            printf("%d. Enter number: ", i + 1);
            scanf("%f", &num[i]);
            sum += num[i];
        }
        avg = sum / n;
        printf("Sum=%.2f\n",sum);
        printf("Average = %.2f", avg);
        return 0;
    }
```

OUTPUT

```
Enter the numbers of elements: 5
1. Enter number: 1
2. Enter number: 2
3. Enter number: 3
4. Enter number: 4
5. Enter number: 5
Sum=15.00
Average = 3.00
```

4.6 LINEAR SEARCH

PROGRAM

```c
#include<stdio.h>
main()
{
  int array[100], search, c, n;
  printf("Enter number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  printf("Enter a number to search\n");
  scanf("%d", &search);

  for (c = 0; c < n; c++)
  {
    if (array[c] == search)
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);
}
```

OUTPUT

```
Enter number of elements in array
6
Enter 6 integer(s)
23
4
56
7
```

```
8
12
Enter a number to search
7
7 is present at location 4.
```

4.7 BUBBLE SORT

PROGRAM

```
#include <stdio.h>
main()
{
int a[25],n;
int i,j,t;
printf("enter the size of array");
scanf("%d",&n);
printf("enter the elements are");
for(i=0;i<n;i++)
{
	scanf("%d",&a[i]);
}
for(i=0;i<n-1;i++)
{
	for(j=0;j<n-i-1;j++)
	{
		if(a[j]>a[j+1])
		{
			t=a[j];
			a[j]=a[j+1];
			a[j+1]=t;
		}
	}
}

printf("sorted array is:");
for(i=0;i<n;i++)
{
	printf("%d\t",a[i]);
}
}
```
OUTPUT
```
enter the size of array5
enter the elements are
34
2
780
33
```

```
45
sorted array is:2       33      34      45      780
```

4.8 PALINDROME OR NOT

PROGRAM

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

main()
{
    char string1[20];
    int i, length;
    int flag = 0;

    printf("Enter a string:");
    scanf("%s", string1);

    length = strlen(string1);

    for(i=0;i < length ;i++)
   {
        if(string1[i] != string1[length-i-1]){
            flag = 1;
            break;
   }
}

    if (flag) {
        printf("%s is not a palindrome", string1);
    }
    else {
        printf("%s is a palindrome", string1);
    }

}
```

OUTPUT

```
Enter a string:malayalam
malayalam is a palindrome
 ./a.out
Enter a string:great
great is not a palindrome
```

4.9 STRING CONCATENATION WITHOUT USING LIBRARY FUNCTION

```
#include<stdio.h>
main()
{
        char Str1[100], Str2[100];
        int i, j;

        printf("\n Please Enter the First String :(end with $!!)");
        gets(Str1);

        printf("\n Please Enter the Second String :(end with $!!)");
        gets(Str2);
        for (i = 0; Str1[i]!='$'; i++);

                for (j = 0; Str2[j]!='$'; j++, i++)
                {
                        Str1[i] = Str2[j];
                }
                        Str1[i] = '\0';
                printf("\n String after the Concatenate = %s", Str1);
}
```

OUTPUT

```
Please Enter the First String :(end with $!!)best$

 Please Enter the Second String :(end with $!!)wishes$

 String after the Concatenate = bestwishes
```

4.10 COUNT VOWELS AND CONSONANTS

PROGRAM

```
#include <stdio.h>
main()
{
        char line[150];
        int vowels, consonant, space,i;
        vowels = consonant = space = 0;
        printf("Enter line of string:(end with $)");
        gets(line);
        for (i = 0; line[i] != '$'; ++i)
        {
            if (line[i] == 'a' || line[i] == 'e' || line[i] == 'i' ||
                line[i] == 'o' || line[i] == 'u' || line[i] == 'A' ||
                line[i] == 'E' || line[i] == 'I' || line[i] == 'O' ||
                line[i] == 'U')
            {
                ++vowels;
            }
           else if ((line[i] >= 'a' && line[i] <= 'z') || (line[i] >= 'A' && line[i] <= 'Z'))
            {
```

```
            ++consonant;
        }
        else if (line[i] == ' ')
        {
            ++space;
        }
    }
    printf("Vowels: %d", vowels);
    printf("\nConsonants: %d", consonant);
    printf("\nWhite spaces: %d", space);

}
```

OUTPUT

```
Enter line of string:(end with $)india is my country$
Vowels: 6
Consonants: 10
White spaces: 3
```

4.11 DISTANCE BETWEEN TWO POINTS

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

int main() {
float x1, y1, x2, y2, gdistance;
printf("Input x1: ");
scanf("%f", &x1);
printf("Input y1: ");
scanf("%f", &y1);
        printf("Input x2: ");
scanf("%f", &x2);
printf("Input y2: ");
scanf("%f", &y2);
gdistance = ((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1));
printf("Distance between the said points: %.4f", sqrt(gdistance));
printf("\n");
return 0;
}
```

OUTPUT

```
Input x1: 3
Input y1: 6
Input x2: -2
Input y2: 4
Distance between the said points: 5.3852
```

4.12 DATA OF EMPLOYEE USING STRUCTURE

PROGRAM

```
#include <stdio.h>


struct employee
{
    char    name[30];
    int     empId;
    float   salary;
};

main()
{

    struct employee emp;

    printf("\nEnter details :\n");
    printf("Name ?:");          gets(emp.name);
    printf("ID ?:");            scanf("%d",&emp.empId);
    printf("Salary ?:");        scanf("%f",&emp.salary);


    printf("\nEntered detail is:\n ------------------ ");
    printf("\nName: %s"    ,emp.name);
    printf("\nId: %d"      ,emp.empId);
    printf("\nSalary: %f\n",emp.salary);

}
```

OUTPUT

```
Enter details :
Name ?:Amrutha
ID ?:349
Salary ?:23000

Entered detail is:
-------------------
Name: Amrutha
Id: 349
Salary: 23000.000000
```

4.13 UNION

PROGRAM

```
#include <stdio.h>
#include <string.h>
#define C_SIZE 50
union Address
{
            char name[C_SIZE];
            char hname[C_SIZE];
            char cityname[C_SIZE];
        char state[C_SIZE];
        char pin[C_SIZE];
};

int main()
{
    union Address record1;

    printf("Enter name:");
    scanf("%s",record1.name);
    getchar();
    printf("Enter house name:");
    scanf("%s",record1.hname);
    getchar();
    printf("Enter city name:");
    scanf("%s",record1.cityname);
    getchar();
    printf("Enter state name:");
    scanf("%s",record1.state);
    getchar();
    printf("Enter pin:");
    scanf("%s",record1.pin);
    printf("Union record1 values .... \n");
    printf(" Name          : %s \n", record1.name);
    printf(" House Name    : %s \n", record1.hname);
    printf(" City Name     : %s \n", record1.cityname);
    printf(" State name    : %s \n", record1.state);
    printf(" Pin           : %s \n", record1.pin);
 }
```

OUTPUT

Note: it is noted that the program will print only Pin because the union will hold only one value at a time .

4.14 FACTORIAL OF A NUMBER USING RECURSIVE AND NON-RECURSIVE FUNCTIONS

PROGRAM

```
#include <stdio.h>
long int factnr(int n)
{ int i;
  long int f=1;
  for(i=1;i<=n;i++)
       f=f*i;
  return f;
}
long int factr(int n)
{
  if(n==0) return 1;
  else
  return (n*factr(n-1));
}
int main()
{int n;
  system("clear");
  printf("Enter the number \n");
  scanf("%d",&n);
  printf("Factorial using non recursive function %d !=%ld\n",n,factnr(n));
  printf("Factorial using     recursive function %d !=%ld\n",n,factr(n));
}
```

4.15 REVERSE A STRING USING FUNCTION

PROGRAM

```
#include <stdio.h>
#include <string.h>
void reversestr(char str[])
{ int i,n;
  char c;
  n=strlen(str);
  for(i=0;i<n/2;i++)
    {   c=str[i];
         str[i]=str[n-1-i];
         str[n-1-i]=c;
     }
}
int main()
{
  char str[100];
  system("clear");
  printf("Enter the string \n");
  scanf("%[^\n]",str);
  reversestr(str);
  printf("Reversed string is=%s\n",str);
}
```

OUTPUT

```
Enter the string
smart
Reversed string is=trams
```

4.16 MATRIX ADDITION, MULTIPLICATION AND TRANSPOSE

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
void readmatrix(int a[][100],int m,int n)
{
 int i,j;
 printf("enter the elements row by row\n");
 for(i=0;i<m;i++)
   for(j=0;j<n;j++)
    scanf("%d",&a[i][j]);
}
void displaymatrix(int a[][100],int m,int n)
{
 int i,j;
 for(i=0;i<m;i++)
 {
  for(j=0;j<n;j++)
    printf("%5d",a[i][j]);
  printf("\n");
  }
}
void addmatrix(int a[][100],int b[][100],int m,int n)
{
 int i,j,c[100][100];
 for(i=0;i<m;i++)
   for(j=0;j<n;j++)
    c[i][j]=a[i][j]+b[i][j];
 printf("Sum of matrix...\n");
 displaymatrix(c,m,n);
}
void transpose(int a[][100],int m,int n)
{
 int i,j,c[100][100];
 for(i=0;i<m;i++)
   for(j=0;j<n;j++)
```

```
    c[j][i]=a[i][j];

 displaymatrix(c,n,m);
}
void multmatrix(int a[][100],int b[][100],int m1,int n1,int n2)
{
 int c[100][100],i,j,k;
    for (i = 0; i < m1; i++) {
        for (j = 0; j < n2; j++) {
            c[i][j] = 0;
            for (k = 0; k < n1; k++)
                c[i][j] += a[i][k] * b[k][j];
        }
    }
 printf("Product of matrix...\n");
 displaymatrix(c,m1,n2);
}
int main()
{ int a[100][100],b[100][100],m1,n1,m2,n2,op;
  system("clear");
  printf("Enter the size of the matrix A row,column\n");
  scanf("%d%d",&m1,&n1);
  printf("Enter Matrix A\n");
  readmatrix(a,m1,n1);
  printf("Enter the size of the matrix B row column\n");
  scanf("%d%d",&m2,&n2);
  printf("Enter Matrix B\n");
  readmatrix(b,m2,n2);
  system("clear");
  printf("Matrix A..\n");
  displaymatrix(a,m1,n1);
  printf("Matrix B..\n");
  displaymatrix(b,m2,n2);
  while(1)
  {
   printf("\n************************************\n");
   printf("1.add 2.multiply 3.transpose 4.exit \n");
   printf("Enter the option..... ");
   scanf("%d",&op);
switch(op)
   {
    case 1: if(m1==m2 && n1==n2)
                addmatrix(a,b,m1,n1);
            else
                printf("Incompatable matrix...cannot add..\n");
             break;
    case 2: if(n1==m2)
                multmatrix(a,b,m1,n1,n2);
            else
               printf("Incompatable matrix...cannot mutliply..\n");
             break;
    case 3: printf("Transpose of A..\n");
             transpose(a,m1,n1);
             printf("Transpose of B..\n");
             transpose(b,m2,n2);
             break;
    case 4: exit(0);
   }
   }
}
```

OUTPUT

```
Enter the size of the matrix A row,column
3 3
Enter Matrix A
enter the elements row by row
1 2 3
1 2 3
1 2 3
Enter the size of the matrix B row column
3
3
Enter Matrix B
enter the elements row by row
1 2 3
1 2 3
1 2 3
Matrix A..
   1   2   3
   1   2   3
   1   2   3
Matrix B..
   1   2   3
   1   2   3
   1   2   3

***********************************
1.add 2.multiply 3.transpose 4.exit
Enter the option.....:1
Sum of matrix...
   2   4   6
   2   4   6
   2   4   6

***********************************
```

```
1.add 2.multiply 3.transpose 4.exit
Enter the option. ....2
Product of matrix...
    6   12   18
    6   12   18
    6   12   18

************************************
1.add 2.multiply 3.transpose 4.exit
Enter the option.....:3
Transpose of A..
    1    1    1
    2    2    2
    3    3    3
Transpose of B..
    1    1    1
    2    2    2
    3    3    3

************************************
1.add 2.multiply 3.transpose 4.exit
Enter the option.....:4
```

4.17 POINTERS

(i) ADD TWO NUMBERS

PROGRAM

```
#include <stdio.h>
int main()
   {
      int first, second, *p, *q, sum;
      printf("Enter two integers to add\n");
      scanf("%d%d", &first, &second);
      p = &first;
      q = &second;
      sum = *p + *q;
      printf("Sum of the numbers = %d\n", sum);
    }
```

OUTPUT

```
Enter two integers to add
34
```

```
9
Sum of the numbers = 43
```

(ii)SWAP TWO NUMBERS

PROGRAM

```
#include <stdio.h>
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
int main()
{
    int x, y;
    printf("Enter Value of x ");
    scanf("%d", &x);
    printf("\nEnter Value of y ");
    scanf("%d", &y);
    swap(&x, &y);
    printf("\nAfter Swapping: x = %d, y = %d", x, y);
    return 0;
}
```

OUTPUT

```
Enter Value of x 45

Enter Value of y 7

After Swapping: x = 7, y = 45
```

4.18 INPUT AND PRINT ARRAY USING POINTERS

PROGRAM

```
#include <stdio.h>
int main()
{
    int arr[100];
    int n, i;
    int * ptr = arr;
    printf("Enter size of array: ");
    scanf("%d", &n);

    printf("Enter elements in array:\n");
    for (i = 0; i < n; i++)
    {       scanf("%d", (ptr + i));
    }

    printf("Array elements: \n");
    for (i = 0; i < n; i++)
    {       printf("%d\n", *(ptr + i));
    }
  }
```

OUTPUT

```
Enter size of array: 5
Enter elements in array:
23
4
5
7
9
Array elements:
23
4
5
7
9
```

4.19 COMPUTE SUM OF ARRAY USING POINTERS AND FUNCTIONS

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
int arraysum(int *ptr,int n)
{
 int sum=0,i;
for (i = 0; i < n; i++)
    {       sum=sum+ *(ptr + i);
    }
 return sum;
}
int main()
{
    int arr[]={4,5,6,7,8,9,10,1,2,3};
    int sum;
    sum=arraysum(arr,10);
    printf("Array elements sum=:%d \n",sum);
}
```

OUTPUT

```
Array elements sum=:55
```

4.20 FILE OPERATIONS

(i) Write Data to file

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
FILE *fp;
fp=fopen("a.txt","w");
if (fp==NULL)
  {
    printf("error opening file..\n");
    exit(1);
   }
else
  {
   fprintf(fp,"%s","Welcome\n");
   fprintf(fp,"%s","to file handling in C\n");
   }
printf("File Created...named a.txt");
fclose(fp);
}
```

OUTPUT

```
File Created...named a.txt
```

(ii) Read data from file and print content on console

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
FILE *fp;
char t[100];
fp=fopen("a.txt","r");
if(fp==NULL)
  {
  printf("Error opening source file..");
  exit(1);
   }
printf("Content of File a.txt\n.................. \n");
while(fscanf(fp,"%s",t)==1)
{
printf("%s\n",t);
}
fclose(fp);
}
```

OUTPUT

```
Content of File a.txt
..................
Welcome
to
file
handling
in
C
```

(iii) Append a file and display content on console

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
FILE *fp;
char t[100];
fp=fopen("a.txt","a");
if(fp==NULL)
  {
  printf("Error opening source file..");
  exit(1);
   }
printf("Enter the contents to append. ............... \n");
while(1)
{
 fgets(t,sizeof(t),stdin);
 if(strcmp(t,"end\n")==0) break;
 fputs(t,fp);
}
fclose(fp);
fp=fopen("a.txt","r");
printf("File contents after appending.  \n");
printf("*********************************\n");
while(fgets(t,sizeof(t),fp)!=NULL)
{
printf("%s",t);
}
fclose(fp);
}
```

OUTPUT

```
Enter the contents to append
.................
mea engg college
```

4.21 OPEN A FILE, COUNT CHARACTER, WORDS AND LINES IN IT

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
FILE *fp;
char fname[50];
int ch;
int nl=0,nc=0,nw=0;
printf("Enter the file name. ..\n");
scanf("%[^\n]",fname);
fp=fopen(fname,"r");
if(fp==NULL)
  {
  printf("Error opening file..");
  exit(1);
   }
ch=getc(fp);
while(ch!=EOF)
{
if (ch=='\n') nl++;
if(ch==' ') nw++;
nc++;
ch=getc(fp);
}
fclose(fp);
printf("Number of lines=%d Number of words=%d ,Number of characters =
%d,\n",nl,nw,nc+nl);
printf("results are written into result.dat file..\n");
fp=fopen("result.dat","w");
fprintf(fp,"Number of lines=%d Number of words=%d ,Number of characters =
%d,\n",nl,nw,nc+nl);
fclose(fp);
}
```

OUTPUT

```
Enter the file name....
a.txt
Number of lines=2 Number of words=4 ,Number of characters = 32,
results are written into result.dat file..
```

****