(2½ Hours) **[Total Marks: 75**

N. B.: (1) **All** questions are **compulsory**.
(2) Make **suitable assumptions** wherever necessary and **state the assumptions** made.
(3) Answers to the **same question** must be **written together**.
(4) Numbers to the **right** indicate **marks**.
(5) Draw **neat labeled diagrams** wherever **necessary**.
(6) Use of **Non-programmable** calculators is **allowed**.

1.	**Attempt *any three* of the following:**		15
a.	What is the difference between machine level language and high level language ?		
	MachineLunguage	High-level language	
	A collection of very detailed, cryptic instructions that control the computer's internal circuitry	Whose instruction set is more Compatible with human languages and human thought processes	
	Machine language is very cumbersome to work	Greatly simplifies the task of writing complete, correct programs.	
	Use multiple instruction to execute an single task	A single instruction in a high-level language will be equivalent to several instructions in Machine language	
	Because every different type of computer has its own unique instruction set. Thus, a machine-language Program written for one type of computer cannot be run on another type of computer without significant Alterations.	A particular high-level language are much the same for all computers, so that a Program written for one computer can generally be run on many different computers with little or no alteration	
	A program written in machine level language uses assemblers	A program that is written in a high-level language must, however, be translated into machine language Before it can be executed. This is known as compilation or interpretation	
	Advantage : fast as it requires no translation, . Uses limited space in RAM	Advantage : simplicity, uniformity and portability	
b.	Describe the structure of a C Program.		
	Structure of a C Program Every C program consists of one or more modules calledfunctions. One of the functions must be called main. The program will always begin by executing the main function, which may access other functions. Any other function definitions must be defined separately, either ahead of or after main Each function must contain: 1. A function heading, which consists of the function name, followed by an optional list of arguments, enclosed in parentheses. 2. A list of argument declarations, if arguments are included in the heading. 3. A compound statement, which comprises the remainder of the function. The arguments are symbols that represent information being passed between the function and other parts of the program. (Arguments are also referred to as		

	parameters.) Each compound statement is enclosed within a pair of braces, i.e., { }. The braces may contain one or more elementary statements (called expression statements) and other compound statements. Thus compound statements may be nested, one within another. Each expression statement must end with a semicolon (;).Comments (remarks) may appear anywhere within a program, as long as they are placed within the delimiters / * and */ (e.g., /* t h i s is a comment */). Such comments are helpful in identifying the program's principal features or in explaining the underlying logic of various program features. /* program to calculate the area of a c i r c l e */ /* TITLE (COMMENT) */ #include <stdio.h> / * LIBRARY FILE ACCESS */ main( ) / * FUNCTION HEADING */ f l o a t radius, area; /* VARIABLE DECLARATIONS */ p r i n t f ("Radius = ? / * OUTPUT STATEMENT (PROMPT) * I ' I) ; scanf ("%'fI , &radius) ; I * INPUT STATEMENT * / area = 3.14159 * radius * radius; /* ASSIGNMENT STATEMENT */ p r i n t f ("Area = %f",area) ; / * OUTPUT STATEMENT */ 1	
c.	List and explain five desirable program characteristics.	
	DESIRABLE PROGRAM CHARACTERISTICS 1. Integrity. This refers to the accuracy of the calculations. It should be clear that all other program enhancements will be meaningless if the calculations are not carried out correctly. Thus, the integrity of the calculations is an absolute necessity in any computer program. 2. Clarity refers to the overall readability of the program, with particular emphasis on its underlying logic. If a program is clearly written, it should be possible for another programmer to follow the program logic without undue effort. It should also be possible for the original author to follow his or her own program after being away from the program for an extended period of time. One of the objectives in the design of C is the development of clear, readable programs through an orderly and disciplined approach to programming. 3. Simplicity. The clarity and accuracy of a program are usually enhanced by keeping things as simple as possible, consistent with the overall program objectives. In fact, it may be desirable to sacrifice a certain amount of computational efficiency in order to maintain a relatively simple, straightforward program structure. 4. Efficiency is concerned with execution speed and efficient memory utilization. These are generally important goals, though they should not be obtained at the expense of clarity or simplicity. Many complex programs require a tradeoffs between these characteristics. In such situations, experience and common sense are key factors. 5. Modularity. Many programs can be broken down into a series of identifiable subtasks. It is good programming practice to implement each of these subtasks as a separate program module. In C, such modules are written as functions. The use of a modular programming structure enhances the accuracy and clarity of a program, and it facilitates future program alterations. 6. Generality. Usually we will want a program to be as general as possible, within reasonable limits. For example, we may design a program to read in the values of certain key parameters rather than placing fixed values into the program. As a rule, a considerable amount of generality can be obtained with very little additional programming effort.	
d.	Explain the following with example i) Symbolic Constants SYMBOLIC CONSTANTS	

A symbolic constant is a name that substitutes for a sequence of characters. The characters may represent a numeric constant, a character constant or a string constant. Thus, a symbolic constant allows a name to appear in place of a numeric constant, a character constant or a string. When a program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence. Symbolic constants are usually defined at the beginning of a program. The symbolic constants may then appear later in the program in place of the numeric constants, character constants, etc. that the symbolic constants represent.
A symbolic constant is defined by writing

```
#define name  text
```

where name represents a symbolic name, typically written in uppercase letters, and text represents the sequence of characters that is associated with the symbolic name. Note that text does not end with a semicolon, since a symbolic constant definition is not a true C statement. Moreover, if text were to end with a semicolon, this semicolon would be treated as though it were a part of the numeric constant, character constant or string constant that is substituted for the symbolic name.

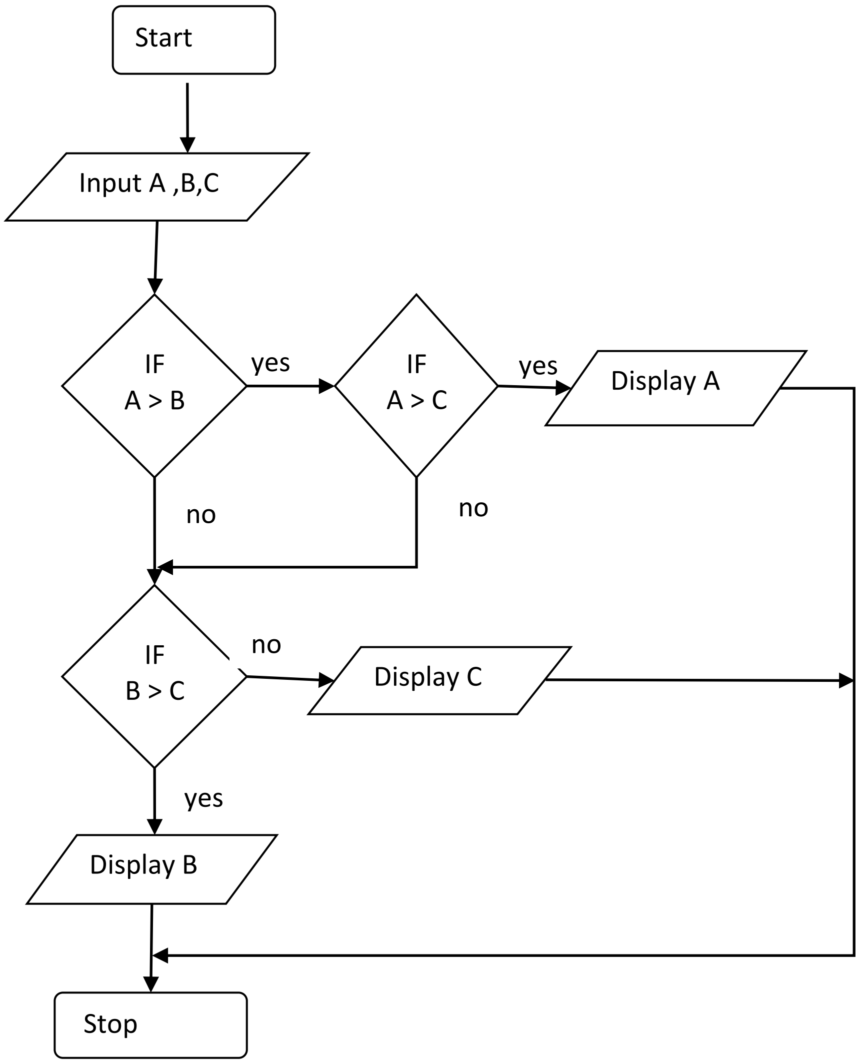
```
#define TAXRATE 0.23
#define PI 3.141593
#define TRUE  1
#define FALSE 0
#define FRIEND "Susan"
```

ii) Escape Sequences

Escape Sequences
Certain nonprinting characters, as well as the backslash (\) and the apostrophe (I), can be expressed in terms of escape sequences. An escape sequence always begins with a backward slash and is followed by one or more special characters. For example, a line feed (LF), which is referred to as a newline in C, can be represented as \n. Such escape sequences always represent single characters, even though they are written in terms of two or more characters. The commonly used escape sequences are listed below

Character	Escape Sequence	ASCII Value
bell (alert)	\a	007
backspace	\b	008
horizontal tab	\t	009
vertical tab	\v	011
newline (line feed)	\n	010
form feed	\f	012
carriage return	\r	013
quotation mark (")	\”	034
apostrophe (')	\’	039
question mark (?)	\?	063
backslash(\)	\ \	092
null	\O	000

An escape sequence can also be expressed in terms of one, two or three octal digits which represent single-character bit patterns. The general form of such an escape sequence is \ooo, where each o represents an octal digit (0through 7). Some versions of C also allow an escape sequence to be expressed in terms of one or more hexadecimal digits, preceded by the letter x. The general form of a hexadecimal escape sequences \xhh, where each h represents a hexadecimal digit (0 through 9 and a through f). The letters can be either upper- or lowercase. The use of an octal or hexadecimal escape sequence is usually less desirable than writing
the character constant directly, however, since the bit patterns may be dependent upon some particular character set.

e.	Draw a flowchart to find the largest of three numbers.  Start Input A ,B,C IF A > B yes IF A > C yes Display A no no IF B > C no Display C yes Display B Stop	
f.	Determine if the following constants are valid in C i) 27,822 invalid → illegal character (,) ii) 0.8E 8 invalid →Illegal character (blank space) iii) “Name: invalid →Trailing quotation mark is missing. iv) “1.3e-12” Valid v) 0xBCFDAL invalid → illegal character (L)	
2.	Attempt <u>any three</u> of the following:	**15**

a.	Write a program in C to finds the Roots of a Quadratic Equation.
	
	/* r e a l roots of a quadratic equation */
	#include <stdio. h>
	#include <math. h>
	main( )
	{
	float a, b, c, d, xl , x2; /* read input data */
	printf("a=");
	scanf("%I'f”,&a);
	printf("b=");	

```
scanf("%I'f",&b);
printf("c=");
scanf("%I'f",&c);
/ * carry out the calculations */
 d = sqrt( b*b  -  4*a*c);
 xl = ( - b + d) / (2 * a);
x2= ( - b - d) / (2 * a);
/ * display the output */
printf(" \ n x l = %e      x2 = %e", x l , x2);
}
```

b. Explain the conditional operator and assignment operators in C with example.

THE CONDITIONALOPERATOR
Simple conditional operations can be carried out with the conditional operator (?:). An expression that makes use of the conditional operator is called a conditional expression.
A conditional expression is written in the form
expression 1 ? expression 2 : expression 3
When evaluating a conditional expression, expression I is evaluated first. If expression 1 is true(i.e., if its value is nonzero), then expression 2 is evaluated and this becomes the value of the conditional expression. However, if expression 1 is false (i.e., if its value is zero), then expression 3 is evaluated and this becomes the value of the conditional expression. Note that only one of the embedded expressions(either expression 2 or expression 3) is evaluated when determining the value of a conditional expression.
Example
In the conditional expression shown below, assume that i is an integer variable.
(i < 0) ? 0 : 100
The expression (i < 0) is evaluated first. If it is true (i.e., if the value of i is less than 0), the entire conditional expression takes on the value 0. Otherwise (if the value of i is not less than 0),the entire conditional expression takes on the value 100.
In the following conditional expression, assume that f and g are floating-point variables.
(f < g) ? f : g
This conditional expression takes on the value off if f is less than g; otherwise, the conditional expression takes on the value of g. In other words, the conditional expression returns the value of the smaller of the two variables. If the operands (i.e., expression 2 and expression 3)differ in type, then the resulting data type of the conditional expression will be determined by the rules of conversion.

Assignment operators
There are several different assignment operators in C. All of them are used to form assignment expressions,which assign the value of an expression to an identifier.
The most commonly used assignment operator is =. Assignment expressions that make use of this operator are written in the form
identifier = expression
where identifier generally represents a variable, and expression represents a constant, a variable or a more complex expression.
x = y
delta = 0.001
sum = a + b
C contains the following five additional assignment operators: +=, -= , *=, /= and %=. To see how they are used, consider the first operator, +=. The assignment expression
expression 1 += expression 2

is equivalent to
expression 1 = expression 1 + expression 2
Similarly, the assignment expression
expression 1 -= expression 2
is equivalent to
expression 1 = expression 1 - expression 2
and so on for all five operators.
Usually, expression 1 is an identifier, such as a variable or an array element.
Similarly *= , /= , %=

a += 8	a = a+8
f -=g	f = f - g
j *= (i - 3)	j = j * (i - 3)
f I= 3	f = f / 3
i %= (j - 2)	i = i % (j - 2)

c. Interpret the following C statements

i) scanf("%8d %*d %121f %121f",&a,&b,&c,&d); [2] [1] [2]

a will be assigned a decimal integer with a maximum field-width of 8; another decimal integer will then be
read into the computer but not assigned; c and d will then be assigned double-precision quantities with
maximum field-widths of 12.

ii) scanf("%c %c %c",&a,&b,&c);
a, b, c, will be assigned a single character indivisually

iii) scanf("%d %d %f %f",&a,&b,&c,&d);
a,b will be assigned intergers with c and d with double-precision quantities

d. Write a program in C to solve the following expression F = P(1+i)^{n}

```
#include<stdio.h>
#include<math.h>
 void main()
{
float p,i;
double f;
int i;
printf(" enter values for i, p and n ");
scanf ("%f%f%%d",&i,&p,&n);
f  = p* pow( 1+i, n);
printf("%lf",f);
}
```

e. What is a relational expression? List all operators used with it.
There are four relational operators in C.
< less than
<= less than or equal to
> greater than
>= greater than or equal to
These operators all fall within the same precedence group, which is lower than the arithmetic and unary operators. The associativity of these operators is left to right.
Closely associated with the relational operators are the following two equality operators,
== equal to
!= not equal to
The equality operators fall into a separate precedence group, beneath the relational operators. These operators also have a left-to-right associativity.
These six operators are used to form logical expressions, which represent conditions that are either true or false. The resulting expressions will be of type integer, since

true is represented by the integer value 1 and false is represented by the value 0.

f. Explain gets and printf statements used in C programming language.
gets statement :- facilitate the input of strings.
The gets functions, which facilitate the transfer of strings between the computer and the standard input devices.
Each of these functions accepts a single argument. The argument must be a data item that represents a string. (e.g., a character array). The string may include whitespace characters. In the case of gets, the string will be entered from the keyboard, and will terminate with a newline character (i.e., the string will end when the user presses the Enter key).

```
#include <stdio.h>
{
main( ) / * read and write text * /
char  line[80] ;
gets(line);
puts(line);
}
```

Printf statement
Output data can be written from the computer onto a standard output device using the library function
p r i n t f . This function can be used to output any combination of numerical values, single characters and
strings. It is similar to the input function scanf, except that its purpose is to display data rather than to enter it
into the computer. That is, the p r i n t f function moves data from the computer's memory to the standard
output device, whereas the scanf function enters data from the standard input device and stores it in the
computer's memory.
In general terms, the p r i n t f function is written as
printf(control string, arg7, arg2, . . . , argn)
where control stringrefers to a string that contains formatting information, and arg7, arg2, . . . ,
argn are arguments that represent the individual output data items. The arguments can be written as constants, single variable or array names, or more complex expressions. Function references may also be included. the arguments in a p r i n t f function do not represent memory addresses and therefore are not preceded by ampersands. The control string consists of individual groups of characters, with one character group for each output data item. Each character group must begin with a percent sign (%). In its simplest form, an individual character group will consist of the percent sign, followed by a conversion character indicating the type of the corresponding data item.
Multiple character groups can be contiguous, or they can be separated by other characters, including white space characters. These "other" characters are simply transferred directly to the output device, where they are displayed. The use of blank spaces as character-group separators is particularly common.

C	Data item is displayed as a single character
D	Data item is displayed as a signed decimal integer
e	Data item is displayed as a floating-point value with an exponent
f	Data item is displayed as a floating-point value without an exponent
g	Data item is displayed as a floating-point value using either e-type or f-type conversion, depending on value. Trailing zeros and trailing decimal point will not be displayed.
i	Data item is displayed as a signed decimal integer
0	Data item is displayed as an octal integer, without a leading zero

S	Data item is displayed as a string
U	Data item is displayed as an unsigned decimal integer
X	Data item is displayed as a hexadecimal integer, without the leading Ox

3. Attempt <u>any three</u> of the following: **15**

a. Write the syntax of the if-else statement in C. What are nested ‘ if ‘ statements ?

THE if - else STATEMENT

The if - else statement is used to carry out a logical test and then take one of two possible actions, depending on the outcome of the test (i.e., whether the outcome is true or false).

The else portion of the if - else statement is optional. Thus, in its simplest general form, the statementcan be written as

if (expression) statement

The expression must be placed in parentheses, as shown. In this form, the statement will be executedonly if the expression has a nonzero value (i.e., if expression is true). If the expression has a value ofzero (i.e., if expression is false), then the statement will be ignored.

The statement can be either simple or compound. In practice, it is often a compound statement whichmay include other control statements.

The general form of an if statement which includes the else clause is

if (expression) statement 7 else statement 2

If the expression has a nonzero value (i.e., if expression is true), then statement 7 will be executed.Otherwise (i.e., if expression is false), statement 2will be executed

It is possible to nest (i.e., embed) if - else statements, one within another. There are several different

forms that nested if - else statements can take. The most general form of two-layer nesting is

```
if e1 if e2 s1
else s2
else if e3 s3
else s4
```

where e1, e2 and e3 represent logical expressions and s 1, s2,s3and s4 represent statements. Now, one complete if - else statement will be executed if e1is nonzero (true), and another complete if - else statement will be executed if e1 is zero (false). It is, of course, possible that s7, s2,s3and s4 will contain other if - else statements.

.

b. Write a program in C to find the sum of the series Y = $1^2 + 2^2 + 3^2 + 4^2 + 5^2 + \ldots\ldots + n$ using a while loop.

```
#include<stdio.h>
#include<math.h>
 void main()
{
 int i, n =0;
float sum = 0 ;
printf(“ enter n for the number of terms”);
scanf(“%d”,&n);
while(i<n)
{
  i++;
sum += pow(i,2);
}
printf(“sum = %f”,sum);
}
```

c. What is the difference between while and do-while statements in C.

While statement	Do While statement
while is an ENTRY CONTROLLED LOOP	do while is an EXIT CONTROLLED LOO
while (expression) statement	do statement while (expression);
The statement will be executed repeatedly, as long as the expression is true (that is as long expression has a nonzero value). This statement can be simple or compound, though it is usually a compound statement. It must include some feature that eventually alters the value of the expression, thus providing a stopping condition for the loop	The statement will be executed repeatedly, as long as the value of expression is true (i.e., is nonzero). Notice that statement will always be executed at least once, since the test for repetition does not occur until the end of the first pass through the loop. The statement can be either simple or compound, though most applications will require it to be a compound statement. It must include some feature that eventually alters the value of expression so the looping action can terminate.
The statements of the loop execute based on the condition	The statements of the loop execute atleast once
Condition to be satisfied is a true condition	Condition to be satisfied is a false condition
Eg	eg

d. Write a function in C to swap two integer variable using call by value and call by reference.

```
#include< sidio.h>
   void swapv(int  , int);
 void swapr(int & , int & );
 void main()
{
  int a, b;
printf("Enter the values of a nd b");
 scanf("%d%d",&a,&b);
printf("%d   %d\n",a,b);  // will display original
swapv(a,b);
printf("%d   %d\n",a,b);  // will display original
swapr(a,b);
printf("%d   %d\n",a,b);  // will display swapped
}
   void swapv(int v1  , int v2)
{
 int temp;
  temp = v1;
  v1 = v2;
  v2 = temp;
printf("%d   %d\n",v1,v2); // will display swapped
}
```

```
  void swapr(int &v1  , int &v2)
{
 int temp;
  temp = v1;
  v1 = v2;
  v2 = temp;
printf("%d   %d\n",v1,v2); // will display swapped
}
```

e. Explain the switch.. case statement in C with an example.

THE switch STATEMENT

The switch statement causes a particular group of statements to be chosen from several available groups,The selection is based upon the current value of an expression which is included within the switch statement.The general form of the switch statement is

```
switch (expression) statement
```

where expression results in an integer value. Note that expression may also be of type char, sinceindividual characters have equivalent integer values.The embedded statement is generally a compound statement that specifies alternate courses of action.Each alternative is expressed as a group of one or more individual statements within the overall embedded statement.
For each alternative, the first statement within the group must be preceded by one or more cuse labels(also called case prefixes). The case labels identifL the different groups of statements (i.e., the differentalternatives) and distinguish then from one another. The case labels must therefore be unique within a given switch statement.

In general terms, each group of statements is written as

```
case expression :
statement 1
statement 2
. . . . .
statement n
```

or, when multiple case labels are required,

```
case expression 1 :
case expression 2 :
. . . . .
case expression m :
statement 1
statement 2
. . . a .
statement n
```

where expression 1, expression 2, . . . , expression m represent constant, integer-valued
expressions. Usually, each of these expressions will be written as either an integer constant or a character constant. Each individual statement following the case labels may be either simple or complex. When the switch statement is executed, the expression is evaluated and control is transferred directly to the group of statements whose case-label value matches the value of the expression. If none of the case labelvalues matches the value of the expression, then none of the groups within the switch statement willbe selected. In this case control is transferred directly to the statement that follows the switch statement.

```
switch (choice = getchar())
 {
case ' r ':
case 'R’:
p r i n t f ( "RED");
break;.
 case ' w ' :
case 'W':
p r i n t f ( "WHITE");
break;
case ' b’ :
case ' B ' :
printf("BLUE");
}
```

f. Explain the following used with function in C

[2]
[1]
[2]

i) function prototype A function definition has two principal components: the first line (including the argument declarations), and the body of the function. The first line of a function definition contains the type specification of the value returned by the function, followed by the function name, and (optionally) a set of arguments, separated by commas and enclosed in parentheses. Each argument is preceded by its associated type declaration. An empty pair of parentheses must follow the function name if the function definition does not include any arguments. In general terms, the first line can be written as
data- type name(type 1 arg 7, type 2 arg 2, . . ., type n arg n)
where data - type represents the data type of the item that is returned by the function, name represents the function name, and type I,type 2, . . . , type n represent the data types of the arguments arg I , arg 2,. . . , arg n. The data types are assumed to be of type int if they are not shown explicitly. However, the omission of the data types is considered poor programming practice, even if the data items are integers.

ii) formal arguments The arguments are called formal arguments, because they represent the names of data items that are transferred into the function from the calling portion of the program. They are also known as parameters or formal parameters. The identifiers used as formal arguments are "local" in the sense that they are not recognized outside of the function. Hence, the names of the formal arguments need not be the same as the names of the actual arguments in the calling portion of the program. Each formal argument must be of the same data type, however, as the data item it receives from the calling portion of the program.

iii) return expression Information is returned from the function to the calling portion of the program via the return statement. The return statement also causes the program logic to return to the point from which the function was accessed.In general terms, the r e t u r n statement is written as return expression, The value of the expression is returned to the calling portion of the program, as in Example 7.2 above. The expression is optional. If the expression is omitted, the r e t u r n statement simply causes control to revert back to the calling portion of the program, without any transfer of information. Only one expression can be included in the return statement. Thus, a function can return only one value to the calling portion of the program via return.A function definition can include multiple return statements, each containing a different expression. Functions that include multiple branches often require multiple returns.

4.	Attempt <u>any three</u> of the following:	**15**
a	Explain the keywords in C i) auto auto i n t a, b, c; Automatic variables are always declared within a function and are local to the function in which they are declared; that is, their scope is confined to that function. Automatic variables defined in different functions will therefore be independent of one another, even though they may have the same name. Any variable declared within a function is interpreted as an automatic variable unless a different storageclass specification is shown within the declaration. This includes formal argument declarations Since the location of the variable declarations within the program determines the automatic storage class, the keyword auto is not required at the beginning of each variable declaration. There is no harm in including an auto specification within a declaration, though this is normally not done. ii) register register i n t a, b, c; the values of register variables are stored within the registers of the central processing unit. Avariable can be assigned this storage class simply by preceding the type declaration with the keyword register. There can, however, be only a few register variables (typically, two or three) within any one function. The exact number depends upon the particular computer, and the specific C compiler. Usually, only integer variables are assigned the register storage class iii) static static int count = 0; Static variables are defined within a function in the same manner as automatic variables, except that the variable declaration must begin with the s t a t i c storage-class designation. Static variables can be utilized within the function in the same manner as other variables. They cannot, however, be accessed outside of their defining function.	[1] [2] [2]
b	What is meant by scope of a variable in a C program ? There are two different ways to characterize variables: by data type, and by Storage class). Data type refers to the type of information represented by a variable, e.g., integer Number, floating-point number, character, etc. Storage class refers to the permanence of a variable, and its scope within the program, i.e., the portion of the program over which the variable is recognized.There are four different storage-class specifications in C: automatic, external, static and register. Theyare identified by the keywords auto, extern, s t a t i c , and r e g i s t e r , respectively. The storage class associated with a variable can sometimes be established simply by the location of the variable declaration within the program. In other situations, however, the keyword that specifies a particular storage class must be placed at the beginning of the variable declaration. Automatic variables are always declared within a function and are local to the function in which they are declared; that is, their scope is confined to that function. External variables, in contrast to automatic variables, are not confined to single functions. Their scopeextends from the point of definition through the remainder of the program. Hence, they usually span two ormore functions, and often an entire program. They are often referred to as global variables Static static variables retain their values throughout the life of the program. Thus, if a function is exited and then re-entered at a later time, the static variables defined within that function will retaintheir former values. register	
c	What is a macro? Write a program in C to find the area of a rectangle and square using macros. The #define statement can be used for more, however, than simply defining	

symbolic constants. In particular, it can be used to define macros; i.e., single identifiers that are equivalent to expressions, complete statements or groups of statements. Macros resemble functions in this sense. They are defined in analtogether different manner than functions, however, and they are treated differently during the compilation process.
Macro definitions are customarily placed at the beginning of a file, ahead of the first function definition. The scope of a macro definition extends from its point of definition to the end of the file. However, a macro defined in one file is not recognized within another file.
Multiline macros can be defined by placing a backward slash (\) at the end of each line except the last. This feature permits a single macro (i.e., a single identifier) to represent a compound statement.

```
 #include <stdio.h>
#define area length * width
#define sqarea length*length
void main( )
{
Int  length, width;
printf ("length =);;
scanf ('%d", &length);
printf ("width = " ) ;
scanf ("%d" , &width);
printf ( " \ n area of rectangle = %d", area);
printf ( " \ n area of square = %d", sqarea);
}
```

d What is an array? How can a single dimensional array be initialized?
Many applications require the processing of multiple data items that have common characteristics (e.g., a setof numerical data, represented by XI, x2, . . . ,xn). In such situations it is often convenient to place the data items into an array, where they will all share the same name (e.g., x). The individual data items can be characters, integers, floating-point numbers, etc. However, they must all be of the same type and the same storage class.
Each array element (i.e., each individual data item) is referred to by specifLing the array name followed by one or more subscripts, with each subscript enclosed in square brackets. Each subscript must be expressed as a nonnegative integer. In an n-element array, the array elements are x[0], x[1],x[2],……x[n-1]. The value of each subscript can be expressed as an integer constant, an integer variable or a more complex integer expression. The number of subscripts determines the dimensionality of the array. For example, x [i] refers to an
element in the one-dimensional array x. Similarly, y[i][j] refers to an element in the two-dimensional array y.

Arrays are defmed in much the same manner as ordinary variables, except that each array name must be accompanied by a size specification (i.e., the number of elements). For a one-dimensional array, the size is specified by a positive integer expression, enclosed in square brackets. The expression is usually written as a positive integer constant.
In general terms, a one-dimensional array definition may be expressed as
storage -class data- type array[expression] ;

where storage -class refers to the storage class of the array, data- type is the data type, array is the array name, and expression is a positive-valued integer expression which indicates the number of array elements. The storage-class is optional; default values are automatic for arrays that are defined within a function or a block, and external for arrays that are defined outside of a function.

Automatic arrays, unlike automatic variables, cannot be initialized. However, external and static may definitions can include the assignment of initial values if desired. The initial values must appear in the order in which they will be assigned to the individual array elements, enclosed in braces and separated by commas.
The general form is
storage- class data- type array[expression] = { value 1, value 2, . . . , value n) ;
where value I refers to the value of the first array element, value 2 refers to the value of the second element, and so on. The appearance of the expression, which indicates the number of array elements, is optional when initial values are present.

```
int digits[lO] = (1, 2, 3, 4, 5 , 6, 7, 8, 9, 10);
static float x(6] = (0, 0.25, 0, -0.50, 0, 0);
char color[3] = { ' R ‘ , 'E', ‘D ' } ;
```

e Write a program in C to find a number in an array of ten integers and display its position

```
#include < stdio.h>

void main()
{
int a[10];
int no,
 printf(“ Enter 10 numbers in the array”);
for (int i=0, i<10,i++)
scanf(“%d”,&a[i]);
prinf(“\n Enter the number to be searched”);
scanf(“%d”, &no);

for (int i=0, i<10,i++)
{
if ( no == a[i])
{
printf(“ the number  %d is found at position  %d”,no,  i+1);
goto abc;
}

}
printf(“ the number  %d is not found”, no);

abc: }
```

f What is a string? Explain different ways of entering data in a string in C.
A string constant consists of any number of consecutive characters (including none), enclosed in (double) quotation marks.
"green" "Washington, D.C. 20005H” "270-32-3456"
"$19.95" "THE CORRECT ANSWER IS:' '2* (I+3)/J "
"Line l\nLine 2\nLine 3" “ “

The string “ “ is a null (empty) string. The compiler automatically places a null character (\O) at the end of every string constant, as the last character within the string (before the closing double quotation mark). This character is not visible when the string is displayed. However, we can easily examine the individual characters within a string, and test to see whether or not each character is a null character. the end ofevery string can be readily identified. This is very helpful if the string is scanned on a character-by-character basis, as is required in many applications. Also, in many situations this end-of-string designation eliminates the need to spec@ a maximum string length. the array is usually written without an explicit size specification (the square brackets are empty). The array name is then followed by an equal sign, the string (enclosed in quotes), and a semicolon. This is a convenient way to assign a string to a character-type array.

```
char x[] = “This string is declared externally\n\n”;
printf ("%s", x) ;

char *y = "This s t r i n g i s declared within main";
printf ( "%s", y) ;


char line[80];
scanf("%s", line);

scanf("%[^\n]",line) ;

gets(line);
```

5. Attempt <u>any three</u> of the following: **15**

a. Explain

i) Pointer declaration

Pointer variables, like all other variables, must be declared before they may be used in a C program. The interpretation of a pointer declaration differs, however, from the interpretation of other variable declarations. When a pointer variable is declared, the variable name must be preceded by an asterisk (*). This identifies the fact that the variable is a pointer. The data type that appears in the declaration refers to the object of the pointer, i.e., the data item that is stored in the address represented by the pointer, rather than the pointer itself.
Thus, a pointer declaration may be written in general terms as

```
data- type *ptvar;
```

where ptvar is the name of the pointer variable, and data-type refers to the data type of the pointer’s object. Remember that an asterisk must precede ptvar.

```
float  u, v;
float  *pv;
```

The first line declares u and v to be floating-point variables. The second line declares pv to be a pointer variable whose object is a floating-point quantity; i.e., pv points to a floating-point quantity. Note that pv represents an address, not afloating-point quantity

ii) ‘ * ‘ use to declare a pointer variable in a declaration statement .

f l o a t *pv; /* pointer variable declaration */

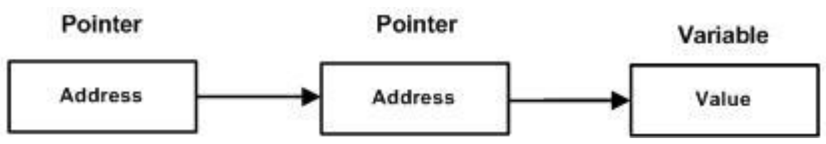
And represents the value of the address when used with a pointer in others statement . printf(“%d”, *pv +4);

‘ & ’ used with pointers to extract the address of the variable

pv = 8v; / * assign v ' s address to pv */

b. What is the concept of pointer to pointer in C? Give suitable example

A pointer to a pointer is a form of multiple indirection, or a chain of pointers. Normally, a pointer contains the address of a variable When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value



A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name.
the following declaration declares a pointer to a pointer of type int

```
int ** p1,
#include<stdio.h>
void main()
{
 int **p1,
int * p2;
int a = 20;

p2 = &a;
p1= &p2;
printf (“%d”,a); // prints the value of a that is 20
printf (“%d”,*p1); // prints the value of a that is 20
printf (“%d”,**p2); // prints the value of a that is 20
}
```

c. Write a program in C to display the cube of ten elements of an integer array using pointers

```
#include < stdio.h>
 #include< math .h>
void main()
{
int a[10];
 int *ptr ;

 printf(“ Enter 10 numbers in the array”);
for (int i=0, i<10,i++)
scanf(“%d”,&a[i]);
ptr = &a[0];
for (int i=0, i<10,i++)
{
 printf(“ the cube of the number  %d is  %lf”,*ptr , pow(*ptr, 3));
 ptr++;
}
}
```

d. Explain the statements of the main( ) function with its output

```
#include<stdio.h>
void main( )
{ int line[80];
   int *p1;
   line[2] = line[1];
   line[2] = *( line+1);
   *( line+3)= line[1] +3 ;
   *( line+2)= *( line+1);
   p1 = & line[1];
   p1 =  line +1;
}
```

	Third and fourth statements assigns the value of the second array element (i.e, line [1]) to the third array element (line [2]). Thus, the statements are all equivalent. Fifth statement adds 3 to the 2nd element of the array and assignes it to 4th element in the array Sixth statement assigns the value of the second array element (i.e, line [1]) to the third array element (line [2]) The last two assignment statements each assigns the address of the second array element to the pointer pl.	

e.	Explain nested structure in C with example.
	A structure variable may be defined as a member of another structure. In such situations, the declaration of the embedded or nested structure must appear before the declaration of the outer structure.
	struct date {
	int month;
	int day;
	int year;
	};
	struct account {
	int acct-no;
	char acct-type;
	char name[80];
	float balance;
	struct date lastpayment ;
	} oldcustomer, newcustomer;
	The second structure (account) now contains another structure (date) as one of its members. Note that the declaration of date precedes the declaration of account.
	
	oldcustomer = (12345, ' R I , "John W. Smith", 586.30, 5, 24, 90);
	The first member (acct-no) is assigned the integer value 12345, the second member (acct-type) is assigned the character ' R I ,the third member (name [801) is assigned the string John W. Smith ",and the fourth member (balance) is assigned the floating-point value 586.30. The last member is itself a structure that contains three integer members (month, day and year). Therefore, the last member of customer is assigned the integer values 5, 24 and 90.	

f.	What is an array within a structure and array of structure?
	An array which is a part of the a structure is an element declared within a structure In the example below it is char name[80]
	struct date {
	int month;
	int day;
	int year;
	};
	struct account {
	int acct-no;
	char acct-type;
	char name[80];
	f l o a t balance;
	s t r u c t date lastpayment;
	} customer[100] ;
	In this declaration customer is a 100-element array of structures. Hence, each element of customer is a separate
	structure of type account (i.e., each element of customer represents an individual customer record).
	Note that each structure of type account includes an array (name[80]) and another structure (date) as members.Thus, we have an array and a structure embedded within	

	another structure, which is itself an element of an array.	