C' Language Notes

What is Complier?

*Compiler is a program that converts the source code (program written in a language other than machine language) in to machine code. Computer can understand machine code and execute machine code**. Therefore we can state that compiling is a process of converting source code to machine code if source code is error free.***

What is the basic structure of c program?

Consider a simple program

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b;
printf("\n enter two values");
scanf("%d %d",&a,&b);
c=a+b;
printf("\n addition=%d",c);
getch();
}
```

We can note following facts:

1. Program starts with #include

*#include is known as preprocessor directive. This preprocessor directive gives instruction to preprocessor program to include contents of file mentioned in angle bracket (or in double quotes) to our source file. The file name in angle bracket is known as include file (because this is used always with include preprocessor) or header file (appears in head/top position of file). **This header files contents macro, constant and function declaration**.*

2. void main()

void specifies function does not return value.

***'main'** is a function which acts as entry point from which our program statement starts running.*

Declaration statement is followed by executable statements.

3. Blank spaces may be inserted to increase readability.

4. Usually all statements are entered in lowercase

5. We can start writing statement from any column position.

6. Any 'C' statement terminates by semicolon;

Write program to swap two values.

(a) using a third variable.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=3,b=4,c;
c=a;
a=b;
b=c;
printf("\n a=%d,b=%d",a,b);
getch();
}
```

(b)without using a third variable.

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

```
void main()
{
int a=3,b=4;
a=a+b;
b=a-b;
a=a-b;
printf("\n a=%d,b=%d",a,b);
getch();
}
```

What does void means in front of main()?
void means function is not going to return a value. If we remove void we have to use 'return 0;' statement before closing brace of main function.

Why do we use getch() before closing brace of main function?
getch() is a library function which can input a character from user. Using getch() before closing brace of main lets us read program output by halting the output window. If we do not give getch() then we must use alt+f5 key combination to read program output because we will return to source code window as soon as program completes.

Can we write inclusion of header file in any order?
Yes, we can include header file in any order.

What is the difference between #include<stdio.h> and #include "stdio.h"?
#include <stdio.h> tells the preprocessor to search the header file in the directory mentioned in integrated development environment only and insert the contents of header file in source file.
#include "stdio.h" tells the preprocessor to search the header file in the directory in which source code lies and if header file not found in the directory where source code lies then search the header file in the directory mentioned in integrated development environment and finally insert the contents of header file in source file.

What is linking/ why do we need linking after compiling?
Linking is a process which runs after compiling process to combine several dependent machine codes (object code) into a single executable code. The program which performs the linking process is known as linker.

What is preprocessor?/Write short notes on preprocessor or list different preprocessor
Preprocessor is a program which works before compiling process. It follows the instruction given by preprocessor directives. After preprocessor has finished its work compiling process begins.
*Preprocessor directive begins with # because this instructs the preprocessor what is followed is not an ordinary **'c'** statement but a preprocessor directive.*
e.g. #include <stdio.h>
This tells the preprocess program to insert contents of stdio.h file into current source file (program written in high level language).
Some preprocessor are:
#include, #define, #ifdef, #ifndef, #undef, #else, #endif, #pragma etc.

What are the popular features of 'C' language?
The features are:

1. Portability: This refers to the ability of a program to run in different environments. Different environments could refer to different computers, operating systems, or different compilers. Since 'C' language is mid level language and its compiler is available for variety of environments we can say it is portable.

2. Flexibility: ‘c’ language is flexible because:
a. It provides facility of creating user defined data type.
b. It is not necessary to write statement in ‘c’ in a particular format which means we can start writing statement in ‘c’ from any column.
3. Mid level language: C is a mid level language combines feature of high level language (faster code development) and low level language (efficient program).
4. Wide acceptability: Majority of people know ‘C’ language.
5. System Programming: system programming and application programming using ‘C’ language are possible.

What is an IDE?
An IDE is nothing else but it is a program which includes all facilities to develop and run program, such as editor, compiler, debugger and so on.

What is C?
C is a programming language developed at AT & B bell lab. of USA in 1972. It was developed by Dennis Ritchie. It is user friendly, general purpose and reliable language.

How C originated? Or Write short notes on history of C language.
By 1960 a hoard of computer language had come into existence almost each for a specific purpose. An International committee was setup to develop a general purpose language using which any type of application can be developed easily and the outcome of the effort was Alogol-60. Algol-60 did not become popular because of its abstract nature and many complicated features. Combined programming language was developed with reduced feature of Alogol-60 but it was still vague and did not become popular. Later on basic combined programming language was developed with reduced feature when compared to combined programming language. Basic Combined Programming Language had very limited feature and did not become popular. **B language was developed by Ken Thompson** as further enhancement to Basic Combined Programming Language. Dennis Ritchie developed some new feature and used features of language ‘B’ and Basic Combined Programming Language and developed ‘c’ and ‘c’ became popular.

Why c is called a middle level language?
All the programming languages can be divided into two categories:

a. *Problem oriented languages or high level languages: These language have been designed to give a better programming efficiency. i.e. faster program development E.g. Fortran, Basic.*
b. *machine oriented languages or low level languages: These language have been designed to give a better machine efficiency means faster program execution.*

‘c’ *stands in between these two categories. That is why it is called a middle level language it has got relatively good programming efficiency (as compared to machine oriented languages) and relatively good machine efficiency (as compared to high level languages).*

What do you mean by character set of C?
A character denotes any alphabet, digit or special symbol used to represent information. All those characters which can be part of a ‘c’ program and program still remains valid is knowns as ‘c’ character set. Few e.g. as alphabets a-z, digits -9, special symbols ,\,{,} etc.

What is Syntax?
Syntax is correct way of writing statement according to rules suggested by language.

What is constant?/What are the several type of constant that c language supports?. what is the difference between variable and constant? how can variable be initialized?
A constant is a quantity that doesn’t change during program run time. This quantity can be stored at a location in memory.

A variable is that whose value can change during run time of program.
initialization of variable: int a=10;
float c=30.4; etc.
Type of c constant :-
(A) primary constant which includes
Integer constant-
1. *Integer constant must have at least one digit.*
2. *It must not have a decimal point.*
3. *It could be either positive or negative.*
4. *If no negative sign is present the constant is treated as positive*
5. *The allowable range is -32768 to 32767*

```
   e.g. 426
        +782
        -343
       0x179  /* ok hexadecimal integer constant */
      0179  /* ok octal integer constant */
     invalid is  172 9   /* constant must be continuous without space */
    invalid is   173 bcdl  /* there is not such bcdl  constant */
```

Real constant-
1 .A real constant must have at least one digit.
2. It must have a decimal point.
3. It could be positive or negative.
4. Default sign is positive.
5. No commas or blanks are allowed within a real constant

```
e.g. +325.34
       463.0
       10.32e+10
invalid is 10.32 e+.08 /* exponent must be integer */
```

Character constant-
1. A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas. Both the inverted commas should point to the left.
2 The maximum length of a character constant can be 1 character.

```
e.g. ‘a’
      ’5’
      ‘y’

invalid is  ‘ab’  /* only single character can be used */
invalid is “a” /* single quote must be used */
```

(B) Secondary constants
Array, pointers, structure, union, enum etc. are known as secondary constant.

What are the different data types available in c?
ans:
char,unsingned char, int,unsigned int,long ,unsigned long,float,double, long double .

What is variable?
Variable is name given to a location of computer memory where we can store value and retrieve the stored value by variable name. Value of variable can change during program run time and it is fundament requirement of any programming language.
Rules for constructing identifier (variable/pointer/function name):
1. *A variable name is any combination of alphabets, digits and underscore. Some compiler supports 40 character lengthy variable names.*
2. *The first character in variable name must be alphabet.*

3. *No commas or blanks or special character can appear in variable name.*
4. *Variable name is case sensitive therefore variable* ***Area*** *and variable* ***area*** *are different.*
5. *Variable name must not match with keywords.*

e.g. few valid variable names are:

```
 si_int
 pop_pe_8
```

few invalid variable names are

```
a b   /* we can not use space */
1b  /* must start with alphabet */
char  /* char is keyword */
```

What is keyword?

Keyword or reserved word is word whose meaning is well defined for compiler we can use it in our program but we can not alter its meaning.
e.g. int, float, do, while etc.
There are such 32 keywords.

What is statement or instruction?

An instruction requests computer to perform some work. These are following types:
1. **Type declaration statement (non executable statement)**
This is useful to declare variables and functions used in c program.
2. **Input/output statement (executable statement)**
This is useful to supply data into program(input statement) or to take output from program(output statement).
3. **Arithmetic assignment statement**
This is useful to perform arithmetic operations between constant and variables.
Arithmetic statement may be of three types:
a. Integer mode statement in which integer variables and constant appear in statement.
b. Real mode statement in which real variables and constant appear in statement.
c. Mixed mode statement in which integer and real variables and constant appear in statement.
4. ***Control statement***
This is useful to control the sequence of execution of various statements. Control statements include looping and branching. Branching includes if..else, switch where as looping include while, do..while, for loop.
5.***Conditional statement***
conditional statement is made of variable, constant, relational operators and logical operators. Conditional statement returns either true or false. In c's point of view, 0 is false and non zero is true.
e.g. int c,a=3,b=4;

```
      c= a = = 3|| b>4;   /* this is conditional statement */
```

What is Expression?

An expression is formed using operator and operand. Operand can be variables or constants on which operator works.
For example c=a+b;
In this statement '+' is an operator .
a and b are operand.

What is type conversion? Why type casting is necessary?

It may happen that the type of the expression and the type of the variable on the left hand side of the assignment operator may not be same. In such a case the value of the expression is promoted or demoted depending on the type of the variable on left hand side of =.
For example: If we consider following statement

```
int I;
float b;
```

I=3.5;
B=30;
Here, in first assignment statement though the expression's value is ***float*** *it can not be stored in* ***I*** *since* ***I*** *it is int therefore float is demoted to int.*
Here in second assignment statement though the expression's value is int it will be stored in b as 30.000000
In some situation type casting becomes necessary. Suppose we want to store result of integer division to a float variable such as
float a;
a=10/3;
In that case a will store 3.000000 rather than 3.333333
If we change the expression as
A=(float)10/3; then value of a will be 3.333333 in this case explicit data type conversion is necessary.

Write abbreviation of sum=sum+x.
ans.
sum+=x;

How one can read or write values of variable or constant?/ explain component of printf , scanf.
(a)Values of variable or constant can be read by:
scanf function whose syntax is
scanf("format strings",&var1,&var2…);
format string can be constructed using format specifier character e.g.
%c, %d, %u, %f.
Which format specifier to use? depends on data type of variable to read value in.
Suppose variables are declared as
int a,b;
float e;
char c;
then we need statement
scanf("%d %d %f %c",&a,&b,&e,&c);
Since a and b are **int** types we used corresponding format specifier %d two times because there are two int type variables to read values in.
Since f and c are float and char types respectively we used corresponding format specifier %f and %c.
It means that no. of variables required read values in, must match with no. of format specifier. Each data type uses a predefined format specifier and we have to use the predefined one.
(b) writing of values of variable or constant: it can be done using printf statement whose syntax is as follows
printf("message to print");
e.g. printf("enter an integer");
or
printf("format string",var1,var2);
if we have declared variables as:
int a=10;
float b=20;
we can give statement
printf("%d %f",a,b);
Since a is int we used %d format specifier.
Since b is float we used %f format specifier
The values of variable or constant to print replace format specifier at print time.
Therefore output will be
10 20.000000
if we give statement
printf("a=%d,b=%f",a,b);

output will be

```
a=10 b=20.000000
```

How can we read character constant in character variable?
(Single character input-output)

Character variable is treated internally as integer therefore we have two ways to read a character constant in character variable.

(1)Character constant can be read using %c format specifier when to read a single character.
Consider following program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char p;
printf("\n enter a character");
scanf("%c",&p); /* or we can use
c=getchar();
c=getche();
c=getch();
*/
printf("\n the character read is %c and ascii value is %d",p,p);
getch();
}
```

If we enter a character 'a' during run time of program then output will be:

```
The character read is a and ascii value is 97
```

(2)Character constant can be read using %d format specifier when to read an integer value corresponding to ascii value of character to read

```
#include<stdio.h>
#include<conioh>
void main()
{
char p;
printf("\n enter ascii value corresponding to character to read");
scanf("%d",&p);
printf("\n the character read is %c and ascii value is %d",p,p);
getch();
}
```

If we enter 97 during run time of program then output will be:

```
The character read is a and ascii value is 97
```

Compare getch(), getchar(), getche() library function.

getch()	***Getchar()***	***getche()***
this function needs conio.h	*This function needs stdio.h*	*This function needs conio.h*
(1)Inputs a character and assigns the character in a character variable.	*(1)Inputs a character and assigns the character in a character variable.*	*(1)Inputs a character and assigns the character in a character variable.*
(2) We need not to press enter key after pressing a key.	*(2)We need to press enter key after pressing a character.*	*(2)We need not to press enter key after pressing a key.*

(3)Does not echoes(displays) the character pressed.	*(2)Echoes the character which was pressed.*	*(3)Echoes(displays) the character pressed.*

What is qualifier?
Qualifier modifies the behavior of the variable type to which they are applied. We have seen declarations such as
int I;
This declaration specifies that is ***'I'*** *is an integer which can take both positive and negative values, that is, I is a signed integer by default. The above declaration could also be written as:-*
signed int i;
qualifier can be two types:
*a)**Size qualifier***
size qualifier modifies the range of value, a variable can store.
short: applies to int
long: applies to int and double

*b)**Sign qualifier***
signed: applies to int, char
unsigned: applies to int, char

What do you mean by typedef statement?/define type declaration in c.
The typedef (keyword) statement allows creating new name for existing data type.
For example if we write a statement
typedef unsigned long ulong;
Then we can use statement :-
ulong a;
This means that:
unsigned long u;

What is the difference between 'a' and "a"?

'a'	***"a"***
'a' is a character constant.	*"a" is a string constant*
Character constant can have a single character enclosed inside single quote.	*String constant can have one or more character enclosed inside double quotes.*
a character constant takes just one byte of memory.	*a string constant takes no. memory that is sum of no. of characters inside double quotes plus 1 byte taken by null character.*
%c is format specifier for formatted input/output.	*%s is format specifier for formatted input/output.*
getch(),getchar(),getche() are unformatted input library functions.	*gets() is a unformatted input library function.*

What are the different types of operators available in c?
Different types of operator are available in 'C' are:

Arithmetic operators	*+,-,*,/,%*

Relational operators	*>,<,>=,<=,= = ,!=*

Logical operators	*&&,||,!*

Assignment operators	*=* *compound assignment operators are:* *+=,-=,%=,/= etc.*

Increment/decrement operators	*++,--*

Conditional operators	*(condition) ?true statement :false statement*

Bitwise operators	*&,|,^,<<,>>,~*

Comma operators	*,*

Other operators	*Sizeof*

Write short notes on shorthand operators.
short hand operator are ++,- -,+=,-=,= etc.*
suppose int a=1;
a++;   value of a is now 2
suppose int a=1;
a--; value of a is now 0
suppose int a=5;
a+=10;   means a=a+10 therefore a will be 15
etc.

What special keyword is used in defining symbolc constant?
ans: const

What are trigraph characters?
the sevent-bit ascii code is the character set of a c source code. the reduced set, iso (invariant code set)) 646-1983 do not have many of the symbols used in the c language. hence, to enable user to write c source in this reduced set, trigraph sequences may be used.
trigraph sequence replaced with the preprocessor
??=   #
??/   \ etc.

What is escape sequence?
Escape sequence is some character which are preceded by \ character and printf statement interprets the character in different way. for e.g.
\n to print newline character
\t to print tab character
\” to print double quote character
*\\ to print a single *
etc.

Write short notes on bitwise operators.

a bitwise operator operates on each bit of data and these operators are used for testing , complementing or shifting bits to the right or left. Usually bitwise operators are not useful in case of float and double variables. a list of bitwise operators are given below:
& bitwise and
| bitwise or
^ bitwise xor
<< left shift operator
>> right shift operator
~ complement operator
suppose int a=13 ,b=7,c;

c=a&b; first row is a's bit pattern ,second row is b's bit pattern ,third row is c's bit pattern

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

c=a|b; first row is a's bit pattern ,second row is b's bit pattern ,third row is c's bit pattern

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

c=a^b; first row is a's bit pattern ,second row is b's bit pattern ,third row is c's bit pattern

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

c=a<<3; first row is a's bit pattern ,second row is c's bit pattern

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0

clearly c= a(2^3)*

c=a>>3; first row is a's bit pattern ,second row is c's bit pattern

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

clearly c= a/(2^3)

c=~a; first row is a's bit pattern ,second row is c's bit pattern

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0

Explain the behavior of right shift operator for positive and negative integer.
Zeros are always inserted in the left if integer is positive otherwise 1 is inserted in the left, if integer is negative when using right shift operator.

if we assign -3 to a variable 'a'
int a=-3;
then binary representation of 'a' in two's complementary system is computed as follows:
binary representation of positive 3 is
0000 0000 0000 0011
taking complementary
1111 1111 1111 1100
by adding 1 to get two's complementary
1111 1111 1111 1101 which is binary pattern of -3
now c=a>>2;
will shift the value of a by 2 bit positions to the right and will insert two 1's in the left. so value of c will be
1111 1111 1111 1111

once again taking complementary
0000 0000 0000 0000
adding 1 to get two's complementary
0000 0000 0000 0001

2's complement of a 2's complement is the original number itself result
indicates it is -1 in decimal.

What do you mean by mixed mode arithmetic?
When a statement uses expression and expression in turn uses variables and constant belonging to different data type we call it mixed mode arithmetic.
For example
int a=10;
float b=35;
float c;
c=a+b;
It is a mixed mode arithmetic because one operand is type int which is a and another is type float which is b.

What is the difference between a++ and ++a/(post increment and pre increment operator)?
(a)If we use following statements
int a=5,b;
b=++a;
printf("\n a=%d b=%d",a,b); output will be a=6 and b=6.
(b)If we use following statements
int a=5,b;
b=a++;
printf("\n a=%d b=%d",a,b); output will a=6 and b=5.

What do you mean by operator precedence and associativity?
If we write a statement and statement contains operator and operands then operator precedence controls the order of evaluation of operands.
e.g.
int a=10,b=15,c=3,d;
*d=a+b*c;*
in the above expression, b is multiplied by c first, the result is added to a and the sum is assigned to d.
*since * has higher operator precedence therefore * operator evaluates first.*
We can change the operator precedence with parentheses
*d=(a+b)*c;*

in above statement a and b is evaluated first then sum of a and b is multiplied by c then assigned to d. Associativity of operator can be left to right or right to left. For unary, compound assignment and ternary operator associativity is right to left for others it is left to right.
Consider c=a+b+d;
If two operators having same operator precedence occurs in a mathematical assignment statement then associativity starts playing its role. Since + operator has associativity from left to right therefore a+b will be evaluated first then result of a+b will be added to d and finally result of evaluation will be assigned to c.

What is Control Structure?
Control structure specifies the order in which the various instructions in a program are to be executed by the computer.
There are four type of control structure or control statement:
* Sequence control instruction –ensures statement are executed in same order in which they appear in program.
* Selection or decision control instruction-if given condition becomes true executes one set of statement if given condition becomes false executes different set of statement.
* Repetition or loop control instruction- given set of statements are executed repeatedly till given condition is true.
* Case control instruction – checks value of a single variable/constant and performs different action depending on case defined for value of single variable/constant.

Why do we need control structure?
We all need to alter our action depending on changing circumstances. For example If weather is fine I will go Dongargarh on bike otherwise I will go by train. In the same way 'C' language too must be able to perform different action under one condition and different action under another condition and this is made possible by control structure. Sometimes a given set of statements are needed to run repeatedly this is done using loop control structure.

Write notes on if control structure.
The general form of if statement is:
(a)if
the general format of if statement is

```
 if (condition is true)
  execute this statement;
```

The keyword **if** tells the compiler that what follows, is a decision control instruction. The condition following the keyword **if** is always enclosed within a pair of parentheses. If the condition, whatever it is, is true, then the statement is executed. If the condition is not true then the statement is not executed; instead the program skips past it.
(b) if -else
if statement by itself can execute only one statement if condition is true. **If it required running a group of statements when condition is true we have to enclose those statements inside curly brace known as compound statement**. The above form of if statement mentioned in (a) will not do anything when condition is false. If we want to run statement when condition is false we need **if-else** construct.
General format of if-else construct is

```
If (condition is true)
   Execute this statement;
Else
   Execute this statement if condition is false;
```

Condition is specified using relational operator
x= =y means x is equal to y.
x!= y means x is not equal to y.
x< y means x is less than y.

x>y means x is greater than y.
x<=y means x is less than or equal to y.
x>=y means x is greater than or equal to y

What is the difference between = and = = ?
= operator is useful to assign value to some variable. = = operator is useful to compare value of variables or constant and can be used to build conditional expression.
What are the different forms of if –else statements?
Different forms of if-else statement are as follows:

```
1. if (condition)
       do this;
2. if (condition)
        {
          do this;
          and this;
         }
3. if (condition)
       do this;
   else
       do this;
4. if (condition)
      {
        do this;
        and this;
      }
   else
      {
        do this;
        and this;
      }

5. if (condition)
       do this;
   else
        {
           do this ;
           and this;
         }
6. if (condition)
      {
        do this;
        and this;
      }
   else
        do this;
```

Write program to find root of quadratic equation.

```
#include<stdio.h>
#include<conio.h>
void main()
{
float a,b,c,d,r1,r2;
printf("\n enter coefficient of quadratic equation");
scanf("%f %f %f",&a,&b,&c);
d=b*b-4*a*c;
if(d==0)
{
printf("\n roots are equal\n");
r1=-b/(2*a);
r2=-b/(2*a);
printf("\n root1=%f,root2=%f",r1,r2);
}
else if(d>0)
{
printf("\n roots are real and different\n");
r1=(-b+sqrt(d))/(2*a);
r2=(-b-sqrt(d))/(2*a);
printf("\n root1=%f,root2=%f",r1,r2);
}
else
{
printf("\n roots are imaginary\n");
r1=-b/(2*a); /* real coefficient */
r1=sqrt(-d)/(2*a); /* imaginary coefficients */
printf("\n root1=%f  +  %f  i=%f",r1,r2);
printf("\n root1=%f  -  %f  i=%f",r1,r2);
}
getch();
}
```

What is the use of logical operator?
Logical operator && or || are useful to join more than one condition whereas '!' operator negates meaning of condition.
Thus we can conclude that the '!', '&&' and '||' are useful in the following programming situations:
* When it is be tested whether a value falls within a particular range or not.
* When after testing several conditions the outcome is either true or false.
* Reverse the logic of condition using '!' logical operator.

State difference between if and conditional operator/(ternary operator).
Conditional operator which is known as ternary operator can use only one 'C' statement if condition is **true** and one statement if condition is **false** whereas if statement can use multiples statement if used compound statement.

Write short notes on conditional operator/ternary operator?
The syntax of conditional operator is as follows:
(condition) condition true do this ? condition false do this;
it is sometimes called ternary operator since this take three arguments. In fact , it form a kind of shortcut notation of if-else.
Limitation of ternary operator is ; we can use one 'C' statement in each true and false part.
e.g.

```
/* find a no. is even or odd */
#include <stdio.h>
#include<conio.h>
```

```
    void main()
{
int n;
printf("\n enter a no. to test even odd");
scanf("%d",&n);
(n%2= =0)?printf("\n even"):printf("\n odd");
getch();
}
```

Explanation: if n is even no. then modulo division by 2 will be 0 and condition will be true therefore statement followed by '?' will run. If no. is odd then modulo division of n by 2 will be 1 and condition will be false therefore statement followed by ':' will run.

Why loop control structure is needed?
Without loops we execute the same series of actions, in the same way, exactly once. Loops allows to execute same portion of program code repeatedly a specified no. of times or until a particular condition is being satisfied. This repetitive operation is done through a loop control structure.
There are four methods using which we can repeat a part of a program which are:
* for statement
* while statement
* do-while statement
* goto statement (not recommended)

Write short notes on goto statement and its limitations.
'goto' is a statement which is quite common in monolithic programming paradigm. some languages like basic, fortran etc. use it deliberately. using goto we can do :
1. *skipping of some statement without executing them.*
2. *transferring of control to a labeled statement.*
3. *coming out of loop whatever level of nesting of loop may be.*

note that break statement can take out of loop inside which it lies but goto statement can take out of loops irrespective of level of nesting of loops.
different type of goto can be:
   a. *conditional-goto : if it is used with 'if' statement.*
   b. *unconditional-goto : if it is not used with 'if' statement.*

limitations of goto*: unwise use of goto make it difficult to trace logic flow of program and makes it difficult to remove logical error of program therefore, as far as possible, we must avoid use of goto.*
a simple program to print 1 to 10 natural nos. using goto.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int ii;
jump: printf("\n %d",ii);
ii=ii+1;
if (ii<=10)
 goto jump;
getch();
}
```

Write short notes on 'while' (entry level control structure/test and do) loop.
In programming we require to execute same set of instructions a fixed number of times. E.g. we want to calculate gross salary of ten different persons, we want to convert temperatures from centigrade to Fahrenheit for 15 different cities. The 'while' loop is suited for this.

```
/* calculation of simple interest for 3 sets of p,n and r */
#include<stdio.h>
```

```
#include<conio.h>
void main()
{
float p,r,t,i;
int count;
count=1;
while(count<=3)
{
printf("\n enter principal,rate and time");
scanf("%f %f %f",&p,&r,&t);
i=p*r*t/100;
printf("\n simple interest =%f",i);
count=count+1;
}
getch();
}
```

Explanation: the program executes all statements inside braces after the while 3 times. The logic for calculating the simples interest is written within a pair of braces immediately after the while keyword. The statements form what is called the body of the ***while*** *loop. The parentheses after the while contain a condition. So long as this condition remains true all statements within the body of the while loop keep getting executed repeatedly. To begin with the variable count is initialized to 1 and every time the simple interest logic is executed the value of count is incremented by one. The* ***variable count is known as loop counter or index variable.***

General format is

1.

```
initialize loop counter
while(condition is true)
  {
    do this;
    increment loop counter;
  }
```

Note: do not use ';' after closing parenthesis.

As a rule the 'while' loop must test a condition that will eventually become false, otherwise the loop would be executed forever, indefinitely.

write program to read a seven digit no. and find its sum of digit.

```
#include<stdio.h>
#include<conio.h>
void main()
{
long n;
int s;
printf("\n enter a number");
scanf("%ld",&n);
for(s=0;n>0;n=n/10)
{
s=s+n%10;
}
printf("\n sum of digit=%d",s);
getch();
}
```

Write short notes on do-while(exit level control structure/do and test) loop.

In programming you require to execute a set of statement a fixed number of times. Perhaps you want to calculate gross salary of ten different persons, or you want to convert temperatures from centigrade to Fahrenheit for 15 different cities. The do while loop is suited for this.

```
/* calculation of simple interest for 3 sets of p,n and r */
#include<stdio.h>
#include<conio.h>
void main()
{
float p,r,t,i;
int count;
count=1;
do
{
printf("\n enter principal,rate and time");
scanf("%f %f %f",&p,&r,&t);
i=p*r*t/100;
printf("\n simple interest =%f",i);
count=count+1;
} while(count<=3);

getch();
}
```

Explanation: the program executes all statements inside braces inside do-while loop body 3 times. After ***do*** *keyword the logic for calculating the simples interest is written within a pair of braces then while is followed by condition. The statements form what is called the body of the do- while loop. The parentheses after the while contain a condition. So long as this condition remains true all statements within the body of the do-while loop keep getting executed repeatedly. To begin with the variable count is initialized to 1 and every time the simple interest logic is executed the value of count is incremented by one. The* ***variable count is known as loop counter or index variable****.*

Differentiate between while and do-while loop.

Body of loop in case of 'while' loop will not execute even once if condition of while loop is false but in case of 'do-while' loop body of loop will run at least once because condition is tested at the end of loop body.

General format is

```
1.
  initialize loop counter
  do      {
       do this;
       increment loop counter;
    } while(condition is true);
2.
 initialize loop counter
  while      {
       do this;
       increment loop counter;
    }
```

Note : use ; after closing parenthesis in do-while loop.

Do while loop must test a condition that will eventually become false, otherwise the loop would be executed forever, indefinitely known as infinite or indefinite loop.

Write short notes on for loop.

‘for’ loop is most popular loop. ‘for’ loop allows us to specify three things about a loop in a single line which are:

1. Setting a loop counter to initial value.
2. Testing the loop counter to determine whether its value has reached the number of repetitions desired.
3. Increasing the value of loop counter each time the program segment within the loop have been executed.

```
#include<stdio.h>
#include<conio.h>
void main()
{
float p,r,t,i;
int count;
for(count=1;count<=3;count=count+1)
{
printf(“\n enter principal,rate and time”);
scanf(“%f %f %f”,&p,&r,&t);
i=p*r*t/100;
printf(“\n simple interest =%f”,i);
}
getch();
}
```

The general format of for statement is as us under:

```
for (initialize counter;test counter;increment counter)
{
  do this;
  and this;
  and this;
}
```

let us examine how for statement gets executed:

1. When the for statement is executed for the first time, the value of count is set to an initial value 1.
2. Now the condition count<=3 is tested. Since count is 1 the condition is satisfied and the body of for loop is executed for the first time.
3. Upon reaching the closing brace of for, control is transferred back to the for statement, where the value of count gets incremented by 1.
4. Again the test is performed to check whether the new value of count exceeds 3.
5. If the value of count is still within the range 1 to 3 the statements within the braces of for are executed again.
6. The body of the for loop continues get executed till count doesn’t execute the final value 3.
7. When count reaches the value 4 the control exists from the loop and is transferred to the statement immediately after the body of for.

‘while’ and ‘for’ loops test condition first, if condition is true then they execute statements in loop body but ‘do-while’ executes the statements in loop body then tests condition therefore do-while assures execution of loop body at least once whereas for and while do not assure execution of loop body at least once.

Explain variations in for loops.

1. int i=1;

```
for (;i<=10;)
{
        printf(“\n %d”,i);
        i=i+1;
}
```

output will be 1,2,3 up to 10

2. int i=1;

```
for (;i<=10;)
         printf("\n %d",i);
         i=i+1;
```

Output will be 1,1 infinite no. of times because only printf statement will run and statement to increment value of 'i' does not execute so braces are necessary to execute printf and i=i+1 statements.

3

```
int i;
i=1;
for (;i<=10;i=i+1);
      {
        printf("\n %d",i);
      }
```

Output will be 11 because closing parentheses is followed by ';' which instructs execution of loop without loop body.

4

```
int j,i;
for (i=1,j=1;i<=10;i++,j++)
      {
        printf("\n %d %d",i,j);
      }
```

the output will be 1 1 2 2 3 3 up to 10 10.

Write short notes on odd loop.

The loops in which no. of times statements inside loop body will execute are known as **finite or determinate loop**. The loops which will never terminate execution of statements inside loop body are known as **infinite loop**. The loop about which we are not certain how many times statements inside loop body will execute but we are sure that it will terminate after some time is known as **odd loop**.

```
/* execution of a loop -unknown number of times */
void main()
{
int f=1;
float p,r,t,i;
while(f= =1)
{
  printf("\n enter no. of principal,rate and time");
  scanf("%f %f %f",&,p,&r,&t);
  i=p*r*t/100;
  printf("\n simple interest=%f",i);
  printf("\n calculate interest on different set of p,r and t -enter 1 for y,0 for no");
  scanf("%d",&f);
}
getch();
}
```

In this example calculation of interest for given principle, rate and time continues until user enter 0 for no.

Write short Notes on 'break' in Loop/write program to check primality.

We often come across situations when we want to jump out of a loop instantly, without waiting to get back to the conditional test. The keyword break allows us to do this. When the keyword 'break' is encountered inside any C loop or switch statement , control automatically passed to the first statement

after the loop or switch statement. A break is usually associated with an 'if' statement applicable to loop.
e.g. ***Testing whether a given no. is prime or not.***
The logic behind checking primality of a given value is a follows:
All we have to do to test whether a number is 'prime' or not, is to divide it successively by all number from 2 to one less than itself. If remainder of any of these divisions is zero, the number is not a prime otherwise it will be prime.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int num,i;
printf("enter a no.");
scanf("%d",&num);
i=2;
while(i<=num-1)
{
 if (num % i = =0)
      break;
i=i+1;
}
if (i= = num)
  printf("\n prime");
else
 printf("\n not prime");
}
```

in this program when num % i becomes zero. The condition of 'if' statement becomes true and break statement runs which causes control to come out of loop.
There are two ways the control could have reached outside the while loop:

1. *It jumped out because the num % i became zero and break statement ran.*
2. *It jumped out because value of 'i' reached to 'n' resulting loop condition to false.*

The keyword break takes the control out of the loop inside which it is placed. Consider the following program which illustrates this fact.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i=0,j=0;
while(i<=10)
 {
    i++;
    j=0;
    while(j<=10)
       {
          j++;
          if (j= =5)
              break;
          printf("\ni= %d,j= %d",I,j);
         }
   }
 getch();
}
```

output will be

```
i=1 j=1
i=1 j=2
i=1 j=3
i=1 j=4
i=2 j=1
i=2 j=2
i=2 j=3
i=2 j=4 and so on for value of 'i' from 1 to 10
```

Write short notes on continue statement.

On some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop which have not yet been executed. The keyword 'continue' allows us to do this. When the keyword continue is encountered inside any loop, control automatically transfers to:

1. *Test condition if using 'while' or 'do-while' loop.*
2. *Increment/decrement/update section if using 'for' loop.*

Now consider program to find division of exactly three sets of two nos. if user enters denominator as '0' program will ask to reenter the two nos.

```
#include <stdio.h>
#include <conio.h>
void main()
{
 float a,b;
int i=1;
while(i<= 3)
{
 printf("\n enter %d set of numerator and denominator");
 scanf("%f %f",&a,&b);
 if(b= =0)
     continue;
  printf("\n division =%f ",a/b);
  i=i+1;
}
getch();
}
```

in above program user has to enter exactly three sets of two nos where denominator is not zero. If user enters zero as denominator; user has to re-enter the two nos. because of continue statement's execution, control is transferred to conditional test after bypassing statements: printf and i=i+1.

Continue	***Break***

1.'continue' statement transfers control to condition test portion for 'while' and 'do-while' loop and increment/decrement/update section portion for 'for' loop.	*'break' statement transfers control to the statement which just follows body of loop.*
2. control does not go out of loop.	*'break' statement causes termination of loop inside which break statement falls.*

State the difference between continue and break?

Write short notes on switch statement/ what will happen if break is absent in switch statement?

The control statement which allows us to make a decision from the number of choices is called a switch, or more correctly a switch-case-default. Since these three keywords go together to make up the control statement. They most often appear as follows:

```
switch(integer or character expression)
{
case constant1:
          do this;
           and this;
case costant2:
           do this;
           and this;
}
```

the integer expression following the keyword switch is any 'c' expression that will generate an integer value. It could be an integer constant like 1,2,3 or an expression that evaluates to an integer or any character constant .

The keyword case is followed by an integer or a character constant. Each constant in each case must be different from all the others. The "do this and this" lines mentioned above are replaced in actual practice with valid 'c' statements.

What happens when we run a program containing a switch? First, the integer expression following the keyword switch is evaluated. The value it gives is then matched, one by one, against the constant values that follow the case statement. When a match is found, then program executes the statements following that case, and all subsequent case and default statements as well. If no match is found with any of the case statement, only the statements following the default are executed. A few examples will show how this control works.

Understanding what happens when break is absent in switch?

```
#include <stdio.h>
#include<conio.h>
void main()
{
int i =2;
switch(i)
{
  case 1:
           printf("\n I am in case 1");
  case  2:
           printf("\n I am in case 2"):
  case 3:
           printf("\n I am in case 3");
  default:
          printf("\n I am in default");
}
getch();
}
```

out put:

```
I am in case 2
I am in case 3
I am in default
```

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

```
#include<conio.h>
void main()
{
int i =2;
switch(i)
{
  case 1:
            printf("\n I am in case 1");
            break;
  case  2:
            printf("\n I am in case 2"):
           break;
  case 3:
            printf("\n I am in case 3");
         break;
  default:
           printf("\n I am in default");
}
getch();
}
```

out put:

```
I am in case 2
```

If you want that only case 2 should be executed, it is up to you to get out of the control structure then and thereby using a break statement.

Consider following example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char ch;
printf("\n enter a alphabet a,b or c");
scanf("%c",&ch);
switch(ch)
{
case 'a':
case 'A':
      printf("\n a for apple");
      break;
case 'b':
case 'B':
      printf("\n b for ball");
       break;
case 'c':
case 'C':
      printf("\n c as cat");
      break;
default:
 printf("\n wish you knew what are a,b and c");
}
getch();
}
```

Here, we are making use of the fact that once a case is satisfied the control simply falls through the case till it doesn't meets a break statement. That is why if an alphabet a is enter the case 'a' is satisfied *and since there are no statements to be executed in this case the control automatically reaches the next case i.e. case 'A' and executes all the statements in this case and comes out of the control structure due to break statement.*

State the difference between switch and if

switch	***if-else***
1.All programs made using switch can be solved using if	*All programs made by 'if' are not possible to solve by 'switch'.*
2. 'switch' can not have case like ***case i<=20***	*'if' can have condition like* ***i<=20 && i>=10****.*
3. 'switch' leads to more structured programming and level of indentation is manageable even if we have nested switch.	*'if' leads to less structured programming and level of indentation is difficult to manage if using nested 'ifs'.*

How can we get input and output of string/ what is the difference between gets and scanf for character array?

We can get input and output of string in following ways

#include<stdio.h>	*#include<stdio.h>*	*#include<stdio.h>*
#include<conio.h>	*#include<conio.h>*	*#include<conio.h>*
void main()	*void main()*	*void main()*
{	*{*	*{*
char a[40];	*char a[40];*	*char a[40];*
printf("\n enter a string");	*printf("\n enter a string");*	*printf("\n enter a string");*
scanf("%s",a);	*gets(a);*	*gets(a);*
printf("\n string is %s",a);	*printf("\n string is %s",a);*	*puts("string is ");*

getch();
 	
}
	getch();
	
	}
		puts(a);
		
		getch();
		
		}

difference between gets and scanf*: gets' can read multiple words but scanf can not read multiple words. e.g. input "surendra verma" for program in first column then output will be "surendra" only the reason behind this is scanf("%s",a) stops scanning when it meets space.*
input "surendra verma" for program in second column then output will be "surendra verma"

Third column explain uses of 'puts' library function ,puts library function accepts string or character array and prints it in new line.

What do you mean by general Input/Output?
General input output means how we can read values for variable or how we can display value of constants and variables in desired format.
Consider following examples of general output facilities offered by printf.
printf with formatting(output formatting):-

```
#include<stdio.h>
#include<conio.h>
void main()
{
float f_var=10.12576893;
int wid_prec;
/* output 1: Ordinary output.*/
printf("Output 1:%f\n",f_var);
/*output 2: Output in a field of width 20 (direct specification)*/
printf("Output 2: %20f\n",f_var);
/*output 3: same as above, but left justified.*/
printf("Output 3: %-20f\n",f_var);
/*explicit sign display*/
printf("Output 4: %+f\n",f_var);
/*Output 5: Indirect width specification (Compare Output 2)*/
printf("\n enter width");
scanf("%d",&wid_prec);
printf("Output 5: %*f\n",wid_prec,f_var);
/* output 6 precision specification 5 digits with rounding */
printf("Output 6: %.5f\n",f_var);
/* output 7 precision specification 5 digits indirect with rounding */
printf("\n enter precision");
scanf("%d",&wid_prec);
printf("Output 7: %20.*f\n",wid_prec,f_var);
/*output *: a combination of field width of 20,left justify,explicit sign, precision of 5 */
printf("Output 8: %+20.5f\n",f_var);
getch();
}
```

scanf width specifier and input formatting :-

```
#include<stdio.h>
#include<conio.h>
void main()
{
char string1[40],string2[40];
printf("enter two strings");
scanf("%4s %5s",string1,string2);
printf("\n the two input strings are %s %s",string1,string2);
getch();
}
```

for input of two string “morning” and “comes” output will be “morn” and “ing”.

What are the different types of characters which can appear in format string?

Different types of characters are:
White space characters (tab, space , new-line), ordinary characters and format specifier.

What do you mean by search sets?

When reading values for variables we want to separate the values being input by some separator characters; those separator characters constructs search set. Search set are set of characters enclosed in square brackets []. These search sets are parts of some format specifier. for example , the format specifier %[/-].
This tells ‘scanf’ to match any sequence of characters consisting of / and -. All these characters are assigned to the corresponding argument. This argument must be the name of character array. Let us see how search sets are used.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int date,month,year;
char sep[2];
printf("\n enter the date:");
scanf("%d%[/-]%d%[/-]%d",&date,sep,&month,sep,&year);
printf("\n date =%d,month=%d,year=%d",date,month,year);
getch();
}
```

If input was 31-13/1999
The output will be
date=12 month=12 year=1999

Explain assignment suppression character./what is the purpose of assignment suppression character?

Till now we were having one argument for every format specifier in the format string. Sometimes this is unnecessary, as shown in above example. Using an assignment suppression character tells the ‘scanf’ that input field should only be scanned and not assigned to any argument(variable).
e.g.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int date,month,year;
printf("input the date:");
scanf("%d*[/-]%d*[/-]%d",&date,&month,&year);
printf("\n date =%d,month=%d,year=%d",date,month,year);
getch();
}
```

if input was 31-13/1999
the output will be

```
date=12 month=12 year=1999
```

we can note there are 5 format specifiers but only 3 argument(variables).
The corresponds are shown as

Format specifier	*Argument*
%d	*&date*
%[/-]*	*(no argument)*
%d	*&month*
%[/-]*	*No argument*
%d	*&year*

What is use of sizeof Operator?
'sizeof' operator allows to find out memory taken in no. of bytes by a variable or data type as illustrated.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a;
float b;
printf("\n no. of bytes taken by an integer=%d",sizeof(a));
printf("\n no. of bytes taken by an integer=%d",sizeof(int));
printf("\n no. of bytes taken by an integer=%d",sizeof(float));
getch();
}
```

in this example output would be 2 2 and 4 because int takes 2 bytes and float takes 4 bytes.

What is use of const keyword (qualifier)?
const keyword is useful to declare a constant with a given symbolic name whose value can not change during program run time.

```
#include<stdio.h>
#include<conio.h>
const float pi=3.1415;
void main()
{
float area,r;
printf("\n enter radius of circle");
scanf("%f",&r);
area=pi*r*r;
printf("\n area=%f",area);
getch();
}
```

What is output of following statement-viva?
int i=0;
printf("\n %d %d",i,i++);
output will be: 1 0

What is return value of scanf?
Answer
(a)1 for successful scanf,
(b)0 for unsuccessful scanf or encountering end of file
(c)Nothing if program indicates abnormal termination.

What is output of following statement?
char name[]=" ";
scanf("%c",name);
printf("%s",name);
output will be : any character equivalent of address of array due to %c
and space due to %s.

What do you mean by nested loop?
nested loop is a special case in which one loop contains other loop inside its loop body. for each value of outer loop counter , inner loop counter changes frequently.
e.g. ***program to print reverse pyramid***

```
#include<stdio.h>
#include<conio.h>
void main()
{
int m,n;
for(m=5;m>=1;m--)
{
for(n=1;n<=5-m;n++)
  printf(" ");
for(n=1;n<=2*m-1;n++)
  printf("*");
printf("\n");
}
getch();
}
```

Write program to evaluate 1/!1+2/!2+3/!3+…

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

```
void main()
{
float term,m,s=0;
int n;
printf("\n how many terms to evaluate");
scanf("%d",&n);
term=1;
for(m=1;m<=n;m++)
{
printf("\t %f",term);
s=s+term;
term=term/m;
}
printf("\n sum=%f",s);
getch();
}
```

Define Array.
Array or Subscripted variables:
Array is collection of variables having common name and common data type. Individual variable in collection is identified by index or subscript. Elements of array occupy contiguous memory location.

How can we declare a single dimension Array?
Declaration (single dimension numeric array):
int a[5];
'a' is name of array and 'a' denotes address of first element of array known as base address and 'a' is constant.
it means there are 5 variables viz. a[0],a[1],a[2],a[3],a[4] index or subscript will change from 0 to 4. All variables will be integers.
float b[10];
It means there are 10 variables viz. a[0],a[1],a[2],a[3] ....a[9] index or subscript will change from 0 to 9. All variables will be floats.

How we can initalise an array?
Initialization of single dimension array (numeric):
int a[5]={2,1,3,4,6};
which means a[0]=2,a[1]=1,a[2]=3,a[3]=4,a[4]=6;

int a[]={2,1,3,4,6};
which means a[0]=2,a[1]=1,a[2]=3,a[3]=4,a[4]=6; and size of array will be 5. index will range from 0 to 4.
int a[5]={2,1,3};
which means a[0]=2,a[1]=1,a[2]=3,a[3]=0,a[4]=0; and size of array will be 5. index will range from 0 to 4.
memory scheme of array of integers having 5 elements

A[0]	*A[1]*	*A[2]*	*A[3]*	*A[4]*

100	*102*	*104*	*106*	*108*

memory scheme of array of floats having 5 elements

A[0]	*A[1]*	*A[2]*	*A[3]*	*A[4]*

100	*104*	*108*	*112*	*116*

following is not correct:
int a[3]={2,1,3,4};
This will result in compile time error.

What special keyword is used in initialization of character arrays in c?
ans: static

What are the three different ways to assign values to string variables?
ans :
char a[]=”ramesh”;
char a[]={‘r’,’a’,’m,’e’,’s’,’h’,’\0’};
strcpy(a,”ramesh”);

What is a null statement?
null statement is a statement terminated by ; without any statement.

What are the usages/Applications of Array?
Application of arrays:
* It is useful to solve simultaneous equation.
* It is useful to solve problems of data structure like searching, sorting, linked list, graph etc.
* Its is useful to solve such problems requiring variable of same data type in contiguous memory location.

If an array is declared as int a[5] can we access element a[5]?
No. if an array is declared as int a[5] then index will range from 0 to 4, element with index does not exist.
Don’t access array element which doesn’t exist.
```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5];
a[6]=11;
printf(“\n value of element with index 6 =%d”,a[6]);
getch();
}
```

Above program will not generate any compile error but it may result the error ‘program has performed an illegal operation’ if window 9x session is active.

What does ‘a’ means if an array is declared as int a[5]?

Name of address denotes address of first element of address known as base address of array. Base address of an array is a constant-pointer or pointer-constant.

```
#include<stdio.h>
#include<conio.h>
void main()
{ int a[5];
printf(“\n base address of  array =%u”,a);
getch();
}
```

Write program to input an array of integers having 5 elements and find its sum.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5],i,sum=0;
printf(“\n enter 5 values for array a:”);
for (i=0;i<5;i++)        /* input routine for array elements */
  scanf(“%d”,&a[i]);
for(i=0;i<5;i++)       /* sum the values of array elements */
  sum=sum+a[i];
printf(“sum=%s”,sum);
getch();
}
```

What is string ? How character array can be declared and initialized?

Collection of characters having last character null character (‘\0’) is known as string. a single character variable can store a single character enclosed in single quote. Sometimes it is required to store name, address and other alphanumeric or special characters; in that case pointer to character or character array is the only solution.

Declaration:

char a[40];

a is a character array having no. of elements 40 and index varying from 0 to 39. we can store maximum 39 character leaving one space for null character.

Initialization:

char a[40]=”ramesh” ;

or

char a[40]={‘r’,’a’,’m’,’e’,’s’,’h’,’\0’};

or

char a[]=”ramesh”; /* in this case array size will be 7(including hidden null character) and index will range from index 0 to 6 */

program to initialize, input and print the strings.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 char a[]=”ramesh”;  /* initialization of character array */
 char b[40];
 printf("\n enter a string:");
 scanf("%s",a); /* do not use &a */
 printf("\n first string is %s,second string is %s",a,b);
 getch();
}
```

Write program to find rotation of given string.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char a[20];
int i,n,j;
int len;
printf("\n enter a string");
scanf("%s",a);
len=strlen(a);
clrscr();

for(n=1;n<=len;n++)
   {
    printf("\n");
    for(j=0;j<len;j++)
      printf("%c",a[(j+n)%len]);

   }
getch();
}
```

Write program to find whether a word/number is palindrome or not?

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[40];
char t;
int m,n;
printf("\n enter word/number to check palindrome");
scanf("%s",a);
/* count no. of characters */
for(m=0;a[m]!='\0';m++);
/* position m to last physical character */
m=m-1;
/* compare first and first from last side character and so on */
for(n=0;n<=m;m--,n++)
 {
 if(a[n]!=a[m])
  break;
}
if (n<=m)
  printf("\n not palindrome");
else
printf("\n palindrome");
getch();
}
```

Write program to find greatest no. in a list of values.

```
#include<stdio.h>
#include<conio.h>
void main()
{
float a[5];
int i,max,p;
printf("\n enter five values");
scanf("%f",&a[i]);
max=a[0];
for(i=1;i<5;i++)
 if (max<a[i])
{
max=a[i];
p=i;
}
printf("\n largest value=%d at index=%d",max,p);
getch();
}
```

Write program to arrange the alphabets of an entered string in alphabetical order.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[80],t;
int n,m,len;
printf("\n enter a string");
gets(a);
/* count no. of characters in string */
for(len=0;a[i]!='\0';len++);

/* use bubble sort */
for(m=0;m<len-1;m++)
for(n=0;n<len-1-m;n++)
  {
   if(a[n]>a[n+1])
     {
      t=a[n];
     a[n]=a[n+1];
     a[n+1]=t;
   }
}
printf("\n sorted string is %s",a);
getch();
}
```

Write a program to copy one string to other string without using library function?

```
#include<stdio.h>
#include<conio.h>
void main()
{
```

```
char a[40];
int
printf("\n enter the string");
gets(a);

#include<stdio.h>
#include<conio.h>
void main()
{
char a[20],b[20];
int j;
printf("\n enter first string");
gets(a);
for(j=0;a[j]!='\0';j++)
b[j]=a[j];
b[j]='\0';
printf("\n string copied to b as %s",b);
getch();
}
```

What is Multi Dimension Arrays? What are their usages?

Array declared with more than one dimension is know as multi-dimension array.

Often multi dimension array is needed to store data for table and matrices. Multi -dimension array has more than one subscript. Two dimension array contains two subscript, three dimension array has three subscript and so on.

Understanding declaration, initialization

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][2]; /* declaration of  2d array having total no. of elements 3*2=6 */
int b[3][3]={2,1,3,2,4,5,6,7,8};
int i,j;
printf("\n enter 6 values for array a"); /* input routine for array of 3x2 */
for(i=0;i<3;i++)
for (j=0;j<2;j++)
    scanf("%d",&a[i][j]);

/* print contains of array a then array b*/
printf("\n contents of array a:");     /* output routine for array 3x2 */
for(i=0;i<3;i++)
{
for (j=0;j<2;j++)
    printf("\t%d",&a[i][j]);
printf("\n");
}

printf("\n contents of array b:");

for(i=0;i<3;i++)
{
for (j=0;j<3;j++)
    printf("\t%d",b[i][j]);
printf("\n");
}
getch();
}
```

Understanding initialization of array b

	Column index 0	Column index 1	Column index 2

Row index 0	2	1	3

Row index 1	2	4	5

Row index 2	6	7	8

Write program to add, multiply two matrix of 3x3

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][3],b[3][3],c[3][3];
int m,n,p;
printf("\n enter 9 values for matrix a");
for(m=0;m<3;m++)
for(n=0;n<3;n++)
 scanf("%d",&a[m][n]);

printf("\n enter 9 values for matrix b");
for(m=0;m<3;m++)
for(n=0;n<3;n++)
 scanf("%d",&b[m][n]);

printf("\n result of matrix addition");
for(m=0;m<3;m++)
{
for(n=0;n<3;n++)
{
c[m][n]=a[m][n]+b[m][n];
printf("\t %d",c[m][n]);
}
printf("\n");
}

printf("\n result of matrix multiplication");
for(m=0;m<3;m++)
{
for(p=0;p<3;p++)
 {
 c[m][p]=0;
for(n=0;n<3;n++)
{
c[m][p]=c[m][p]+a[m][n]*b[n][p];
}
printf("\t %d",c[m][p]);
}
printf("\n");
}
getch();
}
```

Write program to transpose matrix of 4x4 order

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[4][4],c[4][4];
int m,n;
printf("\n enter 16 values for matrix a");
for(m=0;m<4;m++)
for(n=0;n<4;n++)
 scanf("%d",&a[m][n]);

for(m=0;m<4;m++)
for(n=0;n<4;n++)
 c[n][m]=a[m][n];
printf("\n result of matrix transpose");
for(n=0;n<4;n++)
{
for(m=0;n<4;m++)
{
printf("\t %d",c[n][m]);
}
printf("\n");
}
getch();
}
```

Write program to sum diagonal elements of square matrix of 3x3 order

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][3];
int m,n;
int fd=0,bd=0;
printf("\n enter 9 values for matrix a");
for(m=0;m<3;m++)
for(n=0;n<3;n++)
 scanf("%d",&a[m][n]);

for(m=0;m<3;m++)
{
fd=fd+a[m][m];
bd=bd+a[m][2-m];
}
printf("\n sum of forward diagonal=%d,backward diagonal=%d",fd,bd);
getch();
}
getch();
}
```

Write program to make diagonal elements of square matrix of 3x3 order to zero

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][3];
int m,n;
printf("\n enter 9 values for matrix a");
for(m=0;m<3;m++)
for(n=0;n<3;n++)
 scanf("%d",&a[m][n]);

for(m=0;m<3;m++)
{
a[m][m]=0;
a[m][2-m]=0;
}
printf("\n result of making diagonal elements zero");
for(n=0;n<3;n++)
{
for(m=0;n<3;m++)
{
printf("\t %d",a[n][m]);
}
printf("\n");
}
getch();
}
```

Write program to sum row elements and column element of matrix of order 3x3

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][3];
int m,n;
int rs=0,cs=0;
printf("\n enter 9 values for matrix a");
for(m=0;m<3;m++)
for(n=0;n<3;n++)
 scanf("%d",&a[m][n]);


for(m=0;m<3;m++)
{
cs=0;
for(n=0;n<3;n++)
cs=cs+ a[m][n];
printf(“\n column sum=%d”,cs);
}
for(m=0;m<3;m++)
{
rs=0;
for(n=0;n<3;n++)
rs=rs+ a[n][m]; /* index interchanged */
printf(“\n row sum=%d”,rs);
}
getch();
}
```

How can we declare and initialize?
An array of strings is collection of strings which can be stored by double dimension array of characters.
Declaration of array of strings or double dimension character array:
char a[5][20];
it means that a is capable of holding 5 strings each having 19 characters maximum, leaving one space for null character.
Intialization of array of strings
char a[5][20]={“ram”,”shyam”,”ajay”,”vijay”,”murli”};
internal storage of character constant would be as follows:

Row/index	0	1	2	3	4	5	….	…	19

0	‘r’	‘a’	‘m’	‘\0’					

1	‘s’	‘h’	‘y’	‘a’	‘m’	‘\0’			

2	‘a’	‘j’	‘a’	‘y’	‘\0’				

3	‘v’	‘i’	‘j’	‘a’	‘y’	‘\0’			

4	‘m’	‘u’	‘r’	‘a’	‘l’	‘i’	‘\0’		

Program on array of string input and printing the array of strings.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[5][20];
int i;
printf(“\n enter 5 strings”);
for (i=0;i<5;i++)
  scanf(“%s”,a[i]); /* input routine for array of strings */
printf(“\n the strings are \n”);
for(i=0;i<5;i++)
 printf(“\n %s”,a[i]);
getch();
}
```

Write program to demonstrate use of string library functions supported by string.h/ write any 5 string handling library function?

```
/* note header files can be included in any order */
#include<string.h>
#include<conio.h>
#include<stdio.h>
void main()
{
 char a[20],b[20];
 int i;
 /* demo of string copy function ‘strcpy’ used to copy one string to other follows now.
strcpy(destination-single dimension-character-array, source-single dimension character array or
string).
*/
printf(“\n enter a string :”);
scanf(“%s”,a);
/* now copy contents of character array a to b */
strcpy(b,a);  /* destination is given first */
printf(“\n copied string into b is %s”,a);

/* demo of string concatenation (string addition ) begins now */
/*  strcat(character array to add in, string or character array need to add )*/
printf(“\n enter first string “);
scanf(“%s”,a);
printf(“\n enter second string”);
scanf(“%s”,b);
/* note: when contents of character array (string) ‘b’ is added to ‘a’; capacity of array ‘a’ must be
such that it is able to accommodate both contents of array ‘a’ and ‘b’ */
strcat(a,b);
printf(“\n new contents of array is %s”,a);


/* demo of  string comparisons begins now
   two string can not be compared using >,<,>=,<= but we have to use strcmp for comparison. When
comparing two strings if first string comes first in alphabetical order then it will be smaller than
second string which comes later in alphabetical order. Never think no. of characters decides which
string is smaller and which string is greater.
strcmp returns value >0 if string1 comes later when compared to string2 in alphabetical order
strcmp returns value =0 if string1 and string2 are exactly equal.
strcmp returns value<0 if string1 comes before when compared to string2 in alphabetical order
for case insensitive comparison of strings use strcmpi in place of strcmp syntax is similar. E.g. if  string
“ram” is contained in character array ‘a’ and “Ram” is contained in character array ‘b’ strcmp will
give return value >0 because letter ‘r’ has greater ascii value when compared to letter ‘R’ but strcmp
will give return value 0  */
printf(“\n enter first string “);
scanf(“%s”,a);
printf(“\n enter second string to compare”);
scanf(“%s”,b);
i=strcmp(a,b);
if (i<0)
  printf(“\n first string is smaller”);
elseif (i>0)
  printf(“\n first string is greater”);
else
  printf(“\n both the strings are equal”);
/* strlen function gives no. of characters inside a character array or string
  syntax is
  int_variable=strlen(character array or string constant) */
i=strlen("ramesh"); /* or use i=strlen(a); to know no. of characters entered into character array 'a' */
printf("\n no. of characters in string =%d",i);

/* strreverse reverses contents of a character array and returns pointer to reversed string
   syntax is
  strrev(character array or string);
  or
  char *p;
  p=strrev(character array or string) */
printf("\n enter a string to reverse");
scanf("%s",a);
strrev(a);
printf("\n reverse string is %s",a);

/* strlwr converts an string or contents of character array to small letters
   strupr converts an string or contents of character array to capital letter */
printf("\n enter a string to convert to uppercase ");
scanf("%s",a);
strupr(a);
printf("\n string in upper case is %s",a);
strlwr(a);
printf("\n string in lowe case is %s",a);
getch();
}
```

What is function?

a number of statements grouped into a single logical unit is referred to as a function which is made to complete a specific task. A program can be made of many functions but 'main' is the function whose statements are executed first.

Types of functions:

User defined function: *these are those functions which are made by programmer for his programming convenience.*

Library function: *these are those functions which are readymade and are provided by compiler, uses of which are possible by including corresponding header files. Source code of library function is not available to programmer but its object codes are available in precompiled form.*

List out header files in 'c'/write notes on library functions. Why they are needed?

list of header files

we need to include header file in our program to utilize library function, macro definition and structure declaration contained in header file.

though there are many header files in c. some of the quite commonly used header file are discussed here.

#include <stdio.h>

supports printf, scanf, getchar etc.

#include <math.h>

supports some math related function.

A=sqrt(25); now A will have 5

A=pow(3,2); now A will have 9

A=abs(-5); now A will have 5

#include<conio.h>

clrscr(); this statement will clear screen

c=getch(); this will assign a character to variable c
#include<ctype.h>
c=toupper('a'); now c will have uppercase letter 'C'
c=tolower('A'); now c will have lowercase letter 'a'
#include<time.h>
time.h includes clock, ctime, stime, asctime, difftime, time and many more time related library functions.
#include<dos.h>
supports exit etc.
#include<stdlib.h>
supports malloc,calloc,free,realloc functions to dynamically allocate memory
#include<string.h>
support library function strlen,strcpy,strupr etc. for string related operation.

Explain 'main' function.
'main' is a function which is called by operating system when a program is run. 'main' is the foremost function which gets executed even if program is made up of several functions.
Execution of other function: other function executes when the 'main' function calls it directly or indirectly ('main' function calls a function and in turn the called function calls another function and so on). Each program must contain a function 'main' otherwise program will not compile.
e.g.

```
#include<stdio.h>
void main()
{
 printf(" welcome to c programming");
}
```

This program contains a preprocessor #include directive, a function heading void main() where **void indicates function does not return a value, a function body inside curly braces,** a printf statement to print message 'welcome to c programming'.

What are the usages/applications of functions?
Usages are as follows:

1. *Function avoids code repetition:- once a function has been made we can call the function to complete the task when and where necessary.*
2. *Function used carefully makes the process of error removing easier called **debugging**.*
3. *Once function is tested and satisfies the one's needs, one can convert it in to library, and can distribute it others for use.*
4. *Function allows breaking bigger task into smaller manageable subtasks which is soul of modular programming.*
5. *Recursive function (function calling itself at least once) solves some typical programming tasks easily and with few lines of coding which otherwise would have taken several lines of code.*

What is the format of function?
format of function is as follows:

```
<return data type> function-name (datatype var1,datatype var2…)
{
statement1.
statement2.
<return value/variable>
}
```

Define various key concepts/components applicable to function.

Let us explore the various key concepts associated with function using a simple example:
Program to add two numbers(function using more than one parameter/argument)/ Function returning value and function taking argument.

```
1.  #include<stdio.h>
2.  #include<conio.h>
3.  void main()
4.  {
5.  int a,b,c;
6.  int add(int,int);
7.  printf("\n enter two values:");
8.  scanf("%d %d",&a,&b);
9.  c=add(a,b);
10. printf("\n addition=%d",c);
11. getch();
12. }
13. int add(int x, int y)
14. {
15.  int z;
16.  z=x+y;
17.  return z;
18. }
```

(a) function declaration or prototype(line no 6):
function prototype or function declaration is as follows
int add(int ,int);
int add(int x,int y);
int add(int,int y);
int add(int x,int);
all these are applicable.
It means that name of variables or optional and if we like we can leave these.
***int** indicates data type of the value that the user define function will return.*
If we use function declaration as :
int add(int x,int y);
It does not mean that we can use variable x and y inside main function without declaring these variables.
Similarly if we use function declaration as :
int add (int a,int b);
It does not mean that we can use variable a and b inside main function without declaring these variables.
If we use function declaration as :
int add(int x,int y);
*then x and y are know as **dummy/formal parameter/argument.***
Following declaration or prototyping is not allowed
int add(int x,y); this will generate error.
(b) function call(line no. 9):
Function call statement is as follows:
c=add(a,b);
following are incorrect way of calling function:
c=add(int a,b);
c=add(int a,int b);
c=add(int ,int b);
etc.
it means that while calling function never prefix data type just give name of variable or give constant or combination of constant or variable or any mathematical expression
e.g.

c=add(3,a);
c=add(a,b);
*c=add(4*5+3,b);*
etc.
name of variable or constant or expression given here is know as ***actual parameter/argument.***
The function call statement causes control to be transferred to user defined function and value of variable b is copied to variable z and value of variable a is copied to x (right to left). Function stores addition of variable x and y to z and value of z in turn is assigned to variable c in main function. Now statement followed by function call inside main starts execution.
(c) Function header line or function declarator (line no. 13):
It could be written as follows:
int add(int x,int y) / y will receive value of main's b and x will receive value of main's a */*
int add(int a,int b) / b will receive value of main's b and a will receive value of main's a */*
int add(int x,int b) / b will receive value of main's b and x will receive value of main's a */*
etc.
but followings are incorrect:
int add (int x,y) / correct way is int add(int x,int y) */*
int add(int,int)   / variable names are not optional here */*
int add(int x,int y); / don't use semicolon here */*
if we find in some books

```
int add(a,b)
int a,b;
{
----
----
}
```

we must change this to:

```
int add(int a,int b)
{
 --
 --
}
```

(c) Function body(line no. 14 to 18)
Function body is made up of line no. 14 to 18. function body is surrounded by brace.
Function body contents the statements which are necessary to perform the task for which function is made. ***Function body and function header line makes function definition.***
(d) Function return value(line no. 17):
This statement is useful to send value calculated by called function to calling function.
Calling function can send value of variables or constant using actual argument when function is called.
Called function can send either one value(if function has return type) or zero value(if function has no return type) using return statement to calling function.
Calling function is the function which calls another function in our example it is main. Called function is the function which is called by calling function.
Following points are note worthy:
(i)We can not use variables declared in called function inside calling function without declaring them in called function.
(ii) We can not use variable declared in called function in calling function without declaring them in calling function.
(iii) If function header line contents :
int add(int x,int y) then don't declare int x,y inside function body of called function otherwise it will generate error. Don't assign new value to variable x and y otherwise you will loose values passed by actual arguments during function call.
(iv) Assigning new value to variables x and y will not change value of variable a and b in our example(call by value).

*(v) **Omission of function prototype or declaration**: if function definition occurs before call of function it is possible to remove function declaration or function prototype.*
***Function prototype can be global**. If function declaration is global any function can call the user defined function.*
to do so –
Put function declaration outside curly brace of any function's body.
***Function prototype can be local**. If function declaration appears inside function body of any function, function prototype becomes local in such case only the function whose function body contents the function declaration can call the user defined function.*

What is function parameter/differentiate actual and formal parameter?
Function parameters are means of communication between the calling and the called function. They can be classified into formal parameters and actual parameters. The formal parameters are the parameter given int the function declaration and function definition. The actual parameters or actual arguments, are specified in the function call statement.
While using parameters followings points are important:

1. *the number of arguments in the function call and function declarator and function declaration must match.*
2. *the data type of each of the arguments in the function call should be the same as corresponding parameter in the function header-line (function-declarator) and function declaration.*

What is function definition?
Function definition contents function declarator(or function header line) and function body. Function body includes all those statements which will complete the task for which function is made.

What is function declaration?
Function declaration specifies return data type of function, name of function and data type of arguments, if any. Function declaration is necessary if function is called first and defined later. Number of arguments/parameters and corresponding data type must match when calling function and when writing function declarator.
Function declaration can be dropped if definition of function appears before function call. Function declaration can be **global or local**. If function declaration appears inside curly brace of any function body it is known as **local function declaration** in such case the function whose function body contains function declaration can call the user defined function. If function declaration does not fall inside curly brace every function can call the user defined function and function declaration is known as **global function declaration**.

What is return statement?
Return statement is means of communication between called function and calling function as the return statement meets control is transferred to calling function.
(a)return statement can be used to pass a single value to calling function if function returns a value. return statement will take form:
return variable; (or return constant;)
(b)return statement can be used to return the control to the calling function prematurely .
(c)if function does not return any value then return statement will take form
return ;
(d) a function body can use return statement more than one times.

What do you mean by parameter/argument passing technique? Or what do you mean by call by value or call by reference or differentiate call by value and call by reference. or Explain passing value between function.
'C' provides two mechanisms to ass arguments to a function
(i)pass arguments by value, and

(ii)pass arguments by address or pointer or reference.
Ordinary variable (not array) is always passed by value which states that value of actual argument can not be altered by called function.
Array is always passed by reference therefore any changes made to value of elements of an array inside called function will change value of actual parameter.
If it is required to change value of actual argument which is ordinary variable(not array) we need to pass its address or pointer which is know as call by reference.
Consider a example of swapping two numbers:
(i) call by value

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=3,b=4;
void swap(int,int);  /* function declaration */
printf("\n before call of function swap value of a and b inside main %d %d",a,b,);
swap(a,b);
printf("\n after call of function swap value of a and b inside main %d %d",a,b,);
getch();
}
void swap(int a,int b)
{
 int t;
 printf("\n before swapping  value of a and b inside called function %d %d",a,b,);

t=a;
a=b;
b=t;
printf("\n after swapping inside called function value of a and b %d %d",a,b,);
}
```

program will give output:

```
before call of function swap value of a and b inside main 3 4
before swapping  value of a and b inside called function 3 4
after swapping inside called function value of a and b 4 3
after call of function swap value of a and b inside main 3 4
```

clearly value of actual argument did not change to 4 3.
(ii) call by reference or address or pointer

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=3,b=4;
void swap(int*,int*);  /* function declaration */
printf("\n before call of function swap value of a and b inside main %d %d",a,b,);
printf("\n before call of function swap address of a and b inside main %u %u",&a,&b,);

swap(&a,&b);
printf("\n after call of function swap value of a and b inside main %d %d",a,b,);
printf("\n after call of function swap address of a and b inside main %u %u",&a,&b,);

getch();
}
void swap(int *a,int *b)
{
 int t;
 printf("\n before swapping  value of a and b inside called function %d %d",*a,*b,);

t=*a;
*a=*b;
*b=t;
printf("\n after swapping inside called function value of a and b %d %d",*a,*b,);
}
```

program will give output:

```
before call of function swap value of a and b inside main 3 4
before call of function swap address of a and b inside main 3210 3212

before swapping  value of a and b inside called function 3 4
after swapping inside called function value of a and b 4 3

after call of function swap value of a and b inside main 4 3
after call of function swap address of a and b inside main 3210 3212);
```

clearly value of actual argument changed to 4 3.

How can we pass single dimension numeric array to function?

Let us see the technique:

```
#include<stdio.h>
#include<conio.h>
#define size 5
void main()
{
int a[size]={3,2,1,5,7};
void print(int a[]);  /* we can write void print(int []); or we can write void print(int [5]);
or we can write void print(int a[size]); */
int sum(int []);
int s;
print(a); /* call of function */
s=sum(a);/* call of function */
printf("\n sum=%d",a);
getch();
}
void print(int x[])      /* note no semicolon here */
{
int i=0;
printf("\n the values are:");
for(i=0;i<size;i++)
  printf("\n %d",x[i]);
}
int sum(int a[])      /* note no semicolon here  */
{
int i,tot=0;
printf("\n the values are:");
for(i=0;i<size;i++)
  tot=tot+a[i];
return tot;
}
```

explanation: print function uses loop to print value of elements of array if name of parameter in function header line is 'x' then printing will be done using x[i]. if we had used name of parameter in function header line as 'a' then printing will be done using a[i].

sum function when called gets total of the values of elements of array and stores it in variable tot. ‘return’ statement passes value of variable tot to function main. main function receives value in sum and prints value of sum.

How can we pass single dimension character array or string to function?

Let us see the technique:

```
#include<stdio.h>
#include<conio.h>
#define size 20
void main()
{
char a[size];
void input(char a[]);
void print(char []);
input(a); /* call of function */
print(a); /* call of function */
getch();
}
void print(char x[])      /* note no semicolon here */
{
printf(“\n the string is:”,x);
}
void input(char a[])      /* note no semicolon here  */
{
printf(“\n enter a string:”);
gets(a);
}
```

explanation: input function inputs value for character array that is string. Since it is a single dimension character array therefore loop is not required. print function when called prints the string stored by input function when called.

Write a function to accept 10 characters and display whether each input character is digit, uppercase letter or lower case letter.

```
#include<stdio.h>
#include<conio.h>
void main()
{
void disp();
disp();
getch();
}
void disp()
{
int m;
char c;
for (m=1;m<=10;m++)
{
printf(“enter character %d”,m);
scanf(“%c”,&c);
if (c>=65 && c<=90)
  printf(“\n uppercase letter”);
else if (c>=97 && c<=122)
printf(“\n lower case letter”);
else if (c>=48 && c<=57)
```

```
 pritf("\n digit");
else
printf("\n it is neither an alphabet nor digit");
}
}
```

How can we pass double dimension numeric array to function?

Let us see the technique:

```
#include<stdio.h>
#include<conio.h>
#define row 3
#define col 3
void main()
{
int a[row][col],b[row][col],c[row][col];
void input(int a[][col]); /* specifying col is must here, row is optional, variable name is optional here */
void print(int [][col]); /* specifying col is must here, row is optional, variable name is optional here */
void add(int c[][col],int a[][col],int b[][col]); /* specifying col is must here, row is optional, variable
name is optional here */
input(a); /* get input for array a , note during call simply a is written not bracket is there */
input(b); /* get input for array b */
add(c,a,b); /* add contents of array a and b to c */
print (c);
getch();
}
void  input(int a[][col])      /* note no semicolon here, variable name is compulsory */
{
 int i,j;
 printf("\n enter %d values\n",row*col);
 for(i=0;i<row;i++)
    for(j=0;j<col;j++)
          scanf("%d",&a[i][j]);
}

void print(int a[][col])      /* note no semicolon here  */
{
 int i,j;
 printf("\n values of array \n");
 for(i=0;i<row;i++)
   { for(j=0;j<col;j++)
      printf("\t%d",a[i][j]);
     printf("\n");
 }
}
void add(int c[][col],int a[][col],int b[][col])
{
 int i,j;
 for(i=0;i<row;i++)
    for(j=0;j<col;j++)
         c[i][j]=a[i][j]+b[i][j];
}
```

explanation: input function inputs value for numeric double dimension array a and b. when function is called and parameter is passed is array 'a' then whatever values are entered by user are stored in array a. when function is called and parameter is passed is array 'b' then whatever values are entered by user are stored in array ‘b’ because array is always **passed by reference**. Function add when called stores addition of corresponding element of array a and b into corresponding elements of ‘array c’. finally print function prints contents of array c.

How can we pass double dimension character array or array of strings to function?/write program to sort a given list of names.

Let us see the technique:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define row 5
#define col 20
void main()
{
char a[row][col];
void input(char a[][col]); /* specifying col is must here, row is optional, variable name is optional
here */
void print(char [][col]); /* specifying col is must here, row is optional, variable name is optional here
*/
void sort(char [][col]); /* specifying col is must here, row is optional, variable name is optional here
*/
input(a); /* get input for array a , note during call simply a is written not bracket is there */
sort(a); /* sort contents of array a which are strings */
print (a);
getch();
}
void  input(char a[][col])      /* note no semicolon here, variable name is compulsory */
{
 int i;
 /* since it is double dimension character array therefore we will use one loop which will be for change
row */
printf(“\n enter %d strings\n”,row);
 for(i=0;i<row;i++)
      scanf(“%s”,a[i]);
}

void print(char a[][col])      /* note no semicolon here  */
{
 int i,j;
printf(“\n %d strings\n”,row);
 for(i=0;i<row;i++)
    printf(“\n%s”,a[i]);
}
void sort(char c[][col])
{
 int i,j;
 char t[col];  /* used for swapping the strings */
 for(i=0;i<row-1;i++)
    for(j=0;j<row-1-i;j++)
         if(strcmp(a[j],a[j+1])>0)
          {
             strcpy(t,a[j]);
             strcpy(a[j],a[j+1]);
             strcpy(a[j+1],t);
```

```
        }
}
```
explanation: input function inputs value for double dimension character array a. when function is called and parameter is passed array 'a' then whatever values are entered by user are stored in array a. Function sort when called sorts the strings stored by input function and print function prints the sorted string.

What are the different categories of functions?

Function can be of four categories:

1. *return and argument*
2. *no return and argument*
3. *no return and no argument*
4. *return and no argument.*

Let us understand each category with example of addition of two nos. using function.

1. *return and argument*

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
int add(int,int);
printf("\n enter two values:");
scanf("%d %d",&a,&b);
c=add(a,b);
printf("\n addition=%d",c);
getch();
}
int add(int x,int y)
{
int z;
z=x+y;
return z;
}
```

2. *no return and argument*

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
void add(int,int);
printf("\n enter two values:");
scanf("%d %d",&a,&b);
add(a,b);  /* c=add(a,b); is not valid because function does not return a value */
getch();
}
void add(int x,int y)
{
int z;
z=x+y;
printf("\n result of addition=%d",z);
}
```

3. *no return and no argument*

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

```
void main()
{
void add(); /* or void add(void); */
add();   /* function call */
getch();
}
void add(void)   /* or void add() */
{
int a,b,c;
printf("\n enter two values:");
scanf("%d %d",&a,&b);
c=a+b;
printf("\n result of addition=%d",c);
}
```

4. *return and no argument*

```
#include<stdio.h>
#include<conio.h>
void main()
{
int c;
int add(); /* or int add(void); */
c=add();
printf("\n addition=%d",c);
getch();
}
int add(void)   /* or void add() */
{
int a,b,c;
printf("\n enter two values:");
scanf("%d %d",&a,&b);
c=a+b;
return c;
}
```

Compare different categories of functions?

Comparison is show in following table:

Return and argument	*No return and argument*	*No return and no argument*	*Return and no argument*
***Input of variables** are done in calling function.*	***Input of variable** are done in calling function.*	***Input of variables** are done in called function.*	***Input of variables** are done in called function.*
***Output of result** is shown in calling function.*	***Output of result** is shown in called function.*	***Output of result** is shown in called function.*	***Output of result** is shown in calling function.*
***Processing** is done in called function.*	***Processing** is done in called function.*	***Processing** is done in called function.*	***Processing** is done in called function.*

Differentiate actual and formal parameter.

***actual parameter/argument :** actual parameter is used inside function call statement. no. of actual parameters passed must match as specified in function declaration and function definition's header*

line. value of actual argument can not be changed by called function unless passed by reference/address/pointer.
formal/dummy parameter/argument: *formal parameter is that which appears in function declaration or in function definition's function header line. in function declaration name of parameter is optional whereas in function definition's headerline name of variable is compulsory.*

Differentiate function declaration and function definition.

Function declaration	***Function definition***
Function declaration specifies no. of arguments and data types of parameters.	*Function definition specifies no. of arguments and data types of argument. function definition must match with function declaration in terms of no. of arguments and data type.*
Function declaration can be omitted if function definition appears before function use.	*Function definition can not be omitted.*
Function declaration is similar to function definition's function header line but is terminated by semicolon.	*Function definition's function header line never terminates by semicolon.*
Function declaration does not contain function body.	*Function definition contains function body.*

What do you mean by recursion?
Expressing an entity in terms of itself is known as recursion. When function calls itself at least once the function is called recursive function. Recursive function contains a function call statement to call itself normally kept inside if-else construct. Putting function call inside if-else construct allows termination of recursive function call after some time.
While defining a recursive function we must take care of these two points
** Each time a function calls itself it must be closer, in some sense to a solution.*
** There must be a decision criterion for stopping recursive function call otherwise it will execute till the stack overflows.*

Example calculating factorial using recursion:
```
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
long f;
long fact(int);
printf("\n enter a value to know factorial");
scanf("%d",&n);
f=fact(n);
printf("\n factorial=%ld",f);
getch();
}
long fact(int n)
{
if (n<=1)
   return n;
else
   return n* fact(n-1);
}
```

Example Generating fibonacci sequence.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int num,f1=0,f2=1;
 void fib(int,int,int);
 printf("\n how many terms to generate:");
 scanf("%d",&num);
 fib(f1,f2,num);
 getch();
}
void fib(int f1,int f2,int num)
{
 if (num!=0)
    {
       printf("\t %d",f1);
       fib(f2,f1+f2,num-1);
    }
}
```

Example to evaluate x – x to the power 3/ factorial 3+x to the power 5/factorial 5….

```
#include<conio.h>
#include <stdio.h>
int main(void)
{
  float x,s;
  int n;
  float sin(float,float,int,int );
  printf("\n enter value of x and n");
  scanf("%f %d",&x,&n);
  s=sin(x,x,1,n);
  printf("\nvalue=%f,%f",s,g);
  getch();
}


float sin(float term,float x,int i,int n)
{
float p;
if (i<=n)
 {
 printf("\n %f",term);
 p= term+sin(-term*x*x/((2*i)*(2*i+1)),x,i+1,n);
 }
return p;
}
```

Compare iteration and recursion./Write short notes on recursion and Iteration.

Iteration	***Recursion***

Initialization: *the loop variable is initialized along*	***Initialization:*** *the variable in the form of*

with other predefined variables.	*arguments are passed on the function.*

Decision*: The loop variable is compared, and based on the outcome, the loop is executed if condition is true. The test performed in loop variable may not be the only condition; other relational expression may also participate in the decision.*	***Decision:*** *the argument values are used to determine whether further recursive calls are necessary.*

Computation: *the necessary computation is performed within the loop.*	***Computation:*** *the necessary computation is performed using the local variable and the parameters at the current depth.*

Update: *the decision parameter is updated and a transfer is made to the next iteration.*	***Update:*** *the update is done so that the variables can be used for further recursive calls.*

How can we use variable length argument list?

There are some functions which can accept variable length argument list for example printf, scanf etc. variable length argument list allow passing varying no. of variable while making function call.

Consider an example:

```
#include<stdio.h>
#include<conio.h>
#include<stdarg.h>   /* for variable no. of argument list */
void main()
{
void print(const char start[],…);
print("First call","Hello",1,"World",2,NULL);
print("Second call","Good",10,"Morning",20,"to all",30,NULL);
getch();
}
void print(const char start[],…)
{
va_list arg_list;
int i;
char *s;
printf("\n %s\n",start);
va_start(arg_list,start);
while(s)
{
 s=va_arg(arg_list,char *);
 i=va_arg(arg_list,int);
 if(s)
    printf("%s %d",s,i);
}
va_end(va_list);
}
```

explanation:

* va_list is defined in stdarg.h and is an array used in obtaining the arguments that come in place of the ellipsis(…).
* va_start initializes the va_arg of data type va_list.
* va_arg is the one which actually extracts the argument values.
* the first parameter must be present since its name is to passed to va_start.
* va_end releases memory allotted during call of va_start.

How do we access command line parameter and environment variables/explain argc and argv?

```
#include <stdio.h>
#include<conio.h>
void main(int argc,char * argv[])
{
int i;
printf("\n no. of command line arguments is %d",argc);
for(i=0;i<argc;i++)
     printf("\n command line parameters are %s",argv[i]);
getch();
}
```

save the file with name my.c ,compile the program then go to dos prompt and type
c:\tcc\my teena meena and press enter.

```
no. of command line arguments is 3
command line parameters are c:\tcc\my.exe
command line parameters are  teena
command line parameters are  meena
```

explation: arugment count variable contains 3 because name of command(my),teena,meena gives total argument count 3. argv[0] contains name of command, argv[1] contains teena,argv[2] contains meena.
Name of argument can be any in place of argc, or argv.

Using command line argument convert uppercase string to lowercase and lowercase string to uppercase

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(int argc,char *argv[])
{
if(strcmp(argv[1],strlwr(argv[1])==0)
 printf("\n lower case changed to upper case as %s",strupr(argv[1]));
else
printf("\n upper case changed to lower case as %s",strlwr(argv[1]));
getch();
}
```

Scope and Extent

Define extent or lifetime of a variable.
Every variable in a program has some memory associated with it. Memory for variables is allocated and released at different points in the program. The period of time during which memory is associated with a variable is called the extent or life time of a variable

Define scope or accessibility of a variable.
scope of a variable can be defined as the region over which the variable is visible(accessible) or valid.
Scope can be local or global.
***Local**:*
Let us see an example:

```
#include<conio.h>
#include<stdio.h>
void main()
{
int i=10;
void display();
printf("\n value of i in main=%d",i);
display();
printf("\n value of j in display=%d",j); /* will result in compilation error */
getch();
}
void display()
{
 int j=50;
 printf("\n value of  i in main %d",i); /* will result in compilation error */
 printf("\n value of j in main %d",j);
}
```

explanation: variable i is declared in main therefore it is accessible in main only, similary j is declared in display it is accessible in j only.
*Variables declared inside curly brace are **local** to the brace and are not accessible outside the brace.*
Global:
Let us see an example:

```
#include<conio.h>
#include<stdio.h>
int gb=10;
void main()
{
void display();
printf("\n value of global gb  in main=%d",gb);
gb=20;  /* alteration of global variable  by main */
display();
printf("\n value of global gb in main after alteration made  by display=%d",gb); /* will result in
compilation error */
getch();
}
void display()
{
 printf("\n value of  global variable before alteration made by display= %d",gb);
 gb=40;
}
```

Output:

```
value of global gb  in main=10
value of  global variable before alteration made by display= 20
value of global gb in main after alteration made  by display=40
```

explanation: variable gb is not declared inside brace of any function and is accessible by every function as long as they do not contain a local variable with name similar to global variable(in this case we have to use scope resolution operator :: to modify/access value of global variable).
*Variables declared not inside any curly brace are **global.***

Differentiate global and local variables/ What are different types of scope available?

Global	***Local***
Global variable contains default value(if not initialized, assigned or scanned) zero.	*Local variable contains default value(if not initialized, assigned or scanned) garbage for auto storage class, zero for static storage class, garbage for register storage class.*
Global variables are not declared inside curly brace.	*These are declared inside curly brace.*
Every function can access and alter value of global variable except in the case a local variable with same name as that of global does not exist otherwise we would be needed to use :: operator .	*We can access/alter value of local variable inside the brace where it is declared.*
Memory associated for global variable is up to termination of program.	*Memory is associated for local variable till the execution of statement inside the brace where the variable is declared.*
Global variables declared as static int a=10; not inside any brace can be accessed in single file known as static global variable having file scope whereas global variable declared as int a=10; known as global variable can be accessed inside any file if program consists of many files linked together.	*Local variables can be with block scope means local variables can be accessed across the brace where these are declared and inside the nested blocks if variables with same name do not exists in nested block.*

If a global variable with name ‘a’ exists and local variable with name ‘a’ exists how can we access value of global variable?/program to demonstrate local variable and block scope

We need to use :: operator known as scope resolution operator.

```
#include<stdio.h>
#include<conio.h>
int g=10;
void main()
{
 int g=15;
    {
       int g=20; /* nested block */
       printf( “\n %d”,g);
       printf(“\n %d”,::g);
    }
 printf(“\n %d”,g);
 printf(“\n %d”,::g);
 getch();
}
```

output:

```
20
10
15
10
```

explanation: ::g will always refer to global variable. If int g=20; were not declared inside the nested block the print would have been 15 10 15 10.

What do you mean by storage class?

Storage class of a variable defines extent/lifetime(period of time during which memory is associate with variable), default value, place of memory allocation and scope/accessibility.

They are following types:

1. *auto*
2. *static*
3. *register*
4. *extern*

Explain auto variable.
All variables declared within a function are auto by default. The extent of a variable is defined by its scope. Variables declared auto can only be accessed only with the function or the nested block within which they are declared. They are created when the block is entered into and destroyed when it exited.

Explain static variable.
C defines another class of variables called static variables. Static variable are of two types:
1. static : variables that are declared within a function (function static variables). These variables retain their values from the previous call, i.e., the values which they had before returning from the function.
2. file static variable : these variable are declared outside any function using the keyword static. File static variables are accessible only in the file in which they are declared. This case arises when multiple files are used to generate one executable code.
1. function with static variables:

```
void main()
{
void display();
display();
display();
display();
}
void display()
{
static int a=0;
printf("\n count =%d",a);
a++;
}
```

output will be 0 1 2
let us see one more example of printing of Fibonacci series.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,i;
void printfib();
printf("\n how many terms of Fibonacci series to print");
scanf("%d",&n);
for(i=1;i<=n;i++)
    printfib();
getch();
}

void printfib()
{
static int t1=0,t2=1,t;
printf("\n %d",t1);
t=t1+t2;
t1=t2;
t2=t;
}
```

explanation: first call causes initialization of variables t1 to 0, t2 to 1 and t3 to zero(due to static).after the first function call to printfib the value of the variables t1 and t2 are both changed to one and retained for use in the next call and so on. Static variables are initialized only once first time the function is called.

Write short notes on register variables.
Register variable occupy memory allocation in cpu's register which are faster than RAM thus resulting faster program execution. Therefore best suited for loop counter. Default value is garbage and scope is limited to block scope. No. of cpu's register is limited and if busy register variable will change to auto variable without any message. Moreover cpu's register is built on some fixed no. of bits if register is built on 16 bits than we can declare variables of int, char data type as register.
e.g.

```
#include<stdio.h>
#include<conio.h>
void main()
{
register  int i;
for(i=1;i<=1000;i++)
    printf("\n %d",i);
getch();
}
```

Write short notes on extern variables/extern storage class.
Global variable are not declared inside any brace. Every function can access its value and can alter it. If local variable with same name as that of global exists we have to use scope resolution (::) operator to access global variable.
When global variable is declared later but used earlier then we have to use extern keyword.
e.g.

```
#include<stdio.h>
#include<conio.h>
void main()
{
extern int i;      /* indicates a variable i declared global appears later */
printf("\n %d",i);
getch();
}
int i=10;
```

Pointers

What is pointer?
Pointer is a variable that stores address of a variable.

How a pointer can be initialised?
e.g. int a, *b=&a;
char *b="ramesh";

What do you mean by addressing?
Addressing is a technique used to access the contents of one or more memory locations.
There are two types of addressing technique:

1. relative: it consist of base(segment) and offset address and is useful to access memory outside 64kb of memory.

2. absolute: all relative address is first translated to absolute address prior to physically accessing the memory and is useful to access memory within 64 kb limit.

What are the uses of pointers?

Usages are as follows:

1. *accessing array of elements.*
2. *passing arguments to functions by reference(i.e arguments can be modified).*
3. *passing arrays and strings to functions.*
4. *creating data structures such as linked list, trees, graphs and so on.*
5. *obtaining memory from the system dynamically to create dynamic array.*
6. *program using pointer runs faster due to direct dealing with memory.*
   thus pointer provide flexibility in programming.

How to use address operator?

/* program to demonstrate address of operator, dereferencing operator and pointer declaration */

Address operator ‘&’ gives memory location of variable.
Dereferencing operator ‘*’ gives value of variable whose address is stored in pointer.

Let us see by an example.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=10;
int *b;
b=&a;
/*  or   int *b=&a;     */
/*  or    int a=10, *b=&a;    */
printf("\n value of a=%d",a);
printf("\n value of a=%d",*b);

printf("\n address of a=%u",&a);
printf("\n address of a=%u",b);

a=50;
printf("\n value of a=%d",a);
printf("\n value of a=%d",*b);

printf("\n address of a=%u",&a);
printf("\n address of a=%u",b);

*b=200;
printf("\n value of a=%d",a);
printf("\n value of a=%d",*b);

printf("\n address of a=%u",&a);
printf("\n address of a=%u",b);

getch();
}
```

explanation: if address of a is 100 (suppose) then b=&a; will store this memory location 100 as value in variable b. if it is required to access value of variable a we can use *b, here * is known as dereferencing operator. All variables can be accessed directly by variable name are through pointer.

An assignment statement a=50; will change value of variable a to 50 directly and assignment *b=200 will change value of variable a to 100 indirectly (through pointer).
Address of variable ‘a’ will remain same.

What do you mean by call by reference(demonstration of pointer and function)?
When address of actual argument is passed to a function, the function can change value of actual argument know as call by reference or call by address.
*Let us see an example (**example is of pointer and function**)*

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=3,b=4;
void swap(int *,int *);
printf("\n before call of function swap value of a=%d,b=%d",a,b);
swap(&a,&b);
printf("\n after call of function swap value of a=%d,b=%d",a,b);
getch()
}
void swap(int *x,int *y)
{
 int t;
  t=*x;
 *x=*y;
 *y=t;
}
```

output:

```
before call of function swap value of a=3,b=4
after call of function swap value of a=4,b=3
```

clearly called function is able to modify the value of actual argument.

What do you mean by void pointer?
Normally a pointer to integer is able to store address of integer variable. A pointer to float is able to store address of float variable and so on. But what if, we want a generic pointer which can store address of integer, address of float and so on? The solution lies in use of void pointer.
void pointer is special type of pointer which can store address of variable of any data type. When it is required to access value of variable(address of which stored in void pointer) we need type casting. Let us understand:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=10;
float b=20;
char c='a';
void *p;
p=&a; /* pointer to void stored address of integer variable */
printf("\n value of int a=%d and address=%u",(int *)p,p);

p=&b; /* pointer to void stored address of integer variable */
printf("\n value of float b=%f and address=%u",(flaot *)p,p);

p=&c; /* pointer to void stored address of integer variable */
printf("\n value of char c=%c and address=%u",(char *)p,p);
```

getch();
}
clearly before accessing value of variable whose address void pointer contents type casting is necessary.

What is the valid Pointer arithmetic operation?
Followings are valid pointer arithmetic operation:-

```
int a,b,*p,*q;
p=-q;    /* illegal use of pointer */
p<<=1;  /* means dividing a pointer by 2 (a constant) is illegal*/
p=p-b;  /* valid pointer can be subtracted by a constant */
p=p+b;  /* valid pointer can be added by a constant */
p=(int *)p-q;  /* pointers can be subtracted */
p=p-q-a;  /* non portable pointer conversion */
p=(int *)p-q –a /* valid */
p=p+q; /*invalid , two pointers can not be added */
p=p+q+a; /* invalid */
p=p/q;  /* invalid */
p=p*a;    /* invalid */
p=p/q;   /* invalid */
p=p/b   /* invalid */
p=a/p;  /* invalid */
p++;   /* valid */
p-- ;  /* valid */
```

What do you mean by pointer in mathematical expression.
It means that pointer can be used in mathematical expression to refer the value of variable indirectly.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=10,b=20,c;
int *p,*q;
p=&a;
q=&b;
c=*p**q+10/2;
printf("\n value of c=%d",c);
getch();
}
```

output of above program will be 205 due to c=10*20+10/2;

What do you mean by pointer to pointer.
Pointer to pointer is a variable which can store address of another pointer.
Usages:
1. when it is required to pass 2d array to function.
2. when it is required to change value of a pointer by called function we need to pass the address of pointer and the formal parameter must be pointer to pointer.

Let us see an example:

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

```
void main()
{
int a=10;
int * b;
int **c;
b=&a;
c=&b;
printf("\n value of a=%d address=%u",a,&a);
printf("\n value of a=%d address=%u",*b,b);
printf("\n value of a=%d address=%u",*c,**c);

printf("\n value of b=%u address of b=%u",b,&b);
printf("\n value of b=%u address=%u",*c,c);

printf("\n value of c=%u address=%u",c,&c);
getch();
}
```

How can we use pointer and array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a[5]={3,1,5,2,4};
 int i;
 /* routine to print array using pointer notation */
 for(i=0;i <5;i++)
     printf("%d ",*(a+i));
 getch();
}
```

explanation: a[i] is equivalent to *(a+i), &a[i] is equivalent to (a+i);

How can we pass array to function or describe accessing array inside function?

```
#include<stdio.h>
#include<conio.h>
void main()
{
void sort(int *a,int n); /*  pointer a to hold the array and n to process the no. of elements */
int a[100],n,i;
printf("\n how many elements <max 100>");
scanf("%d",&n);
printf("\n enter %d values",n);
for (i=0;i<n;i++)
   scanf("%d",(a+i));  /*   &a[i] is same as  (a+i)   */
sort(a,n);
for(i=0;i<n;i++)
     printf("\n %d",*(a+i));
getch();
}
void sort(int *a,int n)
{
int i,j,temp;
 for(i=0;i<n-1;i++)
    for(j=0;j<n-1-i;j++)
```

```
        {
          if( *(a+j) > *(a+j+1) )
            {
              temp=*(a+j);
              *(a+j)=*(a+j+1);
              *(a+j+1)=temp;
            }
         }
}
```

explanation: in the above example main() declares an array of capacity 100. user is given a change to enter no. of elements he wants to use out of 100. Whatever user gives gets stored in n. array is passed to called function sort with no. of elements n, to process. Inside sort function bubble sort technique of sorting is used to sort the element of array. Sorting of array results in sorting of array inside main because array is always passed by reference. When we passed the array, base address of array is stored in pointer a in formal parameter of function main.

How can we access 2d array using poiter notation(pointer and two dimensional array)?
An element a[i][j] of two dimensional array is equivalent to *(*(a+i)+j). &a[i][j] is equivalent to (*(a+i)+j). let us understand with an example.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[2][3]={1,2,3,4,5,6};
for(i=0;i<2;i++)
  {
   for(j=0;j<3;j++)
        printf("\t%d",*(*(a+i)+j));
   printf("\n");
  }
getch();
}
```

What do you mean by array of pointers?
Just as we can have an array of integer, array of floats so we can have array of pointers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *a[3];  /* a is an array of pointer  */
int b=10;c=20,d=30;
a[0]=&b;
a[1]=&c;
a[2]=&d;
printf("\n value of b=%d, value of c=%d, value of d=%d",*a[0],*a[1],*a[2]);
getch();
}
```

Write program to sort strings using pointer exchange.
Or
How to pass array of pointers to functions/program to sort list of names using pointers/program to demonstrate array of pointers/How array of pointers to character can represent collection of strings?

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

```
#include<stdlib.h> /* for malloc library function */
void main()
{
char *name[5];
void sortbyptr(char **,int); /* function prototype */
int i;
printf("\n enter 5 names");
for(i=0;i<5;i++)
 {
 name[i]=(char *)malloc(20);
scanf("%s",name[i]);
}
sortbyptr(name,5);
printf("\n names in sorted order ");
for(i=0;i<5;i++)
 {
 printf("%s",name[i]);
}
}
void sortbyptr(char **a,int n)
{
char *t;
int i,j;
for(i=0;i<n-1;i++)
 for(j=0;j<n-1-i;j++)
 {
 if (strcmp(a[j],a[j+1])>0)
  {
  t=a[j];  a[j]=a[j+1]; a[j+1]=t;
}
}
}
```

explanation: before reading string in array of pointer to character we need to allocate memory of each pointer element, after memory allocation we can read string using scanf.

What is the difference between pointer to character and character array?

Character array	***Pointer to character***

Name of array denotes address of first element (a[0]'s address) this address is known as base address and is a pointer constant or constant pointer. If array is declared as char a[20]="raman"; then we can not write any statement like a++, a-- ,a=a+2 etc. which will modify 'a'.	*If a pointer to character 'b' holds a string "raman" we can increment/decrement pointer to character b therefore b++,b=b+2 etc. are valid statements.*

if declared char a[20]="raman"; it will take 20 bytes which is wasting of memory because it requires just 6 elements to hold string "raman" in which null character is also counted. So better technique is char a[]="raman" which will take just 6 elements.	*char *b="raman" will take just 6 bytes*

To assign new string to array 'a' declared as char	*To assign new string to pointer to*

[20]=”raman” we need strcpy function as follows strcpy(a,”teena”); we must not assign string having no. of characters more than 19 counting for null. character. We can not write statement a=”teena”;	character b we need b=”rameshwari”; we need not to worry about length of string being assigned.

To input string using scanf we need not to allocate memory for array ‘a’ declared as char a[20] but care must be taken no. of characters given for input must not exceed 19.	To input string using scanf we need to allocate memory for pointer to character b declared as char *b.

Let us see an example.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[20]=”raman”; /* a is pointer constant or constant pointer */
char *b=”raman”;
printf(“\n printing string character by character by incrementing pointer to character”);
for(i=0;*b!=’\0’;b++)
      printf(“%c”,*b);

/*
(i) following is invalid
printf(“\n printing string character by character by incrementing character array”);
for(;*a!=’\0’;a++)
      printf(“%c”,*a);
this is invalid because ‘a’ is pointer constant or constant pointer.
(ii)following is valid
printf(“\n printing string character by character by incrementing i”);
for(i=0;*(a+i)!=’\0’;i++)
      printf(“%c”,*(a+i));
because ‘a’ is not being modified in this case.
(iii)following is valid
q=a; where q has been declared char *q;
then
for(;*q!=’\0’;q++)
      printf(“%c”,*q);
because a is not being modified in this case.

(iv)following is valid
printf(“\n printing string character by character by incrementing i”);
for(i=0;*(b+i)!=’\0’;i++)
      printf(“%c”,*(b+i));
*/
getch();
}
```

Can we store a string in a pointer to character(pointer and string)?

A pointer to character can store a string constant.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char *s=”raman”;
/*  or  char *s={‘r’,’a’,’m’,’n’,’\0’};   */
int i;
printf(“\n the string is as follows\n”);
for (i=0;*s!=’\0’;s++)
        printf(“%c”,*s);
/* we can also use printf(“%s”,s);  */
getch();
}
```

Write a program in c using pointer and function to receive a string and a character as argument and return the no. of occurrences of this character in the string.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int count(char *,char);
char a[80],t;
int n;
printf(“\n enter string “);
gets(a);
printf(“\n enter character to search”);
scanf(“%c”,&t);
n=count(a,t);
printf(“\n no. of occurrences=%d”,n);
getch();
}
int count(char *a,char t)
{
int m,s=0;
for(m=0;a[m]!=’\0’;m++)
  if(t==a[m])
    s++;
return s;
}
```

What is the difference between array of pointer to character and double dimension character array?

Array of pointer	***Double dimension character array***

if we declare array of pointers to character as follows: *char *a[3]={“ram”,”shyam”,”ajay”};* *it consume memory 21 bytes (4 bytes for string ram+ 6 bytes for string shyam+ 5 bytes for string ajay+ 6 bytes*	*If we declare as below then it will take more computer memory.* *char a[3][20]={“ram”,”shyam”,”ajay”};* *it will consume memory 60 bytes.*

for pointer to characters a[0],a[1],a[2]). ***Thus it is memory saving if used initialization.***	***Thus it is more memory consuming if used initialization.***

if we declare array of pointers and want to read string during run time of program, we must allocate memory before reading input.	*We don't need allocation of memory before reading strings in double dimensional character array.*

Differentiate between pointer to constant and constant pointer(pointer constant)?

pointer to constant	***constant pointer or pointer constant***

If we declare pointer *const int *a; or int const *a; then a is pointer to constant.* *Suppose:* *int p=20;* *a=&p;* *then we* ***can not*** *use statement *a=50; to alter value of p.* *but we* ***can use*** *statement a++, a=a+2 etc.*	*(i)When we declare an array as* *int a [5]={1,2,3,4,5};* *then 'a' denotes base address which is address of first element of array. Base address of array is constant pointer(pointer constant) because we* ***can not*** *write any statement like a++ , a-- , a=a+2 etc. to modify 'a'.* *(ii) when we declare a pointer as* *int p=20;* *int * const a=&p; now a is pointer constant or constant pointer, since const appears before 'a'.* *if we write statement* **a=50; will change value of p to 50.* *Again we can not write any statement like a++ , a-- , a=a+2 etc. to modify 'a'.*

const is used before or after data type.	*const is used between * and variable name*

Pointer to constant is useful when actual parameter must not be modified by called function when passing by address.	

What does int const *const p=&i ; means?

It means that pointer to constant and constant pointer. This declaration neither allows p++ nor allows *p=20;

Explain the following notations/ *differentiate int *m[10] and int (*m)[10].*

(i)int (*p)[10];

p is pointer to an array having 10 elements.

(ii)int *p[10];

p is an array of pointers having 10 elements

(iii)int **p;

p is pointer to pointer

(iv) float * p();

p is a function returning pointer to float

(v) float * (*p)();

p is a pointer to function which returns pointer to float

(vi) int (*p)(int,int);

p is a pointer to function which returns integer and takes to integer arguments.

(vii) int (*p[3])(int,int);

p is a an array of pointer to function having 3 elements which returns integer and takes to integer arguments.

Structure and Union:

What is structure?./ write notes on tag and template.

Structure is a user defined data type, similar to record in a database management system. Defining of structure works as **template** (prototype, model or rubber stamp) from which we can create structure variable. It groups variables into a single entity.

e.g.

```
struct emp
{
char name[20];
int age;
};
```

this is known as structure declaration. After structure declaration we can create a structure variable as follows:

struct emp e;

now e will be composed of name and age. Name and age are known as data members of structure.

In above example emp is known as structure **tag**. Structure tag is useful to identify one structure declaration from other when program consists of many structure declaration.

How to declare structure and structure variable/ How can we input/output a structure variable?

```
#include <stdio.h>
#include <conio.h>
void main()
{
struct emp
{
  char name[20];
  int age;
  float salary;
};
struct emp e;
printf("\n enter name, age salary");
scanf("%s %d %f",e.name,&e.age,&e.salary);
printf("\n name=%s,age=%d,salary=%f",e.name,e.age,e.salary);
getch();
}
```

explanation: struct emp e is known as structure variable declaration. Structure variable e is composed of name, age and salary. While using scanf we did not used & in front of e.name because name is character array.

How to declare structure and initialize structure variable?

```
#include <stdio.h>
#include <conio.h>
void main()
{
struct emp
{
  char name[20];
  int age;
  float salary;
};
struct emp e={"ramesh",22,5500};
printf("\n name=%s,age=%d,salary=%f",e.name,e.age,e.salary);
getch();
}
```

explanation: struct emp e is known as structure variable declaration. Structure variable e is composed of name, age and salary.

What are the local and global structure declaration?

When we write structure declaration inside any function it is known as local structure declaration in this case only the function where structure is declared, we can declare structure variable. When we write structure declaration not inside curly brace it is known as global structure declaration in this case we can declare structure variable inside any function.

What are different forms/variation in structure variable declaration?

They are as follows:

```
(a)struct emp
    {
      char name[20];
      int age;
    };
    struct emp e;
(b) struct emp
    {
      char name[20];
      int age;
    }e;

(c)struct
      {
      char name[20];
      int age;
    }e;
```

this form of structure declaration did not used structure tag therefore we can not declare any structure variable from this point onward.

What care we need to take when initializing structure variable?

The values we give for structure variable initialization inside curly brace must match with the order in which data members appears inside structure declaration.
e.g.

```
struct emp
      {
      char name[20];
      int age;
      float salary;
    };
   struct emp e={“rama”,22}; /* from missing data member if numeric, will be zero , for missing data
member
if character array it will be “”  */
```

if structure declaration is as follows :

```
struct emp
      {
      int age;
      float salary;
      char name[20];
    };
```

then initialization will be
struct emp e={22,4200,”rama”};

What do you mean by structure within structure(nested structure)?
When a structure variable is used as data member in another structure declaration it is known as structure within structure or nested structure.
Let us see one example.

```
#include<stdio.h>
#include<conio.h>
struct date
{
int mm,dd,yy;
};
struct emp
{
char name[20];
struct date dob;
int age;
};
void main()
{
 struct emp e;
 printf(“\n enter name ,date of birth (mm dd yy) and age”);
 scanf(“%s %d %d %d %d”,e.name,&e.dob.mm,&e.dob.dd,&e.dob.yy,&e.age);
 printf(“\n name=%s,date of birth (mm dd yy)=%d %d %d,age=%d” ,e.name, e.dob.mm, e.dob.dd,
e.dob.yy, e.age);
getch();
}
```

explanation: e is composed of name, dob and age, and dob is itself composed of mm, dd, yy. When inputting(reading/scanning) we have to use e.name, &e.dob.mm and so on.
Initialization of nested struct will be
struct emp e={“rama”,5,7,1980,23};

What are the different forms of declaration of nested structure?
Forms are:

(a)struct emp	(a) struct date	
{	{	
char name[20];	int mm,dd,yy;	
struct date	};	
{	struct emp	
int mm,dd,yy;	{	
}dob;	char name[20];	
int age;	struct date dob;	
};	int age;	
	};	

What are the different operation which can be performed on structure variable or what are the operations can be performed on union?

On Structure/Union variable we can perform following operation:these concept of operations are useful while learning c++ in next sem/year therefore should be learn carefully.

1. *we can assign one structure/union variable to other structure/union variable when both belong to same structure/union.*
2. *we can initialize the structure/union variable.*
3. *we can access value of data member of structure/union using . (dot) known as member access operator.*
4. *we can assign value to data member of structure/union.*
5. *we can declare array of structure/union*
6. *we can pass structure/union variable to function by value.*
7. *we can pass array of structure/union to function(passed as reference).*
8. *we can pass a structure/union variable to function by reference.*
9. *function can return structure/union variable and can take structure as argument.*
   We can not use existing operator +,-,*,/ on two structure/union variable.
   e.g. c3=c1+c2; are invalid.

How can we assign a structure variable to another structure variable belonging to same structure declaration?

Let us see with one example.

```
#include <stdio.h>
#include <conio.h>
void main()
{
struct emp
{
  char name[20];
  int age;
  float salary;
};
struct emp e={“ramesh”,22,5500},f;
f=e;
printf(“\n name=%s,age=%d,salary=%f”,e.name,e.age,e.salary);

printf(“\n name=%s,age=%d,salary=%f”,f.name,f.age,f.salary);
getch();
}
```
output will be name=ramesh,age=22,salary=5500.000000
output will be name=ramesh,age=22,salary=5500.000000

How can we assign values to data members of structure variable?
Let us see one example
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
struct emp
{
  char name[20];
  int age;
  float salary;
};
struct emp e;
strcpy(e.name,”ramesh”);
e.age=22;
e.salary=5500;
printf(“\n name=%s,age=%d,salary=%f”,e.name,e.age,e.salary);

getch();
}
```
output will be name=ramesh,age=22,salary=5500.000000

How to declare and input/output array of structure?
Let us see an example
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct emp
{
  char name[20];
  int age;
  float salary;
};
```

```
void main()
{
struct emp e[3];
int i;
float temp;
for(i=0;i<3;i++)
 {
    printf("\n enter name,age and salary");
    scanf("%s %d %f",e[i].name,&e[i].age,&temp);
    e[i].salary=temp;
 }
printf("\n details of 3 employees");
for(i=0;i<3;i++)
 {
    printf("\n name=%s,age=%d and salary=%f",e[i].name,e[i].age,e[i].salary);
 }

getch();
}
```

for given input:

```
ram 22 2200
shyam 23 2300
ajay 24 2400
```

output will be

```
name=ram,age=22,salary=2200.000000
name=shyam,age=23,salary=2300.000000
name=ajay,age=24,salary=2400.000000
```

How to declare and initialize and output array of structure?

let us see an example

```
#include <stdio.h>
#include <conio.h>
struct emp
{
  char name[20];
  int age;
  float salary;
};

void main()
{
struct emp e[3]={"ram",22,2200,"shyam",23,2300,"ajay",24,2400};
int i;
printf("\n details of 3 employees");
for(i=0;i<3;i++)
 {
    printf("\n name=%s,age=%d and salary=%f",e[i].name,e[i].age,e[i].salary);
 }
getch();
}
```

output will be

```
name=ram,age=22,salary=2200.000000
name=shyam,age=23,salary=2300.000000
```

```
name=ajay,age=24,salary=2400.000000
```

How to pass a structure variable to function?

let us see an example

```
#include <stdio.h>
#include <conio.h>
struct emp
{
  char name[20];
  int age;
  float salary;
};

void main()
{
struct emp e={"ram",22,2200};
void display(struct emp);
display(e)
getch();
}
void display(struct emp e)
{
printf("\n name=%s,age=%d,salary=%f",e.name,e.age,e.salary);
}
```

output will be

```
name=ram,age=22,salary=2200.000000
```

Demonstrate call by value and structure variable passed to function.

```
#include <stdio.h>
#include <conio.h>
struct emp
{
  char name[20];
  int age;
  float salary;
};

void main()
{
struct emp e={"ram",22,2200};
void alter(struct emp);
printf("\nbefore call of function inside main name=%s,age=%d,salary=%f",e.name,e.age,e.salary);

alter(e);

printf("\n after call of function inside main name=%s,age=%d,salary=%f",e.name,e.age,e.salary);

getch();
}
void  alter(struct emp e)
{
printf("\n inside called function before change name=%s,age=%d,salary=%f",e.name,e.age,e.salary);
strcpy(e.name,"raman");
e.age=25;
e.salary=2500;
printf("\n inside called function after change  name=%s,age=%d,salary=%f",e.name,e.age,e.salary);
}
```

output will be

```
before call of function inside main name=ram,age=22,salary=2200.000000
inside called function before change name=ram,age=22,salary=2200.000000
inside called function after change name=raman,age=25,salary=2500.000000
after call of function inside main name=ram,age=22,salary=2200.000000
```

How to pass array of structure to function?

let us see an example remembering array is always passed by reference.
We can note that passing to single dimension numeric array to function and passing single dimension array of structure are analogous.

```
#include <stdio.h>
#include <conio.h>
#define size 3
struct emp
{
  char name[20];
  int age;
  float salary;
};

void main()
{
struct emp e[size];
void input(struct emp []);
void output(struct emp []);
input(e);
output(e);
getch();
}
void input(struct emp e[])
{
int i;
float temp;
for(i=0;i<size;i++)
{
  scanf("%s %d %f",e[i].name,&e[i].age,&temp);
  e[i].salary=temp;
}
}
void output(struct emp e[])
{
int i;
for(i=0;i<size;i++)
 {
    printf("\n name=%s,age=%d and salary=%f",e[i].name,e[i].age,e[i].salary);
 }
}
```

for given input:

```
ram 22 2200
shyam 23 2300
ajay 24 2400
```

output will be:

```
name=ram,age=22,salary=2200.000000
name=shyam,age=23,salary=2300.000000
name=ajay,age=24,salary=2400.000000
```

What do you mean by pointer to structure?
We know about pointer to integer,pointer to float and so on. Similary we can have pointer to structure. Let us see an example:

```
#include <stdio.h>
#include <conio.h>
struct emp
{
  char name[20];
  int age;
  float salary;
};

void main()
{
struct emp e={“ram”,22,2200};
struct emp *f;
f=&e;
printf(“\n name=%s,age=%d,salary=%f”,f->name,f->age,f->salary);
getch();
}
```

explanation:we can access data members of structure variable e in following ways:
during ouput

```
e.name, e.age, e.salary
f->name,f->age,f->salary;
(*f).name,(*f).age,(*f).salary;  /*    *f.name is invalid  */
```

during input

```
e.name, &e.age, &e.salary
f->name,&f->age,&f->salary;
(*f).name,&(*f).age,&(*f).salary;
```

How can we pass structure variable as reference.

```
#include <stdio.h>
#include <conio.h>
#include<string.h>
struct emp
{
  char name[20];
  int age;
  float salary;
};

void main()
{
struct emp e={“ram”,22,2200};
void alter(struct emp*);
printf(“\nbefore call of function inside main name=%s,age=%d,salary=%f”,e.name,e.age,e.salary);

alter(&e);
printf("\n after call of function inside main name=%s,age=%d,salary=%f",e.name,e.age,e.salary);

getch();
}
void  alter(struct emp *f)
{
printf("\n inside called function before change name=%s,age=%d,salary=%f",f->name,f->age,f-
>salary);
strcpy(f->name,"raman");
f->age=25;
f->salary=2500;
printf("\n inside called function after change  name=%s,age=%d,salary=%f",f->name,f->age,f->salary);
}
```

output will be

```
before call of function inside main name=ram,age=22,salary=2200.000000
inside called function before change name=ram,age=22,salary=2200.000000
inside called function after change name=raman,age=25,salary=2500.000000
after call of function inside main name=raman,age=25,salary=2500.000000
```

Write a program to demonstrate function taking structure as argument and returning structure.
let us see an example

```
#include<stdio.h>
#include<conio.h>
struct complex
{
float real,imag;
};
void main()
{
struct complex c1={1,1},c2={2,2},c3;
struct complex add(struct complex,struct complex);
c3=add(c1,c2);
printf("\n result of addition: real=%f,imaginary=%f",c3.real,c3.imag);
getch();
}
struct complex add(struct complex c1,struct complex c2)
{
struct complex temp;
temp.real=c1.real+c2.real;
temp.imag=c1.imag+c2.imag;
return temp;
}
```

What do you mean by union?
A union is a user defined data type in c which allows the overlay of more than one variable in the same memory location.

Why we need union?
Normally every variable is stored in a separate location and as a result, each of these variables have their own addresses, often, if it is found that some variables used in the program are used only in a small portion of the source code. For example, if a string of 200 bytes called filename is required by the first 500 lines of code oly, and another string called output of 400 bytes is required by the rest of the code(i.e. both strings are not needed simultaneously), it would be a waste of memory to declare

separate arrays of 200 and 400 bytes. The union construct provides a means by which the memory space can be shared and only 400 bytes of memory are used.

How to declare union?
Declaration of union is similar to structure but uses union keyword instead of struct keyword. E.g.

```
union emp
{
char name[20];
int age;
};
```

declaration of union variable.

```
union emp e;
```

or

```
union emp
{
char name[20];
int age;
}e;
```

now e has two data members name and age. Variable e will take no. of bytes required by largest data member which is 20 bytes.

Explain scope of a union/struct variable.
Scope of union is similar to scope of struct. If declaration of struct/union is not inside any curly brace then any function can declare and use struct/union variable known as global declaration. If declaration of struct/union is inside curly brace of function definition then the function which contains structure declaration can declare and use struct/union variable and declaration of structure is known as local scope.

```
#include<stdio.h>
#include<conio.h>
void main()
{
union emp
{
  char name[20];
  int age;
}e;
strcpy(e.name,”ram”);
printf(“\nname= %s”,e.name);
printf(“\nage= %d”,e.age); /* age has meaningless value */

e.age=20;
printf(“\nname= %s”,e.name); /* now name has meaningless value */
printf(“\nage= %d”,e.age);

getch();
}
```

explanation: in above program only main function can declare and use union variable.

Differentiate structure and union.

Structure	***Union***

Structure uses struct keyword.	*Union uses union keyword.*

The amount of memory taken by a structure variable is sum of sizes of all the data members.	*The amount of memory taken by a structure variable is size of largest data member in terms of no. of bytes.*

We can store values in all data members without any loss of value.	*Only one union data member can be assigned value which we can access. The data member which stores value most recently has valid value and others have invalid values.*

We don't need additional variable to identify active(most recently used) member.	*We need additional variable to identify active(most recently used) member.*

Structure consumes more memory when compared to union when not all the members are required to access simultaneously.	*union consumes less memory when compared to structure when not all the members are required to access simultaneously.*	

Write a program to demonstrate memory difference between structure and union

```
#include<stdio.h>
#include<conio.h>
void main()
{
union emp1
{
  char name[20];
  int age;
}e1;
union emp2
{
  char name[20];
  int age;
}e2;
printf("\n no. of bytes taken by structure %d",sizeof(e1));
printf("\n no. of bytes taken by union %d",sizeof(e2));
getch();
}
```

Write short notes on enumerated data type.

enum is a keyword which can be used to create user defined data type. using enum we can assign symbolic name to small list of integers.

```
#include<stdio.h>
#include<conio.h>
enum Boolean {false,true};
void main()
{
enum Boolean e; /* declaration of enum variable  which can take value either false or true */
e=true;
```

printf("\n value of true=%d",e);

getch();

}

explanation: since first item in list is false therefore it will take value 0, second item in list is true therefore it will take value 1 and so on if any more.

Dynamic memory allocation

What do you mean by dynamic memory location/write short notes on dynamic memory allocation/write short notes on malloc, calloc, realloc, free/ how dynamic memory allocation helps to solve complex problems?
dynamic memory allocation refers to the allocation of memory during progam run time for variables which are pointer. Dynamic memory is allotted from heap memory. Dynamic memory allocation helps to solve complex problems of data structure like linked list, stacks, queues where run time memory allocation is prime need.. Dynamic memory is also important when size of array is not known during program run time.

There are some function which are useful in dynamic memory allocation:-
*1.**malloc**-declaration - void * malloc(size_t size);*
The function malloc allocates a block of memory in bytes as specified by parameter size from the free data area (heap). it allows a program to allocate an exact amount of memory explicitly, as and when required. the parameter passed to malloc is of the type size_t. this type is declared in the header file stddef.h. 'size_t' is equivalent to unsigned int data type.
***return value of malloc:** malloc returns address of first byte of memory chunk allotted if memory allocation is successful otherwise returns NULL.*
*int *p;*
*p=(int *)malloc(sizeof(int)*1000);*
Above statement will allocate 2000 bytes of memory from heap.
If it is required to check whether memory allocation was successful or not we can use following statement:
if (p==NULL)
  printf("\n memory allocation failed");

*2.**free**-declaration- void free(void *block);*
free is a library function that dis-allocates memory block allotted to pointer by function calloc,malloc or realloc.
example
free((int)p);*

*3.**calloc**-declaration- void * calloc(size_t nitems, size_t size);*
calloc is a library function to allocate memory. if memory allocation is successful it return address of first byte of memory chunk allotted otherwise returns NULL.
e.g.
two allocate memory of 2000 bytes as in the previous example of malloc we can use
*int *p;*
*p=(int *)calloc(1000,sizeof(int));*

*4.**realloc**-declaration- void *realloc(void *block,size_t size)*
realloc adjusts the amount of memory allocated to the block to size, copying the contents to a new location if necessary.

```
int *p;
p=(int *)malloc(sizeof(int)*5); /* allocated 10 bytes */
```
suppose after some time we need 20 bytes and we do not want to destroy values stored in 10 bytes then we can use realloc as follows
```
p=(int *)realloc(p,sizeof(int)*10);
```

Differentiate malloc,calloc.

malloc	***calloc***
malloc takes one argument which is type unsigned int which stands for total no. of bytes to allocate	*calloc takes two arguments which are no. of items to allocate memory for, and size of each item in bytes*
The memory location allotted by malloc contains garbage value at memory location.	*The memory location allotted by calloc contains default value zero or null string(for pointer to character).*
*Syntax is: void * malloc(size_t size)*	*Syntax is :void * calloc(size_t nitems, size_t size)*

Write short notes on null pointer.
null pointer is special pointer value defined in stdlib.h,,malloc.h,stdio.h etc. when library function malloc,calloc, fopen fails operating successfully; they assign null pointer value to pointer. Sometimes usually in data stature we initialize pointer to value NULL.
using NULL:
```
(i) int *p=NULL; /* pointer initialized to NULL pointer value */
(ii)int *p;
p=(int *)malloc(2000);
if(p==NULL)
 {
 we can give statements to execute when memory allocation fails.
}
(iii)FILE *fp;
fp=fopen(“c:\\ram.txt”,”r”);
if(fp==NULL)
 {
 we can give statements to execute when file open operation fails.
}
```

How to avoid declaration of ordinary variable if using pointer?
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int *p;
p= (int *) malloc (sizeof(int)); /* allocate 2 bytes only */
*p=30;
printf(“\n value =%d”,*p);
getch();
}
```

How to create single dimension (numeric) dynamic array?
```
#include<stdio.h>
#include<conio.h>
```

```
#include<stdlib.h>
void main()
{
int *p;
int n,i;
printf("\n enter no. of elements");
scanf("%d",&n);
p= (int *) malloc (sizeof(int)*n); /* allocate 2*n bytes to generate an array p[n] */
printf("\n enter %d values",n);
for(i=0;i<n;i++)
  scanf("%d",&p[i]);   /*  or  (p+i)    */

printf("\n values are\n");

for(i=0;i<n;i++)
  printf("%d",p[i]);  /*   or   *(p+i)   */
getch();
}
```

How to create single dimension (character) dynamic array?

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
char *p;
int n,i;
printf("\n enter maximum no. of characters");
scanf("%d",&n);
p= (char *) malloc (sizeof(char)*(n+1)); /* allocate 1*(n+1) bytes to generate an array p[n+1] */
printf("\n enter string");
scanf("%s",p);
printf("\n the string is %s",p);
getch();
}
```

How to create double dimension (numeric) dynamic array?/using array of pointers?

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int *p[3]; /* pointers are p[0],p[1],p[2]  */
int n,i,j;
printf("\n enter no. of  columns");
scanf("%d",&n);

for(i=0;i<3;i++)
p[i]= (int *) malloc (sizeof(int)*n);
/* now p[3][n]; exists */

printf("\n enter %d values are\n",n*3);

for(i=0;i<3;i++)
```

```
 for(j=0;j<n;j++)
  scanf("%d",&p[i][j]);  /*   or   (*(p+i)+j)      */

printf("\n the array is \n");
for (i=0;i<n;i++)
{
  for(j=0;j<n;j++)
    printf("\t %d",p[i][j]);         /*    or   *(*(p+i)+j)    */
  printf("\n");
}
getch();
}
```

How to create double dimension (numeric) dynamic array/using pointer to array ?

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int (*p)[3];
int n,i,j;
printf("\n enter no. of  rows");
scanf("%d",&n);

p= (int *[]) malloc (sizeof(int)*n*3);
/* now p[n][3]; exists */
printf("\n enter %d values are\n",n*3);
for(i=0;i<n;i++)
 for(j=0;j<3;j++)
  scanf("%d",&p[i][j]);  /*   or   (*(p+i)+j)      */

printf("\n the array is \n");
for (i=0;i<n;i++)
{
  for(j=0;j<3;j++)
    printf("\t %d",p[i][j]);         /*   or   *(*(p+i)+j)    */
  printf("\n");
}
getch();
}
```

How to create double dimension (numeric) dynamic array/using pointer to pointer?

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int **p;
int m,n,i,j;
printf("\n enter no. of  rows and columns");
scanf("%d %d",&m,&n);

p= (int **) malloc (sizeof(int)*m);
for(i=0;i<m;i++)
 p[i]=(int *)malloc(sizeof(int)*n);

/* now p[m][n]; exists */
printf("\n enter %d values are\n",m*n);
for(i=0;i<m;i++)
 for(j=0;j<n;j++)
  scanf("%d",&p[i][j]);  /*  or   (*(p+i)+j)      */

printf("\n the array is \n");
for (i=0;i<m;i++)
{
  for(j=0;j<n;j++)
    printf("\t %d",p[i][j]);         /*   or  *(*(p+i)+j)    */
  printf("\n");
}
getch();
}
```

How to declare and use pointer to structure without using ordinary variable?

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct emp
{
char name[20];
int age;
};
void main()
{
struct emp *e;
e=(struct emp *)malloc(sizeof(struct emp));

printf("\n enter details of employee name and age:\n");
scanf("%s %d",e->name,&e->age);

printf("\ndetails of employee : name and age:\n");
printf("%s %d",e->name,e->age);
getch();
}
```

How to declare dynamic array of structure using pointer to structure?

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct emp
{
char name[20];
int age;
};
void main()
{
struct emp *e;
int n;
printf("\n enter no. of elements");
scanf("%d",&n);
e=(struct emp *)malloc(sizeof(struct emp)*n);
/* now we have  struct emp e[n]; */

printf("\n enter details of %d employees : name and age:\n",n);
for(i=0;i<n;i++)
 {
scanf("%s %d",e[i].name,&e[i].age);
}
printf("\ndetails of %d employees : name and age:\n",n);
for(i=0;i<n;i++)
 {
printf("%s %d",e[i].name,e[i].age);
}
getch();
}
```

File Handling

How pointer is useful in file handling?

file handling requires declaration of pointer to structure FILE. FILE is a predefined structure in stdio.h. when we open file for file handling purposes(reading, writing, modifying) using fopen library function file is opened in computer's memory location address of which is stored in pointer to structure FILE.

Write short notes on file handling in c/ write short notes on input-output operation on file.

File i/o operation is always done in a program in the following sequences

1.open the file
before performing i/o in any file, file must be opened.
fopen is the library function used to open the file.
declare the pointer to predefined structure FILE (all in uppercase)
*FILE *fp;*
open the file using fopen library function.
fp=fopen("filename","mode");
filename should be any valid dos filename , mode can be any one out of the modes r,w,a,r+,w+,a+. Be careful mode must be given in lowercase only. 'rt' indicates read-text mode 'rb' read-binary.

2. read or write to the file
read/ write data from/to the opened file.

3. close the file
during a write to a file, the data written is not put on the disk immediately but it is stored in the buffer. When the buffer is full, all its contents are actually written to the disk. The process of emptying of buffer by writing its contents to disk is called ***flushing the buffer****.*
closing the file flushes the buffer and releases the memory space taken by the pointer to structure.
closing of file is done using fclose library function
fclose(filepointer);

Write notes on different modes of opening file.

(i)r : stands for read only mode. file must be existing otherwise fopen function will return NULL.
(ii) w: stands for write mode. if file exists file is replaced by new contents. If file doesn't exist new file is created.

(iii) a: stands for append. if file exists new contents can be added at last. If file doesn't exist file is created.
(iv) r+: opens an existing file for reading, adding and modifying. If file doesn't exist fopen will give NULL.
(v)w+: opens a file for reading, adding and modifying. If file exists it will be overwritten.
(vi)a+: opens a file for reading and adding only. If file does not exist new file is created. we can modify existing contents.

What are the different ways of organizing files in c/ differentiate text and binary mode of file handling.

How does append mode differs from write mode?

Dos creates files in two different ways viz. as binary file or text file therefore there are two different ways of organizing the file.

Text mode	***Binary mode***
if the file is specified as the text type, the file i/o functions interpret the contents of the file when reading or writing the file.	*if a file is specified as the binary type, the file i/o functions do not interpret the contents of the file when reading or writing the file.*
When reading file in text mode, end of file character(^z with ascii value 26) must be there to tell that end of file has reached.	*No end of file character is required.*
A new line character is stored in file as carriage return and linefeed.	*A new line character is stored as linefeed character*
Additional disk memory is consumed to store same data when compared to binary mode(for each new line character it takes two bytes)	*Consumes no additional disk memory (for each new line it takes just one byte).*
When numbers oriented data need to save it is disk space consuming when compared to binary mode of file handling for example: *When storing value of integer , int a=1244; it will take 4 bytes in text mode.*	*When numbers oriented data need to save it is less disk space consuming when compared to text mode of file handling for example:* *When storing value of integer , int a=1244; it will take 2 bytes because 'a' is integer*
Text mode of file handling is slow because there is interpretation/conversion from text to binary and binary to text when reading and writing.	*It is fast because there is no interpretation/conversion from text to binary and binary to text.*

Write program to create sequential data file.
example of character by character reading and writing sequentially(unformatted).

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
void main()
{
FILE *fp;
fp=fopen("c:\\ram.txt","w");
if(fp==NULL)
{
printf("\n file open operation failed ");
getch();
exit (1);
}
printf("\n enter contents of file control+z to end the file\n");
while((ch=fgetc(stdin))!=EOF)
 fputc(ch,fp);

fclose(fp);

fp=fopen("c:\\ram.txt","r");

printf("\n contents of file is as follows\n");
while((ch=fgetc(fp))!=EOF)
 fputc(ch,stdout);
getch();
}
```

Write contents of one file to other file.

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
void main(int argc,char *argv[])
{
FILE *fp,*ft;
if (argc!=3)
{
printf("\n usage: programfilename  sorucefilename  targetfilename");
getch();
exit (1);
}
fp=fopen(argv[1],"r"); /* open source file in read mode */
ft=fopen(argv[2],"w");/* open target file in write mode */
/* let us check either of the file open operations failed, if failed then terminate program */
if (fp==NULL || ft == NULL)
{
printf("\n file open operation failed");
getch();
exit (1);
}
while((ch=fgetc(fp))!=EOF)
fputc(ch,ft);

fclose(fp);
fclose(ft);
getch();
}
```

save the file with name cp.c. compile the file (executable file will generate)
go to dos prompt and give command:

```
cp teena.text meena.txt press enter
```

Note:source file teena.txt must be existing and target file name must not be existing with read only attribute set or a directory with same name must not be in use.

Write short notes on error handling(during file i/o operation) in c.

There may be many sources of error when doing file input/output operation
(i)When opening the file for reading one can assume the file already exists but it is not existing.

(ii)When opening the file for reading file exists in bad sector where problem of reading is generating.
(iii)When opening the file for write mode, file already exists with read-only attribute set therefore file can not be replaced
(iv)When opening the file for write mode, disk is write protected or a directory with same name is already in use.
when error in opening file occurs fopen returns NULL therefore further operation on files must be done after checking value of file pointer is not NULL.
a global variable 'errno' is set to a specific value meant to designate the type of error, whenever an error occurs in a file i/o function. Value of 'errno' can be used to perform the required exception processing. The function 'perror' can be used to print the nature of the error. the function 'strerror' can be used to obtain a char pointer to the error string . the function 'clearer' can be used to reset the error indicator of a stream.

Write short notes on fprintf and ,fscanf.

(i)fscanf: fscanf is a library function which can be used to read data from stream(file).
syntax is fscanf(filepointer,"format string",&var1,&var2...);

(ii)fprintf:fprintf is a library function which can be used to write data to stream(sequence of bytes or file).
syntax is fprintf(filepointer,"format string",var1,var2...);
fscanf and fprintf is used for formatted data read/write to stream sequentially,

Write short notes on Random access file.
random access file allows to read/ write data from any location. random access file uses function fread,fwrite for the purpose of reading and writing. when reading/writing data in random access file length of record must be fixed. structure declaration and its variable declaration allows user to read and write data in terms of fixed no. of bytes. Noramlly random access file uses binary mode of file handling.

Write a program using fscanf, fprintf to read/write sequential formatted input/output file operation.
```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
void main()
{
FILE *fp;
fp=fopen("c:\\ram.txt","w");
if (fp==NULL)
{
printf("\n file open operation failed");
getch();
exit (1);
}
printf("\n enter name,age and salary control+z to end");

while(fscanf(stdin,"%s %d %f",name,&age,&salary)!=0)
 fprintf(fp,"%s %d %f",name,age,salary);

fclose(fp);

fp=fopen("c:\\ram.txt","r");
while(fscanf(fp,"%s %d %f",name,&age,&salary)!=0)
 fprintf(stdout,"%s %d %f",name,age,salary);
fclose(fp);
getch();
}
```

Write short notes on buffered and unbuffered files.
the character written to a stream are immediately output to the file or device in unbuffered file. the character written to a stream are accumulated and then written as a block to the file or device in buffered files. buffering speeds file input/output operation. stdin and stdout are unbuffered if they are not redirected otherwise they are fully buffered.

Write short notes on low-level file handling.
when the functions fread, fwrite, etc. , are used data transfer occurs first to the buffers and then to the disk. in low level file functions, the data is not buffered but directly written to the file. the low leve file function use file handles. open, read, write etc. are used as unbuffered file handling.

```
#include<stdio.h>
#include<io.h>
void main()
{
FILE *fp;
fp=fopen(“text”,”wt”);
fwrite(“1. this is fwrite\n”,1 strlen(“1. this is fwrite\n”),fp);
write(fileno(fp),”2. this is write \n”);
fclose(fp);
}
```

data of file “text” is as follows

```
2. this is write
1. this is fwrite
```

Write short notes on following functions:-
fseek, ferror, ftell, fflush, fread/fwrite, feof, rewind, perror, error/Write the format of fseek function

(i)fseek:
*int fseek(FILE *stream,long offset,int position-with-respect-to);*
the function fseek set the file pointer associated with a stream to a new position.
parameters are as follows:
stream:stream whose file pointer fseek will set.
offset:difference in bytes between position-with-respect-to and the new position for the stream.
position-with-respect-to can be any one out of three settings:
SEEK_SET (seek from beginning),SEEK_CUR (seek from current position), SEEK_END (seeks from end of file).
return value: returns value 0 on success.
on failure it returns a non zero value. fseek returns an error code only on an unopened file or when a non-existing device is accessed.

(ii)ferror: ferror is a macro that tests the given stream for a read or write error. If the stream error indicator has been set, it remains so until clearer or rewind is used or stream is closed.
*syntax: int ferror(FILE * stream)*
return value: returns a non-zero if an error is detected on the specified stream.

(iii)ftell: fetll returns the current file pointer position for the specified stream.
*syntax: long ftell(FILE *stream)*
return value: on success returns the current file pointer position.
on error returns -1 and sets errno to a positive value.

(iv)fflush: if the given steam has a buffered output then fflush writes the output of the stream to the associated stream.
*syntax: int fflush(FILE *stream)*
return value: on success it returns 0
on failure it returns EOF.

Appendix:

Write a c program that read the afternoon day temperature for each day of the month and then report the month average temperature as well as the hottest ad coolest day.
this program requires single dimension array to store day temperature of each day. it requires addition of elements of single dimension array finding average and coolest and hottest day.

```
#include<stdio.h>
#include<conio.h>
void main()
{
float a[31];
float sum=0,avg,hot,cool;
int n,i,hotday,coolday;
printf("\n enter no. of days in current month");
scanf("%d",&n);
printf("\n enter each day temperature");
for(i=0;i<n;i++)
   scanf("%f",&a[i]);

/* find sum now */
for(i=0;i<n;i++)
   sum=sum+a[i];

/* find average */
avg=sum/n;

hot=a[0];
cool=a[0];
hotday=1;
coolday=1;
for(i=1;i<n;i++)
  {
  if(hot<a[i])
    {
     hot=a[i];
    hotday=i;
    }
 if(cool>a[i])
   {
    cool=a[i];
    coolday=i;
  }
}
printf("\n average temperature=%f",avg);
printf("\n hottest temperature %f on day %d",hot,hotday);
printf("\n coolest temperature %f on day %d",cool,coolday);
getch();
}
```

Write program to print all combination of 1 2 3.

this requires nested loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,k;
clrscr();
for(i=1;i<=3;i++)
  for(j=1;j<=3;j++)
for(k=1;k<=3;k++)
  if(i!=j && j!=k && i!=k)
     printf("\n %d %d %d",i,j,k);
getch();
}
```

Write program using the function power(a,b) to calculate the value of a raised to b.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
float a,b,c;
float power(float,float);
printf("\n enter value of a and b");
scanf("%f %f",&a,&b);
c=power(a,b);
printf("\n value =%f",c);
getch();
}
float power (float a,float b)
{
return pow(a,b);
}
```

Write program to count no. of tabs, new lines, character and space of a file.

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
int i,ch=0,sp=0,tb=0,ln=0;
char rd[80];
fp=fopen("c:\\ram.txt","r");

while(fgets(rd,80,fp)!=0)
 {
   ln++;
   for(i=0;rd[i]!='\0';i++)
{
if(rd[i]=='\t')
```

```
  tb++;
else if(rd[i]==' ')
  sp++;
else
  ch++;
}
  }

fclose(fp);
printf("\n space=%d,tab=%d,newline=%d",sp,tb,ln);
}
```

Write program to create a file 'data' containing a series of integers and count all even numbers present in the file 'data'.

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
int i,even=0,odd=0,n;

fp=fopen("c:\\ram.txt","w");

printf("\n enter integers , control+z to end\n");
while(fscanf(stdin,"%d",&n)!=EOF)
 {
   fprintf(fp,"\t%d",n);
 }
fclose(fp);
fp=fopen("c:\\ram.txt","r");
while(fscanf(fp,"%d",&n)!=EOF)
 {
   printf("\t%d",n);
   if(n%2==0)
      even++;
   else
      odd++;
 }
 printf("\n no. of even=%d,no. of odd =%d",even,odd);
 getch();
}
```

What will be output statement to print 'welcome to programming world' in c and c++

ans:

in c:

```
#include<stdio.h>
#include<conio.h>
void main()
{
printf("welcome to programming world");
getch();
}
```

in c++:

```
#include<iostream.h>
```

```
#include<conio.h>
void main()
{
count<<"welcome to programming world";
getch();
}
```

Write statement to double the value of v and halve the value of x

ans:

```
v=v*2;
x=x/2;
```

if a program consists the statement scanf ("%3d",&n) and input is 1234567:
(a)what value is assigned to n (assuming n is integer):
ans 123
(b)what value will be assigned to n if control string reads scanf("%7d",&n);
ans since 1234567 is out of range of signed integer therefore it will store negative value.

Write program to accept two numbers and operator (+,-,*,/) and display the result using if-else.

```
#include<stdio.h>
#include<conio.h>
void main()
{
float a,b;
char ch;
printf("\n enter +, -, * ,/ as one of the operator ");
printf("\n enter expression as follows: operand1 operator operand2\n");
scanf("%f %c %f",&a,&ch,&b);
if(ch=='+')
  printf("\n addition=%f",a+b);
else if(ch=='-')
  printf("\n subtraction=%f",a-b);
else if(ch=='*')
  printf("\n multiplication=%f",a*b);
else if(ch=='/')
  printf("division=%f",a/b);
else
  printf("\n invalid operator");
getch();
}
```

Write a c program to read following matrix (row wise) and print the same column wise:
2 3 6
-7 0 8

ans:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[2][3];
int i,j;
printf("\n enter values in row measure order\n");
for(i=0;i<2;i++)
  for(j=0;j<3;j++)
```

```
    scanf("%d",&a[i][j]);
printf("\n the matrix in printed in column measure order\n");
for(i=0;i<3;i++)
  {
   for(j=0;j<2;j++)
    printf("\t%d",a[j][i]);
   printf("\n");
  }
getch();
}
```

What function is enables a user to input information while program is in execution?
ans : scanf, getch, getchar, getche ,gets ,fgets ,fscanf etc.

Write a program to read item number, rate and quantity from an inventory file and print the followings:
1. ***items having quantity > 5.***
2. ***total cost of inventory.***

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
struct item
{
int itemnumber;
int quantity;
float rate;
};

void main()
{
FILE *fp;
struct item e;
float cost=0;
int n;
fp=fopen("inv","wb");
if(fp==NULL)
  {
   printf("\n file open operation failed");
   exit (1);
  }

while(1)
{
 printf("\n enter itemnumber,quantity and rate:\n");
 scanf("%d %d %f",&e.itemnumber,&e.quantity,&e.rate);
 fwrite(&e,1,sizeof(e),fp);
 printf("\n enter more item if yes then enter 1,if no enter 0:");
 scanf("%d",&n);

 if(n==0)
    break;
}
fclose(fp);
fp=fopen("inv","rb");
while(fread(&e,1,sizeof(e),fp)!=0)
{
 cost=cost+e.rate*e.quantity;
 if(e.quantity>5)
   printf("\n item number=%d,item quantity=%d,item rate=%f",e.itemnumber,e.quantity,e.rate);
}
printf("\n total cost of inventory=%f",cost);
getch();
}
```

Write program of binary search .

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5],st=0,en=4,mi=(st+en)/2,val,i;
printf("\n enter five values in ascending order");
for(i=0;i<5;i++)
scanf("%d",&a[i]);

printf("\n enter value to search");
scanf("%d",&val);
for ( ;st<=en;mi=(st+en)/2)
{
if (val>a[mi])
   st=mi+1;
else if (val<a[mi])
  en=mi-1;
else
break;
}
if(st<=en)
 printf("\n value found at index %d",mi);
else
printf("\n value not found");
getch();
}
```