Practical No 1.

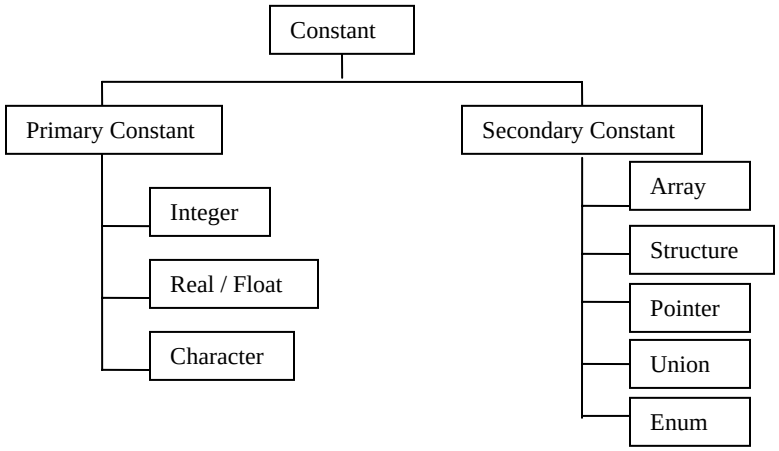
Write a C program to display hexadecimal, decimal and octal format of the entered number.

THEORY:

- C is a programming language developed at AT and T's Bell laboratories USA in 1972.
- C was designed and developed by 'Dennis Ritchie'.
- There are mainly two types of languages 1) High level languages and 2) Low level languages.
- High level languages are human understandable languages like English, Example COBOL .
- Low level languages interact with computer hardware. Example Assembly language.
- C is a high level language but some times it is also called as middle level language because it combines the feature of high level languages and functionality of low level languages.

CONSTANT:

- " A constant is a quantity that does not change ".
- The constant quantity can be stored at a memory location in.
- Example, 2x + 5y = 30,
  In this example since 2, 5 and 30 can not be changed, they are called as constants.
- **Types of Constants:**



VARIABLE:

- " Variable is an entity that may be changed ".
- Variable can be considered as a name given to the location in memory where constant is stored.
- Naturally constant of the variable can be changed.
- Example, 2x + 5y = 20, here x and y can vary or change hence called as variables.
- So we use variables to store data in memory for later use.
- All variables must be declared before they can be used.
- General form of declaring a variable is

```
                    datatype    variable name;
for ex,   int  j ;
          float b;
```

- **Rules for constructing variable name**:

1) A variable name can be any combination of alphabets and digits.
2) First character in variable name must be an alphabet.
3) No commas and blank spaces are allowed in a variable name.
4) No special symbol except underscore (_) can be used in a variable name. ex, gross_salary.
5) Do not use any if C reserved words i.e. Keywords.
6) Case is significant ex, AREA is a different variable and area is different variable as C is a case sensitive language.

KEYWORDS:

- In C, keywords are the reserved words used for its internal purpose.
- The meaning of keywords has been already explained to the C compiler.
- These keywords can not be used as a variable name.
- There are only 32 keywords available in C as follows,

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

DATA TYPE:

- The data is a collection of facts and data type is a specification provided with the variable name.
- The data types are used to specify what type of data are.
- In C data types are used to declare a variable before it is used.
- There are mainly three types of data type,

1) Primary data type: integer, float/real, character.
2) Derived data type: Array, structure, union etc
3) User defined data type: data types defined by user

- Following table shows memory requirements for different data types,

Data type	Size (in Bytes , 1 byte = 8 bits)	Range
char or signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
int or signed int	2 bytes	-32767 to 32767
unsigned int	2 bytes	0 to 65565
short int or unsigned short int	1 bytes	-128 to 127
long int or unsigned long int	4 bytes	0 to 2147483646
signed long int	4 bytes	-2147483647 to 2147483646
unsigned long int	4 bytes	0 to4294967295
float	4 bytes	3.4e-38 to 3.4e+38
double	8 bytes	1.7e-308 to 3.4e+308
long double	10 bytes	3.4e-4932 to 1.1e+4932

BASIC STRUCTURE OF C PROGRAM:

Document section : consists of set of comments line. This part is ignored by compiler

Link section **:** header files are declared here. Header files link functions in C library

Definition section **:** defines all symbolic constants used in program

Global declaration section **:** declares variables used in more than one function

main() function section **:** program execution starts with main() function

{

Declaration part **:** declares entire variables used in executable part

Executable part **:** contains set of statements

}

Subprogram section **:** consists of functions which are defined by user

Function1
Function2

RULES WHILE WRITING C PROGRAM:

1) Every program starts with header file declaration
2) Every program contains main() function. Its body starts with { and ends with }.
3) C is case sensitive language, hence each instruction should be written in lower case.
4) Extension of C program should be .c
5) Every c statement should terminate with semicolon except loop.
6) Any function should be written with parentheses i.e. ( ).
7) Variable used in program declared in main() or above main().
8) While initializing a variable single character will be closed in ''.

STEPS TO WRITE AND RUN C PROGRAM:

1) Open TC.exe
2) Click on file menu and choose "New" option.
3) Type C program in edition window.
4) Save your program by pressing "F2" and give desired file name along with the extension .c
5) To compile program press "Alt" and "F9" simultaneously.
6) If any error occurs remove those errors by making changes in the program.
7) To run the program press "Ctrl" and " F9".

DATA TYPE SPECIFIERS FOR printf() FUNCTION:

1) %c : print a single character.
2) %d : print a decimal integer.
3) %f : print a floating point value.
4) %e : print a floating point value
5) %g : print a floating point value
6) %h : print a short integer.
7) %i : print a decimal, hexadecimal or octal integer.
8) %o : print a octal integer.
9) %s : print a string.
10) %u : print an unsigned decimal integer.
11) %x : print a hexadecimal integer.
12) %p : print a pointer.

DATA TYPE SPECIFIERS FOR scanf() FUNCTION:

13) %c : reads a single character.
14) %d : reads a decimal integer.
15) %f : reads a floating point value.
16) %d : reads a short integer.
17) %o : reads an octal integer.
18) %s : reads a string
19) %u : reads unsigned decimal integer.
20) %l : reads long integer.
21) %x : reads hexadecimal integer.

// Program to display Hexadecimal, Decimal and Octal format of the entered number.

```
#include<stdio.h>
void main()
{       int i;
        printf("\nEnter the number");
        scanf("%d",&i);
        printf("\nHexadecimal Format : %x",i);
        printf("\nDecimal Format : %d",i);
        printf("\nOctal Format : %o",i);
        getch();
}
```

Practical No 2

Title: Write a C program to demonstrate all possible formatting specifiers.

Theory:

- Input and Output are the processes of accepting and displaying data from and to the users.
- The simplest mechanism is to read a character at a time with getchar().
- The drawback of getchar() is it buffers until any key is pressed.This leaves one or more characters waiting in the input queue.
- For input / output of strings, the functions gets() and puts() are used.
- **gets()** function reads a strings of characters entered through keyboardand places them at a certain address in memory.
- **puts()** function takes a string as its argument and writes it on screen.
- Apart from these we have formatted input and formatted output functions.
- **Formatting** means the way of displaying the message or result data on the screen.

Formatted Input:

- Formatted input means reading data in formats which are desired by the user.
- 'C' provides scanf() function which scans or reads formatted input.
- General Syntax:

  scanf("control string", arg1, arg2, arg3...argn);

  Here control string will be data type specifier and arguments will be the variables where the data is to be stored.Comma seperates the control string and arguments.The **&** is used before variable name.
- Example, to read values 10 and 56.78 and store them in variables n and m,

  scanf("%d%f", &n, &m);

  so, n will be 10 and m will be 56.78

Formatted Output:

- 'C' provides printf() function used for printing message and numerical values of variables.
- 1) printf() for printing message:
- General Syntax:

  printf("control string");

  Here control string may be a character, a string or a message that you want to be printed.
- Example: to printf the message "I am Indian",

  printf("I am Indian");
- 2) printf() for printing numerical values of variables:
- General Syntax:

  printf("control string", arg1, arg2, arg3...argn);

  Here control string will be data type specifier and arguments will be variables, whose value is to be printed.
- Example: To print value of variable a=10 and b=45.76,

  printf(" %d %f ", a, b);

Backslash Characters:

- 'C' provides backslash characters for formatting the output.
- Backslash characters are used in print() statement for changing fromat of output.
- Different Backslash characters in 'C' are,

Character	Meaning
\ b	Backspace
\ n	New Line
\ t	Horizontal Tab
\ v	Vertical tab
\ "	Double Quote
\ '	Single Quote
\ \	Backslash
\ r	Carriage return
\ 0	Null

Practical No 3

- **Title:** Write a C program to find greatest / smallest of 3 numbers.
- **Theory:**
- **Operators:**
    1. "The operators are the symbols used to indicate the operations on the operands."
    2. There are different types of operators which are used as per users need.
    3. The different types of operators available in 'C' are,

1) Arithmetic Operators:

"The arithmetic operators are used to do basic operations like addition, subtraction, multiplication, division and modulo division."

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

2) Relational Operators:

"The relational operators, also known as comparison operators are used to check the relation between two values or variables."

Operator	Meaning
<	Less than
>	Greater than
< =	Less than Equals to
> =	Greater than Equals to
.= =	Equals to
! =	Not Equals to

3) Logical Operators:

"A logical operator combines one or two conditions into a single new condition."

Operator	Meaning
&&	AND
||	OR
!	NOT

4) Assignment Operators:

"The assignment operator is used to assign some value to a variable."

Operator with example	Meaning
A = 20	20 is assigned to variable A
A + = B	A = A + B

5) Increment and Decrement Operators:

"The increment operator is used to increase value of operand by 1."

"The decrement operator is used to decrease value of operand by 1."

Operator	Meaning
A + +	A = A+1
A - -	A = A – 1
+ + A	Called as Preincrement / Predecrement Operators. Here value of A will be calculated first, then this value will be used in expression
- - A	

6) Conditional Operators:

1. "The conditional operator checks for the condition and evaluates the result on the status of that condition i.e either true or false "
2. It uses combination of **?:** symbols and called as ternary operator.

The general form is

Exp1 ? Exp2 : Exp3

Example: **max = a > b ? a : b**

Here first condition a > b will be checked.

If the result is true then max = a or if the result is false max = b

7) Bitwise operators:

1. "The bitwise operations refers to testing, setting or shifting actual bits in a byte or a word "
2. The bitwise operators can be used on character and integer data.
3. The bitwise operators can not be used on float, double, long double, void or other complex type.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Left shift
>>	Right shift
~	1's Complement

8) Special Operators:

"The special operators perform some special operations."

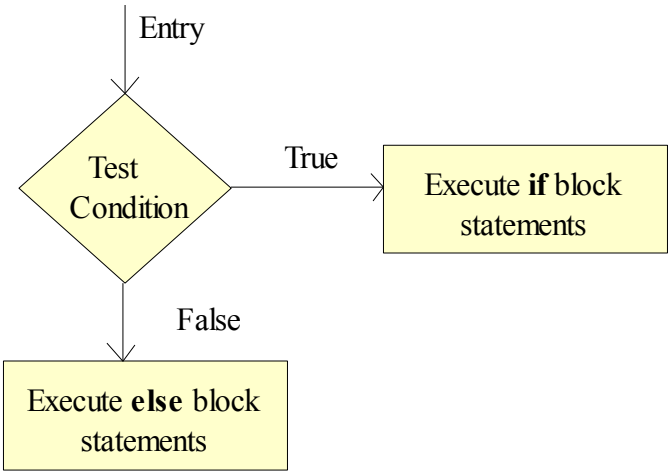
Operator with example	Meaning
&A	The address of A
*A	At the address A
sizeof vaiable	Returns length in bytes of variable or parenthesized data type.
sizeof (data type)	

- **The if-else statement:**
  1. Simple if statement can be used to check whether condition is true or not.
  2. In simple **if** statement only true part is included and it does nothing when condition is false.
  3. With the help of **if-else** statement, when condition is true one group of statements is executed and when the condition is false another group of statements is executed.

Syntax:

```
if ( condition )
  {
      statements;
  }
 else
  {
      statements;
  }
```

Flowchart:



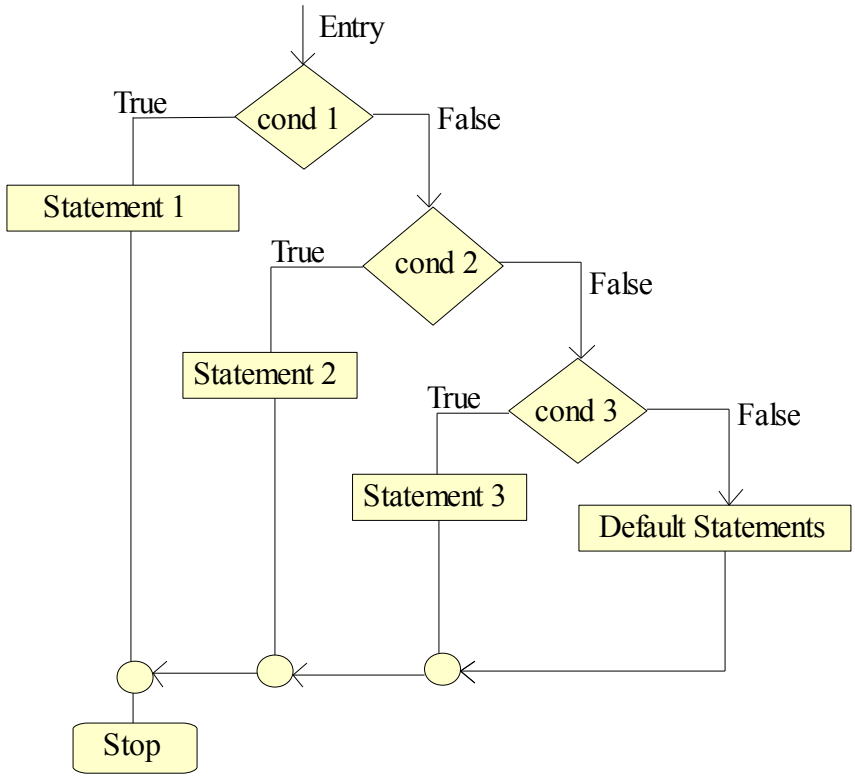
- **The if-else if ladder:**

The if-else if ladder is used when we want to check multiple conditions.

Syntax:

```
if ( condition1 )
  {
     statement1;
  }
else if( condition 2)
  {
     statement2;
  }
else if( condition 3)
  {
     statement3;
  }
else
  {
     default statements;
  }
```

Flowchart:



Practical No 4

Title: “ Write a C program to find even and odd numbers “

Theory:

- **The for loop:**
- The for loop is the most powerful loop.
- It allows to execute set of statements till certain condition is true.
- The for loop is used to repeat the action within the limit.

- Syntax:

```
for( initialize counter ; test condition ; increment/decrement counter )
  {
        statements
  }
```

- The for loop has three parts,

1. Initial condition from which loop execution starts.
2. Final condition which is the test condition. Loop run till the condition is true.
3. Step i.e. increment / decrement. This step directs loop how to proceed.

- Working of for loop:

The working of for loop is very simple.

1. Loop starts with the initial condition.
2. It iterates as per steps given i.e it repeats itself till the test condition is true.
3. Once condition becomes false, loop get break i.e control goes out of for loop.

- Example,

```
for( i=1 ; i<=5 ; i + +)
  {
     printf(“ \t %d ”,  i );
  }
```

- Output: **1 2 3 4 5**

Practical No 5

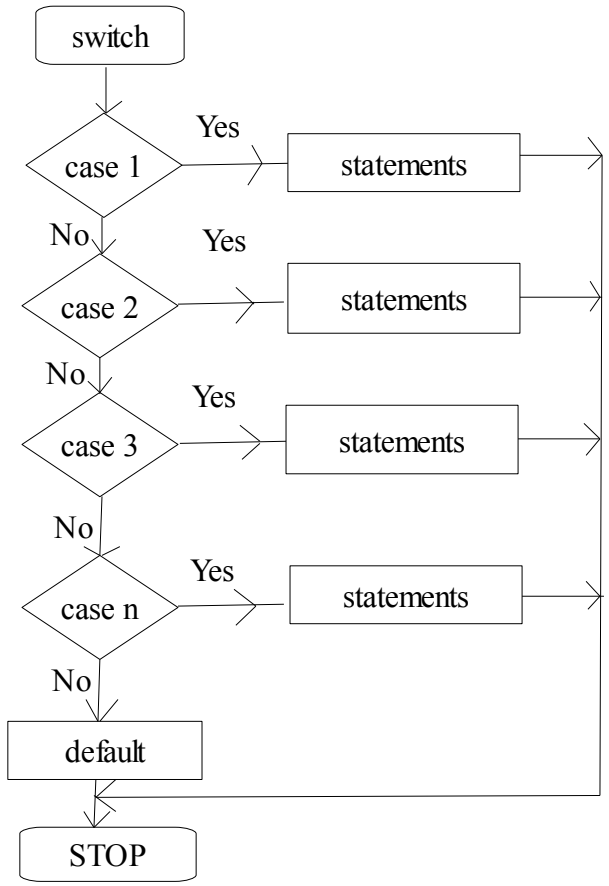
Title: Write a C program to display menu using switch-case.

Theory:

The switch-case statement:

- The **if** statement looks very easy if there are less number of conditions but if the conditions are more then if statements looks very confusing and complex.
- Another way to make a decision among multiple conditions is the switch-case statement.
- "The control statement which allows to make a decision from number of choices is called as switch-case statement"
- The switch-case statement accepts value and tests it for various conditions.if all conditions are falsethen default statements are executed.
- The different conditions in switch are called as case.Within these cases statements to be executed can be written.These cases are completed by '**break'** keyword.
- If any of these cases is true,statements written within same case are executed and case is completed by break keyword.
- In switch-case statements '**break'** is used to break control of switch itself when selected case executes.
- **Syntax of switch-case statenment:** **Flowchart:**

```
switch(choice)
 {
   case constant1:
                        statements;
                        break;
   case constant2:
                        statements;
                       break;
   case constant3:
                        statements;
                        break;
   case constant n:
                        statements;
                       break;
            default:
                      statements;
}
```



- The keyword case is followed by an integer constant or character constant which is matched with each case constant.Constant in each case must be different from all other cases.
- The above diagram is a pictorial representation of execution logic of switch-case.
- The conditions are cheked one by one. If condition is true statements of the selected case are executed else control passes to next case.

Practical No 6

Title: Write a C program to perform addition of 1-100 numbers.

Theory:

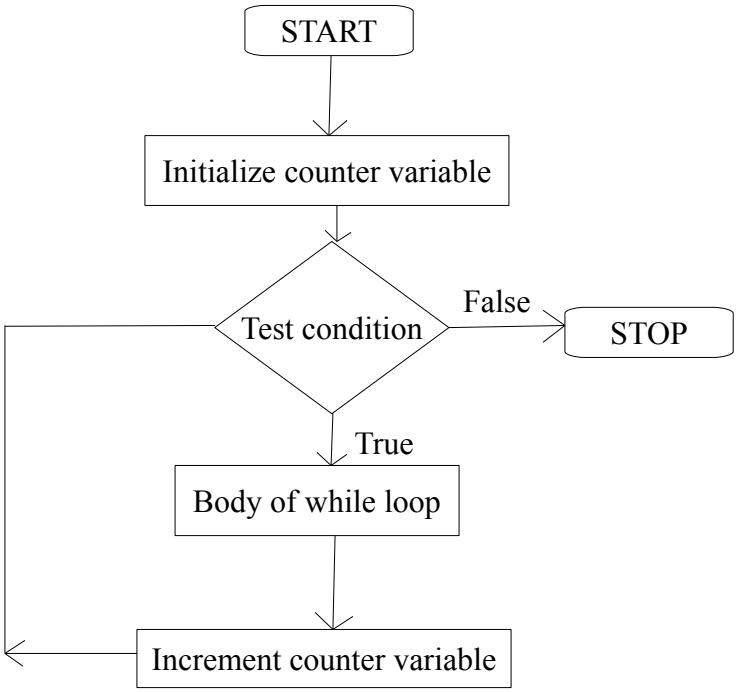
The while loop:

- The while loop is used fro the purpose of repetition.
- The while loop repeats itself till the given condition is true.
- Once condition becomes false, the loop gets break.
- For example if the user wants to print his college name 20 times, then there is no need to write printf() statement 1o times. Just write the printf() statement inside body of while loop and give the maximum condition of 10.
- **Syntax of while loop:**

```
initialize counter variable;
while( condition )
 {
    statements;

   increment counter variable;
 }
```

Flowchart:



- Example:

```
// C program to print college name 10 times
#include<stdio.h>
#include<conio.h>
void main()
 {
    int i =1;
    clrscr();

     while( i < = 10)
      {
          printf("\n Sandip Polytechnic ");
          i + +;
      }
  getch();
 }
```

Output:

```
Sandip Polytechnic
Sandip Polytechnic
Sandip Polytechnic
Sandip Polytechnic
Sandip Polytechnic
Sandip Polytechnic
Sandip Polytechnic
Sandip Polytechnic
Sandip Polytechnic
Sandip Polytechnic
```

Practical No 7

Title: Write a C program to find smallest / largest number from array elements.

Theory:

Array:

- In C language, there are three data types
  1) Basic data types
  2) Derived data types
  3) User defined data types
- Basic data types are integer, float and character. Array is collection of same types of elements but elements can be of only basic data types.
- Due to this arrays are called as derived data types. The base of derived data types is always basic data types i.e integer, float or character.

- **Definition of array:**
  " Array is a bounded collection of elements having same data type"
  The elements of array are stored in contiguous memory locations.

- **Types of array:**

  1) One dimensional array
  2) Two dimensional array
  3) Multidimensional array

- **One Dimensional Array:**
  One dimensional array has its element in one dimension i.e row.

- **Declaration of one dimensional array:**

  Syntax:

  data type arrayname [size] ;

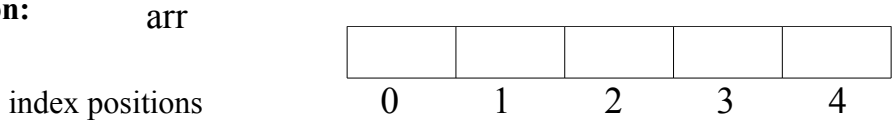
  here data type is data type of array elements and size is the capacity of array to hold elements.

  Example:

  int arr [5];

  This will declare integer array arr having capacity to hold 5 integer elements.
  The array elements have index positions. Array indexes always starts from 0.
  So, first array element has index 0 and last array element has index (arraysize -1).

- **Memory representation:**



- **Initialization of one dimensional array:**
  When any array is declared, in memory its locations are stored with garbage values.
  Garbage values are automatically stored on locations of array.
  Hence it is important to initialize array elements when array is declared.
  Initialization of array elements must be done at the time of array declaration.

  Syntax:

  data type arrayname [size] = { list of elements separated by comma } ;

  Example:

  int arr [5] = { 10, 56, 3, 40, 9 };

  This declares an integer array arr to contain 5 integer elements and initializes elements as 10, 56, 3, 40, and 9.

  arr [0] = 10;
  arr [1] = 56;
  arr [2] = 3;
  arr [3] = 40;
  arr [4] = 9;

  ○

- **Memory representation:**

arr	10	56	3	40	9
index positions	0	1	2	3	4

- **Reading and printing array elements with the help of for loop:**

```
//C program to read array elements using scanf()

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

void main()
 {
    int marks[5] , i ;
    clrscr();
    printf("\n Enter elements");

    for( i = 0; i < = 4; i + +)
     {
       scanf( "%d", & marks[i] );
     }

  getch();

 }
```

```
//C program to initialize and print array elements

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

void main()
 {
    int marks[5] = { 67, 78, 59, 80, 90 };
    int  i;
   clrscr();
   printf("\n The array elements are : ");

   for( i = 0; i < = 4; i + +)
    {
       printf( "%d",  marks[i] );
    }

  getch();

 }
```

Practical No 8

Title: Write a C program to calculate multiplication of 2 dimensional matrix..

Theory:

Two dimensional array:

- The 2 dimensional arrays are used to store the tabular data. This means when the data is in the form of rows and columns, we need powerful array to store it.
- Example,

ROLL NO	MARKS	RANKS
10	95	1st
4	87	2nd
55	79	3rd

- A two dimensional array has its element in two dimensions i.e rows and columns.
- A two dimensional array having 2 rows and 2 columns is also called as **Matrix**.
- **Declaration of two dimensional array:**
  Syntax:

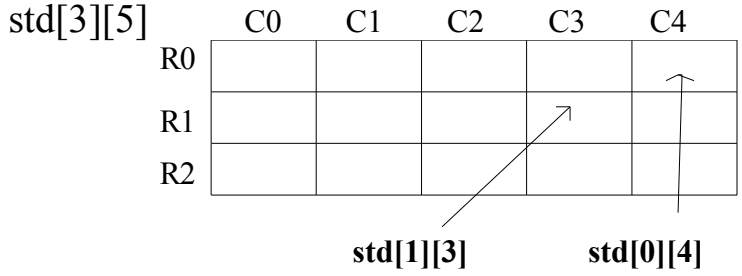
  data type arrayname [rows] [columns] ;

  here data type is data type of array elements, first square bracket indicates no. of rows and second square bracket indicates no. of columns.

  Example:

  int std [3] [5];

  This will declare a two dimensional integer array **std,** having 3 rows and 5 columns.
- **Memory representation:**



- **Initialization of one dimensional array:**
  Syntax:

  data type arrayname [rows] [columns] = { {1st row},{2nd row},.....,{nth row}};

  Example:

  int std [3] [5] = { {10,56,3,40,9} , {30,6,4,8,7} , {1,89,45,32,12} };

  This declares a 2 dimensional matrix to contain 15 integer elements and initialize elements as,

  std [0][0] = 10; std [0][1] = 56; std [0][2] = 3; std [0][3]= 40; std [0][4] = 9;
  std [1][0] = 30; std [1][1] = 6; std [1][2] = 4; std [1][3]= 8; std [1][4] = 7;
  std [2][0] = 1; std [2][1] = 89; std [2][2] =45; std [2][3]= 32; std [2][4] = 12;
- In above initialization, the internal brackets can be omitted. as,

  int std [3] [5] = { 10, 56, 3, 40, 9, 30, 6, 4, 8, 7, 1, 89, 45, 32, 12 };

- **Memory representation:**

std[3][5]	C0	C1	C2	C3	C4
R0	**10**	**56**	**3**	**40**	**9**
R1	**30**	**6**	**4**	**8**	**7**
R2	**1**	**89**	**45**	**32**	**12**

- **Reading and printing two dimensional array/matrix elements with the help of nested for loops:**
- Generally with two dimensional array we use nested for loops to read elements into array or to display elements from it.
- The outer for loop is used to count no. of rows and inner for loop is used to count no. of columns, so elements will be read or print row wise.

```
/*   C program to read elements in 3*5
     matrix using scanf()   */

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

 void main()
  {
     int std[3][5] , i , j ;
     clrscr();
     printf("\n Enter elements");

     for( i = 0; i < = 2; i + +)
      {
         for( j = 0; j < = 4; j + +)
          {
               scanf( "%d", & std[i][j] );
          }
     }

    getch();
  }
```

```
  /*   C program to initialize elements in 3*5
       matrix and print them using printf()   */

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

 void main()
 {
  int  std [3][5]={{10,56,3,40,9},{30,6,4,8,7},{1,89,45,32,12}};
  int  i , j ;
     clrscr();
     printf("\n The array elements are : ");
    for( i = 0; i < = 2; i + +)
       {
          for( j = 0; j < = 4; j + +)
           {
                printf( "%d \t ", std[i][j] );
           }
      printf( " \n ");
     }
    getch();
    }s
```

OUTPUT:

10	56	3	40	9
30	6	4	8	7
1	89	45	32	12

Practical No 9

Title: Write a C program to demonstrate output of standard library string functions.

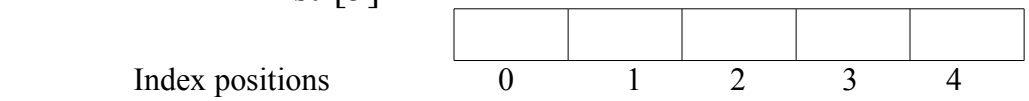
Theory:

String:

- "String is a collection of characters terminated by null character (\0)"
- When we write a word or sentence, it is treated as string.
- Strings are used to store text information and to perform operations on them.
- String is nothing but a character array.
- Each location of character array stores each single character of string.
- Example, "Sandip Polytechnic" is a string.

- **Declaration of string:**
  Syntax: **char stringname [size];**
  Here **char** is data type of string.
  A string having size n can actually hold (n-1) characters. Because nth character is \0.
- Example:
  char str [5];
  This will declare a string **str,** having capacity to store 4 elements.

- **Memory representation:**



- **Initialization of string:**
- Strings can be initialized by two ways:

1) initializing string with separate characters .
Syntax:

char stringname [size] = { 1st char, 2nd char, 3rd char, ……., nth char, '\0' };

Example: char str [7] = { 's', 'a', 'n', 'd', 'i', 'p', '\0' };

This declares a string **str** to contain 6 characters and initialize elements as,
str [0] = s; str [1] = a; str [2] = n; str [3] = d; str [4] = i; str [5] = p; str [6] = \0;
In above initialization, it is necessary to include null character '\0' at the end of string.

2) initializing string with collection of characters directly:
Syntax:

char stringname [size] = "complete string";

Example: char str [7] = "sandip";

This declares a string **str** to contain 6 characters and initialize elements as,
str [0] = s; str [1] = a; str [2] = n; str [3] = d; str [4] = i; str [5] = p; str [6] = \0;
In above initialization, there is no need to include null character '\0' at the end of string.
The compiler automatically includes '\0' at the end of string

- **Memory representation:**

str[5]

	s	a	n	d	i	p	\0
Index positions	0	1	2	3	4	5	6

- It is also not necessary to give string size at the time of initializing a string.
- The compiler automatically allocates memory to the string
- Example, char str[ ] = "sandip"; is allowed.

```
// program to read characters in string using scanf() and print string using printf()
#include<stdio.h>
#include<conio.h>
void main()
{
  char str[5];
  printf("\n Enter the string: ");
  scanf("%s", str);                    // & is not necessary
  printf("\n The string is : ");
  printf(" %s", str);
  getch();
}
```

Output of program:
Enter the string: SANDIP POLYTECHNIC
The string is : SANDIP

- **The gets() and puts() functions:**
- One limitation of character array is it terminates string at first blank space
- So in above program even we have entered string as SANDIP POLYTECHNIC, when we print the string output is SANDIP. Because character array terminates string at first blank space hence output string is SANDIP.
- To overcome on these disadvantages two functions are provided.

1) The gets() function: It reads entire string including blank spaces also
Syntax: **gets(stringname);**

2) The puts() function: It prints entire string.
Syntax: **puts(stringname);**

```
/* program to read characters in string using gets() and print the string using puts() */
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{ char str[5];
  puts(" Enter the string: ");
  gets(str);
  puts(" The string is : ");
  puts(str);
  getch(); }
```

Output of program:
Enter the string:
ABC PQR
The string is :
ABC PQR

- **String handling functions:**

1) strlen() :

"The function strlen() is used to calculate length of given string"

This function returns length of given string.

- Example: **char str1[ ]= "SANDIP";**
  int l;
  l = strlen(str1);
- This will store length of string str1 in variable l i.e. 6.

2) strcpy() :

"The function strcpy() copies one string into another string"

This function takes two arguments.

- Example: **char str1[] = "Sandip", str2[] = "Polytechnic";**
  strcpy(str1,str2);
- This will copy str2 into str1 i.e. contents of both string will be "Polytechnic".

3) strcmp() :

"The function strcmp() compares two string"

If strings are equal, it returns 0

If strings are not equal, it returns a non zero value i.e difference between two strings.

- Example: **char str1[] = "Sandip", str2[] = "Sandip";**
  strcmp(str1,str2);
- This will return value 0 meaning both strings are equal.

4) strcat() :

"The function strcat() is used to concatenate two strings"

It appends one string at the end of another string.

- Example: **char str1[] = "Sandip", str2[] = "Polytechnic";**
  strcat(str1,str2);
- This will append str2 at the end of str12 i.e. after concatenation str1 will be "SandipPolytechnic".

Practical No 10

Title: "Write a C program to calculate factorial of any given number using recursion"

Theory:

Function:

- Function is a subprogram or set of instructions written to do a particular task.
- Program written by the programmers may be too longer and also difficult to debug.
- If such a program is divided into subprograms and compiled separately, task becomes easy.
- These subprograms are called as functions.
- Basically functions are categorized into two types:
  1) Library functions: These are predefined functions defined by standard C library.
  Example, getch() , clrscr(), printf(), scanf() etc.
  2) User defined functions: These are defined by user.
  Example, add(), sub() etc.
- **Advantages of functions:**
  1) Function helps solving complex logical problems in the program.
  2) The length of the program get reduced because functions are called only whenever they are required in the program.
  3) The functions written once can be used many times.
  4) The function approach saves time and memory space.
- **Elements of user defined functions:**
- There are three main sections which are used while including the function in the program:
  1) Function declaration
  2) Function call
  3) Function definition
- **Function declaration:**
  Function declaration is similar to the variable declaration.
  Functions can be declared in the main() or above the main() function.
  The functions declared above main() are called as global functions and functions declared inside main() function are called local functions as they are accessible to only main().
  Syntax: **return_type function_name (list of data types of parameters) ;**
  Example, **int add(int , int);**
- **Function call:**
  Function call includes calling the function and passing actual parameters to that function.
  Syntax: **function_name (list of actual parameters) ;**
  Example, **add (5 , 8) ;**
- **Function definition:**
  In function definition, instructions are written inside function body.
  Syntax:

```
return_type function_name ( list of formal parameters )
 {      statements;
        return statement;
 }
```

  Example,

```
int  add ( int  x, int  y )
 {
      return  (x+y) ;
 }
```

- **Recursion:**
  " Calling a function inside definition of same function is called as recursion **".**

Practical No 11

Title: "Write a C program to demonstarte call by value, call by reference"

Theory:

- **Function Call:**

- **Function call is calling a function in the program by writing function name and passing the parameters if necessary.**
- **If the function is returning any value then, we can store the return value in the variable which is of the same data type of that return value.**
- **In the function call, the function can be called by two methods:**
  1) Call by value
  2) Call by reference

  1) Call by value:
- **In call by value, the arguments are passed to the function by value.**
- **This method copies the value of arguments into the formal parameters of the function.**
- **In this case changes made to the parameters have no effect on the value of arguments.**
- **Example:**

```
int n1=5,n2=10;
int add(n1,n2);
```

  here arguments stored in variables n1 and n2 are passed by value at the time of function call.

  2) Call by reference:
- **At the time of calling a function, if we pass references of variables of values , it is called call by reference.**
- **Refernce is the address of variable stored in the memory.**
- **This address id generally an integer value**
- **In call by reference, cahnges made to the arguments remains permanent.**
- **Example:**

```
int n1=5, n2=10;
int add(&n1,&n2);
```

  here arguments stored in variables n1 and n2 are passed by refernece i.e addresses of n1 and n2 are passed at the time of function call.

- **Actual arguments:**
  When we pass values to function at the time of function call, then these values are called as actual arguments.

- **Formal parameters:**
  When we recieve the values at the time of function definition , the variables in the function definition are called as formal parameters.

Practical No 12

- **Title:** Write a C program to maintain and manipulate student data using structure.
- **Theory:**

Limitation of array:

1. Array stores collection of elements of same data type.
2. Arrays allow user to store multiple values of same data type.
3. But limitation of array is, it cannot store elements having different data types.

Structure:

1. "Structure is a collection of one or more variables possibly of different data types, grouped together under a single name for convenient handling"
2. Structure allows user to have collection of different data type elements.
3. Example, if user wants to store the data related to books, structure can be used to store the information like, book_id, book_name, publication, price etc.
4. Structure is a small body which allows user to declare different variables of different data types and use them with the same structure.

General format for defining structure:

```
struct   tag_name
 {
    datatype     member1;
    datatype     member2;
    datatype     member3;
    ------------
    ------------
 };
```

where,

1. **struct** is a keyword used to declare structure
2. **tag_name** is the name of structure
3. All **members** (i.e variables) must be enclosed within { }

Example,

```
struct   book
 {
     int     book_id;
     char   book_name[20];
     float   price;
 };
```

Declaring structure variable:

Structure variables can be declared by two ways:

1)

```
struct   book
 {
     int     book_id;
     char   book_name[20];
     float   price;
 }b;
```

declaring structure variable

here, variable **b** is declared with declaration of structure

2)
```
struct   book
 {
     int     book_id;
     char   book_name[20];
     float   price;
 };

struct  book  b;
```
declaring structure variable

Initialization of structure variable:

Example,

```
struct   book
 {
     int     book_id;
     char   book_name[20];
     float   price;
 };

struct  book  b={1123,"PIC",100};
```
initializing structure variable

Accessing structure members:

Each member of structure can be accessed by using dot operator (.).

Example, to access member variables from structure book we can use,

```
b.book_id;
b.book_name;
b.price;
```

here, b is the structure variable and dot operator selects members from structure.

Array of structure:

When we would like to store more than one record in the memory, we can use array of structure.

Declaration of array of structure:

```
struct   tag_name
 {
     datatype   member1;
     datatype   member2;
      ------------------
} strcturevariablename [size];
```

Example,
```
struct   book
 {
     int     book_id;
     char   book_name[20];
     float   price;
 }b[3];
```

here, **b[3]** is array of structure which can store 3 records of books.
Each record contains book_id, book_name and price.

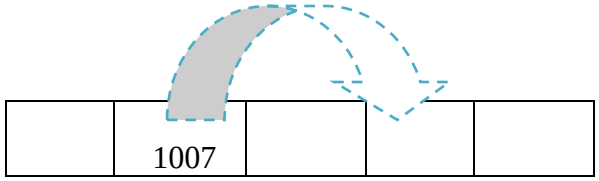
Practical No 13

- **Title:** Write a C program to perform 4 arithmetic functions on pointer
- **Theory:**

Pointer:

1. "Pointer is a variable that holds memory address of another variable"
2. If one variable contains memory address of another variable, the first variable is said to point to second variable.
3. Memory consists of millions of cells and each cell has got its own memory address. This location can be accessed by its address with the help of pointers.
4. So, pointers save memory space and time as they process the data very fast.

Memory representation:



1. Here the variable having address 1003 is pointing to variable whose address is 1007.
2. Meaning variable having address 1003 is storing address 1007.

Declaration and initialization of pointer variables:

1. A pointer declaration consists of base type, * operator and pointer variable name.
   Syntax:

   basetype * pointername ;

where, base type is type of variable to which pointer points and * means "value at address"

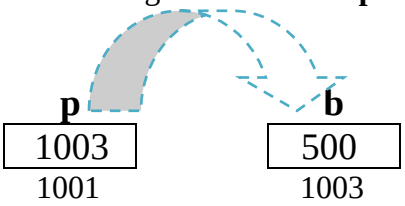
2. To initialize a pointer we can write,

   pointername = & variablename ;

   Example, **int * p ;**

Here p is the pointer variable whose base type in integer.

3. If b is an integer variable and **p = & b;** then



Hence , * p will give value stored at address 1003 i.e 500.

Important concepts:

To print address of variable which is stored in pointer, we can use %u and pointer name.
To print value stored at the address hold by pointer, we can use * pointername.

Pointer Arithmetic:

Pointer arithmetic concerns with the arithmetic operations with the pointers.
We can perform +,-,* and / operations on pointers.
As *pointername denotes value whose address is stored in pointer, some of the following operations are available.

```
Ex,     addition = (*p1) + (*p2) ;
        subtraction = (*p1) - (*p2) ;
        multiplication = (*p1) * (*p2) ;
        division =  (*p1) / (*p2) ;
                x = 5 * ( *p1 / * p2);
```

We can also increment pointer by p1 + + and decrement pointer by p1 - - .

Array of pointers:

1. Pointers and arrays are closely related to each other.
2. In case of arrays, pointers are very useful.
3. As array is collection of elements having same data type, array elements are stored in contiguous memory locations.
4. So, elements of array can be accessed by using pointers by storing elements base address.
   Example,

if, int *p;
int arr[10];
and p = & arr[0] ;

above statement will store base address of array element i.e. 0^{th} elements address into pointer p.

if, p = arr[1] ;

will store base address of array element at index 1 into pointer p.

```
// program to accept array elements and print them using pointers.
#include<stdio.h>
#include<conio.h>
void main()
 {
        int *p;
        int arr[5];
        int i=0;
        clrscr();
        printf("Enter array elements:");
        for(i=0;i<=4;i++)
          {
                scanf("%d",&arr[i]);
          }
        p = & arr[0];
        i = 1;
        while ( i<=5 )
         {
                printf("\n %d is stored at %u", *p, p);
                p++;
                i++;
         }
   getch();
 }
```

OUTPUT OF PROGRAM:

```
Enter array elements: 5  10  20  58  14
5 is at 65516
10 is at 65518
20 is at 65520
58 is at 65522
14 is at 65524
```