SHIV NADAR
UNIVERSITY
CHENNAI

[redacted]

LABORATORY RECORD

NAME : ABDULLAH SHERIFF

Reg. No. : [redacted]

Dept. [redacted] : CSE Sem. : 1ST SEM. Sec. : [redacted]

SHIV NADAR
UNIVERSITY
CHENNAI

BONAFIDE CERTIFICATE

Certified that this is the bonafide record [redacted] the practical work done in the

PROGRAMMING IN [redacted] Laboratory by

Name ABDULLAH SHERIFF.

Register Number [redacted]

Semester [redacted]

Branch AI & DS.

SHIV NADAR UNIVERSITY, Chennai

During the Academic year 2021-22

Faculty **Head of the Department**

Submitted for the.....Practical Examination held at SNUC on.....

Internal Examiner **External Examiner**

INDEX

Name : ABDULLAH SHERIFF Reg. No. [redacted]

Sem : 1ST SEM. Sec : [redacted]

Ex. No.	Date of Expt.	Title of the Experiment	Page No.	Signature of the Faculty	Remarks
01	27/09/21	COMPILING AND EXECUTING C PROGRAMS IN LINUX	01		
01	27/09/21	ENVIRONMENT.	01		
02	01/10/21	PROGRAMS USING I/O STATEMENTS AND EXPRESSIONS.	04		
03	04/10/21	PROGRAMS USING DECISION MAKING STATEMENTS.	09		
04	11/10/21	PROGRAMS USING LOOPING STATEMENTS.	13		
05	18/10/21	PROGRAMS USING 1D ARRAYS.	17		
06	22/10/21	PROGRAMS USING STRINGS.	23		
07	29/10/21	PROGRAMS USING 2D ARRAYS.	28		
08	12/11/21	PROGRAMS USING FUNCTIONS.	35		
09	22/11/21	PROGRAMS USING RECURSION.	42		
10	29/11/21	PROGRAMS USING POINTERS.	48		
11	20/12/21	PROGRAMS USING STRUCTURES AND UNIONS.	54		
12	27/12/21	FILE HANDLING.	62		

Ex. No: 1 27/09/21	**Compiling and Executing C Programs in Linux Environment**

Aim:

To learn basic Linux Commands and explore the Linux environment to compile and execute C Programs.

I) Basic Linux Commands

Open "Terminal" in Linux and practice Linux commands to do the following:

1. Create a new Directory

   `mkdir sample` - Creates a new directory named "sample"

2. Change Directory

   `cd sample` - Navigates to sub-directory names "sample"

   `cd ..` - Navigates to parent directory

3. Display the contents of a file

   `cat file1.txt` - Displays the contents of the file "file1.txt"

4. List the files in a directory

   `ls` - Lists all files in the current directory

5. Copy a file to another directory

   `cp file1.txt /home/sample` - Copies file1.txt to sample2 directory

6. Move a file to another directory

   `mv file1.txt /home/sample` - Moves file1.txt to sample2 directory

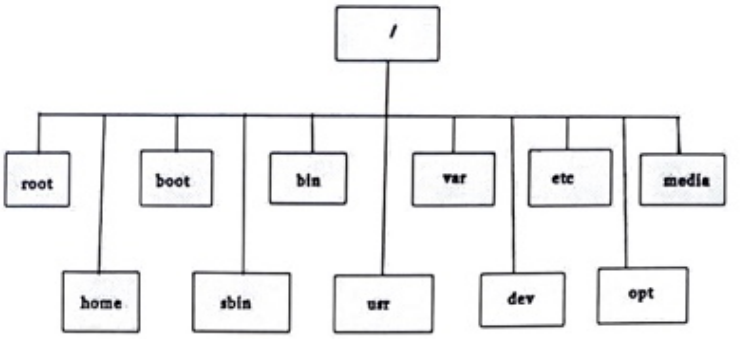
7. Rename a file

   `mv file1.txt file2.txt` - Renames file1.txt into file2.txt

8. Delete a file

   `rm file2.txt` - Deletes file2.txt

II) Linux File System Hierarchy



In the Linux file system, / is the root directory. All other directories are sub-directories of the root (/) directory. Some of the directories that come directly under root directory are 'var', 'etc', 'bin', 'home', 'media', 'dev', etc. 'home' directory is usually used by users to store their personal files. Individual 'home' directory is created for every user and it is not shared.

III) Creating and Executing C Programs

1. To navigate to your 'home' directory, type the following command in the terminal:

```
cd ~
```

2. In your home directory, create a new directory named "cprograms"

```
mkdircprograms
```

3. Create a file to type your C program using gedit command. The file name should be with a .c extension

```
gedit ex1.c
```

   Now the gedit editor window will open. Type your C program in this window and save. Once you have completed and saved the program, close the window. Back in the terminal, you will get the prompt only after you close the gedit window.

4. In the terminal, type the following command to compile the C program.

```
gcc ex1.c
```

5. If there are any errors in the program, they will be displayed now. Open gedit again as in step 3 to debug the errors in the program. After debugging, compile again using gcc command.

6. If there are no errors, run the program using the following command:

```
./a.out
```

   The program's output will now be printed.

Result:

Basic Linux commands were learnt and Linux environment was studied to compile and execute C programs.

Ex. No. 02.	PROGRAMS USING I/O STATEMENTS AND EXPRESSIONS.
01/10/21	

AIM:

To learn programs using input/output statements and expressions.

QUESTION 1:

Write a program to convert temperature from Fahrenheit to Celsius.

ALGORITHM:

#1 Start.
#2 Get Fahrenheit as input.
#3 Calculate Celsius = $\frac{5}{9}$ (Fahrenheit − 32).
#4 Display Celsius.
#5 End.

QUESTION 2:

Write a program to swap the values of two variables.

ALGORITHM:

#1 Start.
#2 Get input for variables 'a' and 'b'.

#3 Let a new variable 'c' = 'a'.

#4 Let a = b and b = c.

#5 Display 'a' and 'b'.

#6 End.

QUESTION 3:

Write a program that prints the ASCII value of a given sentence.

ALGORITHM:

#1 Start.

#2 Get 'input_char' as an input variable of 'char' type.

#3 Assign 'input_char' to an integer variable 'ascii_val'.

#4 Display 'ascii_val'.

#5 End.

Program 1:

```
/* Convert Fahrenheit to Celsius  */

#include <stdio.h>

int main() {
    int fahrenheit;
printf("Enter temperature (fahrenheit): ");
scanf("%d", &fahrenheit);
    double celsius = (5.0/9.0)*(fahrenheit - 32);
printf("Temperature in Celsius = %f degree(s)\n", celsius);
    return 0;
}

/*
Formula:
 C/100 = (F-32)/180
 (or)
 C = (5/9)*(F-32)
 */
```

INPUT / OUTPUT :

Enter temperature (Fahrenheit): 93

Temperature in Celsius = 33.888889 degree(s)

Program 2:

```
/* Swap the value of 2 variables  */

#include <stdio.h>

int main() {
    int a, b;
printf("Enter value for variable 'a': ");
scanf("%d", &a);
printf("Enter value for variable 'b': ");
scanf("%d", &b);
    int c;    // Dummy variable
    c = b;
    b = a;
    a = c;
printf("*** Swap Values *** \n");
printf("Value of variable 'a': %d \n", a);
printf("Value of variable 'b': %d \n", b);
    return 0;
}
```

INPUT / OUTPUT :

```
Enter value for variable 'a': 4
Enter value for variable 'b': 5
*** Swap Values ***
Value of variable 'a' : 5
Value of variable 'b' : 4
```

Program 3:

```
/* Print ASCII values of a given character */

#include <stdio.h>

int main() {
    char input_char;
printf("Enter a character: ");
scanf("%c", &input_char);    // 'char' datatype input -> '%c'
    int ascii_val = input_char;    // Integer value of input char
printf("The ASCII value is %d. \n", ascii_val);
    return 0;
}
```

RESULT:

Concepts related to programs using i/o statements and expressions were studied.

INPUT / OUTPUT :

Enter a character : $
The ASCII value is 36.

Ex. No. 03	PROGRAMS USING DECISION MAKING STATEMENTS.
04/10/21	

AIM:

To learn programs using decision making statements.

QUESTION 1:

Given the coordinates of a point (x, y), find if the point lies inside, outside or on the circumference of a circle centered in origin with radius 'r'. (Input should be (x, y) and 'r'. Output should be "inside", "outside" or "on the circle".)

ALGORITHM:

#1 Start.
#2 Get input 'x', 'y' and 'r'.
#3 Calculate distance from origin $= \sqrt{x^2 + y^2}$.
#4 If distance is equal to 'r'
    display "on the circle".
else if distance > 'r'
    display "outside".
else

| display "inside".

#5 End.

QUESTION 2:

Write a program to find if a triangle is equilateral, isosceles or scalene given the lengths of its 3 sides.

ALGORITHM:

#1 Start.

#2 Get length of side1, side2 and side3.

#3 If all three sides are equal

| display "Equilateral".

else if only any two sides are equal

| display "Isosceles".

else

| display "Scalene".

#4 End.

Program 1:

```
/*
Given the coordinates of a point (x,y) find if the point lies inside,
 outside or on the circumference of a circle centered in origin and with
radius 'r'.
 */

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

int main() {
    int x, y, r;
printf("Enter 'x' coordinate: ");
scanf("%d", &x);
printf("Enter 'y' coordinate: ");
scanf("%d", &y);
printf("Enter radius 'r' of circle: ");
scanf("%d", &r);

    int dist_from_origin = pow((pow(x, 2) + pow(y, 2)), 0.5);    // Distance
from origin = ((x-0)^2 + (y-0)^2)^(1/2)
    if (dist_from_origin != r)
    {
        if (dist_from_origin> r)
        {
printf("Point lies outside. \n");
        }
        else
        {
printf("Point lies inside. \n");
        }
    }
    else
    {
printf("Point lies on the circle. \n");
    }
    return 0;
}

/*
Distance between two points (x1, y1) & (x2, y2)
dist = ((x2 - x1)^2 + (y2 - y1)^2)^(0.5)
 */

/*
pow() is imported from <math> module
#include <math.h> -> Pre-processor Directive
 */
```

INPUT / OUTPUT :

Enter 'x' coordinate : 8

Enter 'y' coordinate : 6

Enter radius 'r' of circle : 10

Point lies on the circle.

Program 2:

```
/* Check whether triangle is scalene, isosceles or equilateral using side
lengths */

#include <stdio.h>

int main() {
    int s1 , s2, s3;
printf("Enter length of side 1: ");
scanf("%d", &s1);   // Do not forget &<varname> in scanf function
printf("Enter length of side 2: ");
scanf("%d", &s2);
printf("Enter length of side 3: ");
scanf("%d", &s3);

    if (s1 == s2 && s2 == s3)
    {
printf("Equilateral triangle.");
    }
    else
    {
        if ((s1 == s2 && s2 != s3) || (s2 == s3 && s3 != s1) || (s3 == s1 &&
s1 != s2))
        {
printf("Isosceles triangle.");
        }
        else
        {
printf("Scalene triangle.");
        }
    }
}

// && - Logical AND
// || - Logical OR

/*
Scalene - No sides of triangle are equal
Isosceles - Two sides of triangle are equal
Equilateral - All sides of triangle are equal
 */
```

RESULT:

Concepts related to programs using decision making statements were studied.

INPUT / OUTPUT:

```
Enter length of side 1: 5
Enter length of side 2: 6
Enter length of side 3: 5
Isosceles triangle.
```

Ex. No. 04.	PROGRAMS USING LOOPING STATEMENTS.
11/10/21	

AIM:

To learn programs using looping statements.

QUESTION 1:

Write a C program to print the multiplication table of a number.

ALGORITHM:

#1 Start.

#2 Get 'num'. Initialise variable 'prod' to 1.

#3 For values of 'i' from 0 to 12

prod = num × i.

display 'prod'.

#4 End.

QUESTION 2:

Write a program to generate twin prime numbers between 1 and 100. (If the difference between successive prime numbers is 2, they are known

as twin prime numbers. Eg. 3 and 5.)

ALGORITHM:

#1 Start.

#2 Set 'prev_prime' to 2.

#3 For 'i' from 2 to 100,

| for 'j' from 2 to 'i' (exclusive of 'i')

| | if 'i' is divisible by 'j'

| | | go directly to the next value of 'i'.

| if i - prev_prime is 2

| | display 'i' and 'prev_prime'.

| prev_prime = i

#4 End.

Program 1:

```
/* Print multiplication table of a number */

#include <stdio.h>

int main() {
      int num;
      printf("Enter number: ");
      scanf("%d", &num);
      printf("*** Multiplication Table *** \n");
      int i;
      int prod;
      for (i=1; i<=12; i++)
      {
            prod = num * i;
            printf("%d x %d = %d \n", num, i, prod);
      }
      return 0;
}
```

INPUT / OUTPUT :

```
Enter number: 13
*** Multiplication Table ***
13 × 1 = 13
13 × 2 = 26
13 × 3 = 39
13 × 4 = 52
13 × 5 = 65
13 × 6 = 78
13 × 7 = 91
13 × 8 = 104
13 × 9 = 117
13 × 10 = 130
13 × 11 = 143
13 × 12 = 156
```

Program 2:

```
/* Program to generate twin prime numbers between 1 and 100 */

#include <stdio.h>

int main() {
    int i, j, k;
    int prev_prime = 2;
printf("*** Printing Twin Prime Numbers *** \n");

    for (i=2; i<101; i++)
    {
        for (j=2; j<i; j++)
        {
            if (i%j!=0)
            {
                continue;
            }
            else if (i%j==0)
            {
goto next;
            }
        }

        if (i - prev_prime == 2)
        {
printf("%d & %d\n", i, prev_prime);
        }
prev_prime = i;

        next:
        continue;
    }

    return 0;
}

/*
If difference between two successive prime numbers is two (i.e.)
a - b = 2, (a & b are prime numbers), they are known as twin prime numbers.
Eg. 3 and 5
*/
```

RESULT:

Concepts related to programs using looping statements were studied.

INPUT / OUTPUT :

```
*** Printing Twin Prime Numbers ***
5 & 3
7 & 5
13 & 11
19 & 17
31 & 29
43 & 41
61 & 59
73 & 71
```

Ex. No. 05.	PROGRAMS USING 1D ARRAYS.
18/10/21	

AIM:

To learn programs using 1D arrays.

QUESTION 1:

Write a program to arrange a given set of numbers in ascending and descending order (sorting).

ALGORITHM:

#1 Start.

#2 Get 10 numbers as an array.

#3 For 10 iterations

for 'n' from 1 to 10

if 'n'th element > 'n+1'th element

swap 'n'th and 'n+1'th elements.

#4 Display array elements from index 0 to 9 (ascending) and 9 to 0 (descending).

#5 End.

QUESTION 2:

Given an array which contains values within a particular range (e.g. 0 to 5), write a program to count the frequency of occurence of each element.

ALGORITHM:

#1 Start.

#2 Get array of elements within a particular range.

#3 Declare 'freq-arr' to hold the frequency of times used of 6 elements (0 to 5).

#4 For each element in array
| increment value in 'freq-arr''s corresponding index by 1.

#5 Display index number 0 - 5 and corresponding values in respective indices of 'freq-arr'.

#6 End.

Program 1:

```
/* Program to arrange a given set of numbers in ascending and descending order
*/

#include <stdio.h>

int main()
{
    int arr[30] = {0};
    int i, j;
printf("*** Enter any 10 numbers *** \n");

    // Input array
    for (i=0; i<10; i++)
    {
printf("Enter element [%d]: ", i+1);
scanf("%d", &arr[i]);
    }

    // Bubble sort
    int tmp = 0;
    for (j=0; j<10; j++)
    {
        for (i=1; i<10; i++)
        {
            if (arr[i-1] >arr[i])
            {
tmp = arr[i-1];
arr[i-1] = arr[i];
arr[i] = tmp;
            }
        }
    }

    // Print ascending order
printf("Ascending Order \n");

    for (i=0; i<9; i++)
    {
printf("%d, ", arr[i]);
    }

printf("%d \n", arr[9]);

    // Print descending order
printf("Descending Order \n");

    for (i=9; i>0; i--)
```

```
    {
printf("%d, ", arr[i]);
    }

printf("%d \n", arr[0]);

    return 0;
}
```

INPUT / OUTPUT :

```
*** Enter any 10 numbers ***
Enter element [1] : 4
Enter element [2] : 7
Enter element [3] : 9
Enter element [4] : 13
Enter element [5] : 81
Enter element [6] : 7
Enter element [7] : 1
Enter element [8] : 3
Enter element [9] : 2
Enter element [10] : 10
Ascending Order
1, 2, 3, 4, 7, 7, 9, 10, 13, 81
Descending Order
81, 13, 10, 9, 7, 7, 4, 3, 2, 1
```

Program 2:

```
#include <stdio.h>

int main()
{
      int count;
      printf("Enter number of elements: ");
      scanf("%d", &count);
      printf("*** Enter elements from 0 - 5 *** \n");

      int arr[50] = {0};
      int freq_arr[6] = {0};
      int i;

      for (i=0; i<count; i++)
      {
printf("Enter element [%d]: ", i + 1);
scanf("%d", &arr[i]);

        if (arr[i] > 5 || arr[i] < 0) {
printf("Please enter a valid input between 0 - 5. \n");
i - 1;
        }
        else {
freq_arr[arr[i]] += 1;
        }
      }

      printf("*** Frequency of Occurence ***\n");

      for (i=0; i<6; i++)
      {
            printf("'%d': %d \n", i, freq_arr[i]);
      }

      return 0;
}

/*
      // Alternative
      int freq_arr[6] = {0};
      int num;
      printf("*** Enter numbers 0-5. Enter any other number to quit *** \n");
      while (1)
      {
            scanf("%d", &num);
            if (num> 5 || num< 0)
                  break;
```

```
            freq_arr[num]++;
        }

        int i;
        for (i=0; i<6; i++)
            printf("'%d': %d \n", i, freq_arr[i]);
*/
```

RESULT:

Concepts related to programs using 1D arrays were studied.

INPUT / OUTPUT :

```
Enter number of elements : 6
*** Enter elements from 0 - 5 ***
Enter element [1] : 0
Enter element [2] : 2
Enter element [3] : 3
Enter element [4] : 3
Enter element [5] : 3
Enter element [6] : 0
*** Frequency of Occurence ***
'0' : 2
'1' : 0
'2' : 1
'3' : 3
'4' : 0
'5' : 0
```

Ex. No. 06.	PROGRAMS USING STRINGS.
22/10/21	

AIM:

To learn programs using strings.

QUESTION 1:

Write a program to count the number of words in a given sentence.

ALGORITHM:

#1 Start.

#2 Get sentence.

#3 Set 'wordCount' to 1.

#4 For each character in the sentence

| if the character is a space (' ')

| increment 'wordCount' by 1.

#5 Display 'wordCount'.

#6 End.

QUESTION 2:

Write a program to check if a given string is a palindrome or not (First, reverse the string).

ALGORITHM:

#1 Start.

#2 Get string.

#3 Set 'i' and 'j' to 0. Create another empty string 'rev_string'.

#4 For each 'i'th character in the string

| if ascii value of character is between 65 & 90

| increment the ascii value by 32.

| assign the 'j'th character of 'rev-string' to 'length of string − i − 1'th character of original string.

| increment 'j' by 1.

#5 Display original string and rev-string.

#6 If both strings are same

| display "Palindrome".

else

| display "Not palindrome".

#7 End.

Program 1:

```
/*
Program to count the number of words in a given sentence
*/

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

int main()
{
    char sentence[100] = "";
printf("Enter sentence: ");
    gets(sentence);

    int i;
    int wordCount = 1;

    for (i = 0; i<strlen(sentence); i++)
    {
        if (sentence[i] == ' ')
        {
wordCount += 1;
        }
        else
        {
            continue;
        }
    }

printf("Sentence has %d word(s). \n", wordCount);

    return 0;
}
```

INPUT / OUTPUT:

```
warning: this program uses gets(), which is unsafe.
Enter sentence: Shiv Nadar University is located at
                Kalavakkam.
Sentence has 7 word(s).
```

Program 2:

```
/* Program to reverse a given string and check whether it is palindrome or
not. */

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

int main()
{
    // Read input
    char string[30] = "";
    char rev_string[30] = "";
printf("Enter string: ");
scanf("%s", string);

    // Reverse string + lowercase all characters
    int i, ascii_value;
    for (i = 0; i<strlen(string); i++)
    {
ascii_value = string[i];
        if (ascii_value>= 65 &&ascii_value<= 90)
        {
            string[i] = string[i] + 32;
        }
rev_string[strlen(string) - 1 - i] = string[i];
    }

    // Output
printf("Original : %s \nReverse  : %s \n", string, rev_string);
    if (strcmp(string, rev_string) == 0)    // strcmp() --> 0 - true, 1 -
false
    {
printf("It is palindrome. \n");
    }
    else
    {
printf("It is not palindrome. \n");
    }

    return 0;
}
```

```
/*
Palindrome - Number / word must be same when read forwards or backwards.
Eg. Level, refer, 1234321, etc.
 */
```

RESULT:

Concepts related to programs using strings were studied.

INPUT / OUTPUT :

Enter string : Malayalam
Original : malayalam
Reverse : malayalam
It is palindrome.

Ex. No. 07.	PROGRAMS USING 2D ARRAYS.
29/10/21	

AIM:

To learn programs using 2D arrays.

QUESTION 1:

Write a program to multiply two matrices. You need to check if the dimensions of the input matrices are compatible before proceeding to multiply.

ALGORITHM:

#1 Start.

#2 Get dimensions of matrix 'A' and 'B' as input.

#3 If number of columns of 'A' is not equal to the number of rows of 'B'
  display "Matrices are not compatible".
  end program.

#4 Get elements of 'A' and 'B' as input.

#5 Create a matrix 'C' with dimensions as number of rows of 'A' x number of columns of 'B'.

#6 For every 'i'th row
  for every 'j'th column.
    let sum = 0.
    for 'K' from 0 to number of columns of 'A'
      increment sum with 'A''s 'i x k'th element multiplied by 'B''s 'k x j'th element.
    Assign sum to 'C''s 'i x j'th element.

#7 Display elements of matrix 'C'.

#8 End.

QUESTION 2:

Write a program which would sort a list of names in alphabetical order.

ALGORITHM:

#1 Start.

#2 Get number of names.

#3 Create a list of names and get names as input (de-capitalize names while getting input).

#4 (Insertion sort)

For 'i' from 0 to name count times

| for each name in name list

| while ascii value of 'n+1'th name
| compares less than 'n'th name

| Swap 'n'th and 'n+1'th
| names.

#5 For each name in name list

| display name.

#6 End.

Program 1:

```
/*
Program to multiply 2 matrices. Check if dimensions of the input matrices are
compatible before proceeding to multiply.
*/

#include <stdio.h>

int main()
{
    int matrixA_rowCount;
    int matrixA_colCount;
    int matrixB_rowCount;
    int matrixB_colCount;

    // Input dimensions of matrices
printf("*** Matrix - A Dimensions *** \n");
printf("Enter number of rows: ");
scanf("%d", &matrixA_rowCount);
printf("Enter numer of columns: ");
scanf("%d", &matrixA_colCount);

printf("\n*** Matrix - B Dimensions *** \n");
printf("Enter number of rows: ");
scanf("%d", &matrixB_rowCount);
printf("Enter numer of columns: ");
scanf("%d", &matrixB_colCount);

    int matrixA[matrixA_rowCount][matrixA_colCount];    // Error if matrix
dimensions are less than requirement. Create matrix with dimensions > required
dimensions.
    int matrixB[matrixB_rowCount][matrixB_colCount];
    int matrixC[matrixA_rowCount][matrixB_colCount];

    // Check if dimensions are compatible
    if (matrixA_colCount == matrixB_rowCount)
    {
        int matrixC_rowCount = matrixA_rowCount;
        int matrixC_colCount = matrixB_colCount;
        int row, col;

        // Input matrix - A
printf("\n*** Matrix - A *** \n");

        for (row = 0; row <matrixA_rowCount; row++)
        {
            for (col = 0; col <matrixA_colCount; col++)
            {
```

```
printf("Enter element [%d][%d]: ", row + 1, col + 1);
scanf("%d", &matrixA[row][col]);
            }
        }

        // Input matrix - B
printf("\n*** Matrix - B *** \n");

        for (row = 0; row <matrixB_rowCount; row++)
        {
            for (col = 0; col <matrixB_colCount; col++)
            {
printf("Enter element [%d][%d]: ", row + 1, col + 1);
scanf("%d", &matrixB[row][col]);
            }
        }


        // Matrix - C
printf("\n*** Matrix - C = Matrix - A x Matrix - B *** \n");
        int sum, k;

        for (row = 0; row <matrixC_rowCount; row++)
        {
            for (col = 0; col <matrixC_colCount; col++)
            {
                sum = 0;

                for (k = 0; k <matrixA_colCount; k++)
                {
                    sum += matrixA[row][k] * matrixB[k][col];    // c11 =
(a11)*(b11) + (a12)*(b21) + (a13)*(b31) .... (a1k)*(bk1)
                }

matrixC[row][col] = sum;
printf("%d\t", matrixC[row][col]);
            }
printf("\n");
        }
    }

    else
    {
printf("\nMatrix dimensions are incompatible for multiplication. \n");
    }

    return 0;
}
```

INPUT / OUTPUT :

```
*** Matrix - A Dimensions ***
Enter number of rows : 2
Enter number of columns : 3

*** Matrix - B Dimensions ***
Enter number of rows : 3
Enter number of columns : 2

*** Matrix - A ***
Enter element [1][1] : 4
Enter element [1][2] : 13
Enter element [1][3] : 1
Enter element [2][1] : 6
Enter element [2][2] : 2
Enter element [2][3] : 3

*** Matrix - B ***
Enter element [1][1] : 6
Enter element [1][2] : 5
Enter element [2][1] : 6
Enter element [2][2] : 7
Enter element [3][1] : 6
Enter element [3][2] : 7

*** Matrix - C = Matrix - A x Matrix - B ***
108  118
66   65
```

Program 2:

```
/*
Program to sort a list of names in alphabetical order.
*/

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

int main()
{
    // char nameArray[25][25] = {{"Anjana"}, {"Adithi"}, {"Maanya"},
{"Cynddia"}, {"Dharani"}, {"Vaishnavi"},
    // "{Arvind"}, {"Akshat"}, {"Agilan"}, {"Akshay"}, {"Kamalnath"},
{"Abhay"}, {"Aakash"}, {"Abdullah"}};

    int nameCount;

printf("*** Alphabetical Order Sorting *** \n");
printf("Enter number of people: ");
scanf("%d", &nameCount);

    char nameArray[nameCount][20];
    int letter, ascii;
    int name;

    // Input
    for (name = 0; name <nameCount; name++)
    {
printf("Enter name [%d]: ", name + 1);
scanf("%s", nameArray[name]);

        for (letter = 0; letter < 25; letter++)
        {
            // De-capitalise
            ascii = nameArray[name][letter];
            if (ascii >= 65 && ascii <= 90)
            {
nameArray[name][letter] = ascii + 32;
            }
        }
    }

    // Sort names alphabetically
    int i;
    char temp[20] = "";

    // Insertion sort
    for (i = 0; i<nameCount; i++)
```

INPUT / OUTPUT :

```
*** Alphabetical Order Sorting ***
Enter number of people : 5
Enter name [1] : Harry
Enter name [2] : Daniel
Enter name [3] : Anna
Enter name [4] : Andy
Enter name [5] : Rita

*** Sorted List ***
Andy
Anna
Daniel
Harry
Rita

Program ended.
```

```
    {
        for (name = 0; name <nameCount; name++)
        {
            while (strcmp(nameArray[name], nameArray[name - 1]) < 0)
            {
                // Swap names
strcpy(temp, nameArray[name - 1]);
strcpy(nameArray[name - 1], nameArray[name]);
strcpy(nameArray[name], temp);
            }
        }
    }

    /*
    // Bubble sort
    for (i = 0; i<nameCount; i++)
    {
        for (name = 1; name <nameCount; name++)
        {
            if (strcmp(nameArray[name], nameArray[name - 1]) < 0)
            {
                // Swap names
strcpy(temp, nameArray[name - 1]);
strcpy(nameArray[name - 1], nameArray[name]);
strcpy(nameArray[name], temp);
            }
            else
            {
                continue;
            }
        }
    }
    */

    // Output
printf("\n*** Sorted List *** \n");

    for (name = 0; name <nameCount; name++)
    {
nameArray[name][0] -= 32;
printf("%s \n", nameArray[name]);
    }

printf("\nProgram ended. \n");

    return 0;
}
```

RESULT:

Concepts related to programs using 2D arrays were studied.

INPUT / OUTPUT :

```
*** Alphabetical Order Sorting ***
Enter number of people : 5
Enter name [1] : Karthik
Enter name [2] : Aneesha
Enter name [3] : Lohith
Enter name [4] : Keerthana
Enter name [5] : Aniruth

*** Sorted List ***
Aneesha
Aniruth
Karthik
Keerthana
Lohith

Program ended.
```

Ex. No. 08.	PROGRAMS USING FUNCTIONS.
12/11/21	

AIM:

To learn programs using functions.

QUESTION 1:

Write a program to check if a given number is a strong number or not. A strong number is a number that equals the sum of factorials of its digits. Write a function to calculate factorial.

ALGORITHM:

#1 Start.

#2 Create a recursive factorial function.

#3 Get number as input and set 'sum' to 0.

#4 While number > 0

increment factorial of last digit of number (least significant digit) to sum. (Call the factorial function.)

divide 'num' by 10.

```
#5  If 'num' == 'sum'
        | display "strong number".
    else
        | display "Not a strong number".
#6  End.
```

QUESTION 2:

Write a program to find the roots of a quadratic equation. You must write your own function for finding square root. The algorithm for finding square root of a number 'n' is as follows:

Step 1: Make a guess (g) (you may use rand() function).

Step 2: Divide the original number 'n' by 'g'.

Step 3: Find the average of 'g' and the quotient obtained in step 2.

Step 4: Use average as the next guess and repeat steps 2 to 4 until the algorithm converges (when there is not much difference between the previous guess and the next guess).

ALGORITHM:

#1 Start.

#2 Create a square root function using algorithm given in question.

#3 Get co-efficient of x^2, x and 1 as input.

#4 Calculate discriminant = $b^2 - 4ac$. (Where 'a' is co-efficient of x^2, 'b' is co-efficient of x and 'c' is co-efficient of 1.)

#5 If discriminant is 0

display "Two real and identical roots exist".

Calculate root = $-b/2a$.

Display root.

else if discriminant is greater than 0

display "Two real and distinct roots exist".

calculate root1 = $\frac{-b+\sqrt{b^2-4ac}}{2a}$ & root2 = $\frac{-b-\sqrt{b^2-4ac}}{2a}$.

display root1 and root 2.

else

display "No real roots exist."

real part = $-b/2a$ & imaginary part = $\frac{\sqrt{\text{discriminant}}}{2a}$.

display 2 complex roots (real + imag. & real - imag.)

#6 End.

Program 1:

```
/*
Program to check if a given number is strong number or not.
A strong number is a number that equals the sum of factorials of its digits.
Function to calculate factorial is included.
*/

#include <stdio.h>

int fact(int num) {
    if (num == 0) {
        return 1;
    }
    else {
        return num * fact(num-1);
    }
}

int main() {
    int inputNum, num;
    int sum = 0;

    // Input
printf("*** Strong Number Or Not *** \n");
printf("Enter a number: ");
scanf("%d", &inputNum);

    // Strong number or not
num = inputNum;
    while (inputNum> 0) {
        sum += fact(inputNum%10);
inputNum /= 10;
    }

    // Output
    if (num == sum) {
printf("%d is a strong number. \n", num);
    }
    else {
printf("%d is not a strong number. \n", num);
    }

    return 0;
}
```

INPUT / OUTPUT:

```
*** Strong Number Or Not ***
Enter a number : 40585
40585 is a strong number.
```

Program 2:

```
/*
Program to find the roots of a quadratic equation. Function for finding square
root is included.
The algorithm for finding square root of a number 'n' is as follows:

Step 1: Make a guess (g) (using rand() function)
Step 2: Divide the original number 'n' by 'g'
Step 3: Find the average of 'g' and the quotient obtained in step 2
Step 4: Use the average as the next guess and repeat steps 2 to 4 until the
algorithm converges
(when there is not much difference between the previous guess and the current
guess).
*/

#include <stdio.h>
#include <stdlib.h>

float squareRoot(float num) {
    int i;
    float average;
    float guessNum = rand();

    for (i = 0; i< 10000; i++) {
        average = ((num / guessNum) + guessNum) / 2;
guessNum = average;
    }

    return guessNum;
}

// 'num' power 'a' ('a' - whole number)
float power(float num, int a) {
    if (a == 1) {
        return num;
    }
    else {
        return num * power(num, a - 1);
    }
}

int main() {
    // Quadratic Equation (highest power of x is 2) -> ax^2 + bx + c = 0
    float a, b, c;    // a -> Co-efficient of x^2, b -> Co-efficient of x, c -
> Co-efficient of 1 (Constant)

    // Input
printf("*** Root(s) of a Quadratic Equation *** \n");
```

```
printf("Enter co-efficient of x^2: ");
scanf("%f", &a);
printf("Enter co-efficient of x: ");
scanf("%f", &b);
printf("Enter co-efficient of 1: ");
scanf("%f", &c);

    int discriminant = power(b, 2) - (4*a*c);    // (b^2 - 4*a*c)

    // Quadratic formula (gives roots of quadratic equation) is -> [-b +/-
sqrt(b^2 - 4*a*c)] / 2*a

    if (discriminant == 0) {
        // One real root
printf("Two real and identical roots exist. \n");
        float root = - b / (2*a);
printf("The real root is %lf. \n", root);
    } else if (discriminant > 0) {
        // Two distinct real roots
printf("Two real and distinct roots exist. \n");
        float root1 = (-b - squareRoot(discriminant)) / (2 * a);
        float root2 = (-b + squareRoot(discriminant)) / (2 * a);
printf("The real roots are %lf and %lf. \n", root1, root2);
    } else {
        // Two complex roots
printf("No real roots exist. \n");
        discriminant = - discriminant;    // i = sqrt(-1)
        float real = - b / (2 * a);
        float imag = squareRoot(discriminant) / (2 * a);
printf("The imaginary roots are %lf+(%lf)i and %lf-(%lf)i. \n", real, imag,
real, imag);
    }

    return 0;
}
```

RESULT:

Concepts related to programs using functions were studied.

INPUT / OUTPUT:

```
*** Root(s) of a Quadratic Equation ***
Enter co-efficient of x^2 :  2
Enter co-efficient of x : -6
Enter co-efficient of 1 :  12
No real roots exist.
The imaginary roots are 1.500000+(1.936492)i and
1.500000-(1.936492)i.
```

Ex. No. 09.	PROGRAMS USING RECURSION.
22/11/21	

AIM:

To learn programs using recursion.

QUESTION 1:

Write a recursive function to count the number of digits in a number. Modify the function to calculate the sum of digits of a number.

ALGORITHM:

#1 Start.

#2 Get number.

#3 Create 'digits()' function which takes number as input to calculate number of digits.

If number is zero
    return 0.
else
    return 1 and call 'digits()' function with 'number %10' as argument.

#4 Create 'sumOfDigits()' function which takes number as input to calculate and return the sum of digits.

If number is zero
    return 0.
else
    return remainder of 'number % 10' and call sumOfDigits function recursively with 'number % 10' as argument.

#5 Pass number into both functions and display output.

#6 End.

QUESTION 2:

Write a recursive function to print the elements of an array.

ALGORITHM:

#1 Start.

#2 Get the array of elements.

#3 Create recursive function 'printArr()' which takes a pointer to the array's first element and number of elements as arguments.

#4 If length of array is 1
| print element in index 'array length - 1'.
else
| print element in '0'th index.
| return array pointer pointing to next element
| inside the array. Decrement array length by 1.

#5 Call function.

#6 End.

Program 1:

```
/*
Program to count the number of digits in a number using recursive functions.
The function is then modified calculate the sum of digits of the number.
*/

#include <stdio.h>

int digit(int num) {
if (num == 0) {
return 0;
        }
else {
return 1 + digit(num/10);
        }
}

int sumOfDigits(int num) {
if (num == 0) {
return 0;
    }
else {
return (num % 10) + sumOfDigits(num/10);
    }
}

int main() {

        // Input
int num;
printf("*** Number of Digits in a Number *** \n");
printf("Enter number: ");
scanf("%d", &num);

        // Output
printf("There are %d digits. \n", digit(num));
printf("Sum of digits is %d. \n", sumOfDigits(num));
return 0;
}
```

INPUT / OUTPUT :

```
*** Number of Digits in a Number ***
Enter number: 57241
There are 5 digits.
Sum of digits is 19.
```

Program 2:

```
/*
Program with a recursive function to print the elements of an array.
*/

#include <stdio.h>

void printArr(int array[], int arrLen)
{
	// Base case
	if (arrLen == 1) {
		printf("%d \n", array[arrLen-1]);
	}
	else {
		printf("%d ", array[0]);
		return printArr(array + 1, arrLen-1);	// array + 1 -> Returns
array from 1st index position. Eg. [2, 3, 4, 5] -> [3, 4, 5]
	}
}

int main() {
	// Input
	int numOfElements, element;
	printf("Enter number of elements: ");
	scanf("%d", &numOfElements);

	int array[numOfElements];

	for (element = 0; element <numOfElements; element++) {
		printf("Enter element [%d]: ", element + 1);
		scanf("%d", &array[element]);
	}

	// Output
	printf("*** Printing Elements *** \n");
	printArr(array, numOfElements);

	return 0;
}

/*
array + n = (Address of array's first element) + (n * (Number of bytes in
array's datatype)) -> Points to array's 'n+1'th element
Eg.
int array = [1, 2, 3, 4, 5, 6]
(Eg. Array's address is 1000)
array + 4 -> array's address + 4 * 4 ('int' datatype size is 4 bytes) ->
array's address + 16 -> 1000 + 16 -> 1016
```

```
1020 is the address of 5th element (or) 4th index position of array.

Simply, it points to 'n+1'th element of the array.
*/
```

RESULT:

Concepts related to programs using recursion were studied.

INPUT / OUTPUT :

```
Enter number of elements : 7
Enter element [1] : 41
Enter element [2] : 8
Enter element [3] : 27
Enter element [4] : 6
Enter element [5] : 3
Enter element [6] : 9
Enter element [7] : 13

*** Printing Elements ***
41 8 27 6 3 9 13
```

Ex. No. 10.	PROGRAMS USING POINTERS
29/11/21	

AIM:

To learn programs using pointers.

QUESTION 1:

Write a program to swap the values of two numbers using pass-by-reference.

ALGORITHM:

#1 Start.

#2 Get values of two integer variables.

#3 Create two pointer variables pointing to the addresses of the two input variables.

#4 Create a function with two pointers as paramet-ers.

#5 Pass the two pointers into the function.

#6 Inside the function, create a temporary variable.

#7 temp. var. = variable pointed by ptr1's value.
variable value pointed by ptr 1 = var. val. pointed by ptr 2.
variable val. pointed by ptr2 = temp. variable's value.

#8 Display swapped variable values.

#9 End.

QUESTION 2:

Pass a float array comprising of five elements to a void function named 'reverseArray()'. Modify the contents of the array in such a way that the original array is reversed by making use of a 'swap()' function called within 'reverseArray()'. The 'swap()' function takes the pointer to two elements which should be swapped so as to reverse the original array.

ALGORITHM:

#1 Start.

#2 Create a recursive swap function using pass-by-reference.

#3 Create a reverse array function.

For each 'n'th element in array,

swap 'n'th index element and 'length of array - 1'th index element using swap function.

#4 Get array of elements as input. Pass array to the reverse array function.

#5 For each element in reversed array
    | display element.

#6 End.

Program 1:

```
/*
Program with a function to swap two numbers. Parameters of the function are
pointer variables.
*/

#include <stdio.h>

// int swap(<parameter(s)>)
void swap(int *ptr1, int *ptr2) {
    int temp;
    temp = *ptr1;    // *ptr1 = value at the address stored in ptr1
    *ptr1 = *ptr2;
    *ptr2 = temp;
}

int main()
{
    int a, b;
    printf("Enter value of 'a': ");
    scanf("%d", &a);
    printf("Enter value of 'b': ");
    scanf("%d", &b);

    int *ptr1, *ptr2;    // Use '*' to declare 'ptr1' and 'ptr2' as pointer
variables

    // Store address of 'a' and 'b' in pointer variables 'ptr1' amd 'ptr2'
respectively
    ptr1 = &a;
    ptr2 = &b;

    // Output
    printf("Before swapping: a = %d, b = %d \n", a, b);
    swap(ptr1, ptr2);    // swap(<argument(s)>)
    printf("After swapping: a = %d, b = %d \n", a, b);
    return 0;
}

/*
Every memory location has an address.
'a = 5;' - When a variable is declared, a name is added to memory location.
'int *ptr; ptr = &a;' - Pointer variables store a memory location's address.
(using '&' - address of operator)
*ptr = 10; - '*' - De-referencing operator is used to access the value at the
address stored in the pointer variable.
So, value of a is now 10.
*/
```

INPUT / OUTPUT :

Enter value of 'a' : 81
Enter value of 'b' : 3
Before swapping : a = 81 , b = 3
After swapping : a = 3 , b = 81

Program 2:

```
/*
Pass a float array comprising of five elements to a void function named
reverseArray( ).
Modify the contents of the array in such a way that the original array is
reversed by making use of a swap( ) function called within reverseArray( ).
The swap( ) function takes the pointer to two elements which should be swapped
so as to reverse the original array.
*/

#include <stdio.h>

void swap(float *ptr1, float *ptr2) {
    float temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
}

void reverseArray(float *array) {
    int i;
    for (i = 0; i< 5 / 2; i++) {
        swap((array + i), (array + (4-i)));
    }
}

int main() {

    // Input
    int i;
    float arr[5];
    printf("*** Floating-Point Array *** \n");
    for (i = 0; i< 5; i++) {
        printf("Enter element [%d]: ", i + 1);
        scanf("%f", (arr+i));
    }

    printf("*** Original Array *** \n");
    printf("arr[5] = {");
    for (i = 0; i< 4; i++) {
        printf("%f, ", *(arr + i));
    }

printf("%f} \n", *(arr+4));    // *(arr + 4) -> Jump 4 times to the right (+)
= 5th element

// Reverse array
reverseArray(arr);    // Function call
```

```
printf("*** Reversed Array *** \n");
printf("arr[5] = {");
    for (i = 0; i< 4; i++) {
printf("%d, ", *(arr + i));
    }
printf("%d} \n", *(arr+4));

    return 0;
}
```

RESULT:

Concepts related to programs using pointers were studied.

INPUT / OUTPUT :

```
*** Floating-Point Array ***
Enter element [1]: 5.3
Enter element [2]: 62.8
Enter element [3]: 41.0
Enter element [4]: 5.79
Enter element [5]: 3.14
*** Original Array ***
arr[5] = {5.300000, 62.799999, 41.000000, 5.790000
                                , 3.140000}
*** Reversed Array ***
arr[5] = {3.140000, 5.790000, 41.000000, 62.799999
                                , 5.300000}
```

Ex. No. 11.	PROGRAMS USING STRUCTURES AND UNIONS.
20/12/21	

AIM:

To learn programs using structures and unions.

QUESTION 1:

Write a program to add 2 distances given in feet and inches. Feet and inches must be stored in a structure called distance.

ALGORITHM:

#1 Start.

#2 Create a structure 'distance' with two members 'feet' and 'inches'.

#3 Get values of two structure variables 'd1' and 'd2'.

#4 Variable 'feet' = d1.feet + d2.feet + (d1.inches + d2.inches) / 12.

#5 Variable 'inches' = (d1.inches + d2.inches) % 12.

#6 Display sum of two distances in feet and inches.

#7 End.

QUESTION 2:

Write a program to create an array of numbers. The numbers may be of any format (number name, whole number, floating-point number, etc.). For this, create a union named 'Number' with fields char number[], int num, float fnum, double dnum. Each number may be stored and retrieved in any of the above formats depending on the choice of the user.

ALGORITHM:

#1. Start.

#2 Create a union with 4 members of type string, integer, float and double.

#3 Get number of elements.

#4 Display datatypes in the form of a menu.

#5 For 'i' number of times from 0 to number of element -s, ask for input type from user.

if type is string

get string input and store in member variable's index 'i'.
Save type array's index 'i' as "s".

else if type is integer

get integer input and store in member variable's (array) index 'i'.
Save type array's index 'i' as "i".

else if type is float

get float input and store in member variable's index 'i'.
Save type array's index 'i' as "f".

else

get double input and store in member variable's index 'i' [member variable is an array of unions].
Save type array's index 'i' as 'd'.

#6 Display elements of respective types.
For 'i' from 0 to number of elements

if type array's index 'i' element is "s"

Print string type data stored in member variable's index 'i'.

else if type is integer "i"

print integer type data stored in member variable's index 'i'.

else if type is floating point "f"

print floating point data stored in member variable's index 'i'.

else

print double type data stored in member variable's index 'i'.

#7 End.

Program 1:

```
/*
Program to add 2 distances given in feet and inches.
Feet and inches must be stored in a structure called distance.
*/

#include <stdio.h>

struct distance
{
    int feet;
    int inches;
};

int main() {
    struct distance d1, d2;

    // Input
printf("***** Sum of Distances (in feet & inches) Calculator ***** \n");
printf("*** Distance - 1 *** \n");
printf("Enter feet value: ");
scanf("%d", &d1.feet);
printf("Enter inch value: ");
scanf("%d", &d1.inches);

printf("*** Distance - 2 *** \n");
printf("Enter feet value: ");
scanf("%d", &d2.feet);
printf("Enter inch value: ");
scanf("%d", &d2.inches);

    // Sum of distances
    int feet = (d1.feet + d2.feet) + (d1.inches + d2.inches) / 12;
    int inches = (d1.inches + d2.inches) % 12;

    // Output
printf("Sum is %d feet and %d inches. \n", feet, inches);

    return 0;
}

/*
Note: 1 feet = 12 inches
('feet' is a standard unit)

Imperial system - Used only in America, Liberia & Myanmar - Use feet, inches,
pounds, ounces, etc.
*/
```

INPUT / OUTPUT :

```
***** Sum of Distances (in feet & inches) Calculator *****
*** Distance - 1 ***
Enter feet value : 4
Enter inch value : 11
*** Distance - 2 ***
Enter feet value : 7
Enter inch value : 8
Sum is 12 feet and 7 inches.
```

Program 2:

```
/*
Program to print an array of numbers.
The numbers may be of any format (number name, whole number, floating point
number, etc.).
For this, a union named 'Number' is created with fields char number[], int
num, float fnum, double dnum.
Each number may be stored and retrieved in any one of the above formats
depending on the choice of the user.
*/

#include <stdio.h>

union Number{
    char number[50];
    int num;
    float fnum;
    double dnum;
} num1[100];    // Array of numbers

int main() {
    // Input
    int numOfElts;
printf("Enter number of elements: ");
scanf("%d", &numOfElts);

    // Store numbers in any format
    int option;
printf("\n** Input Datatype Menu ** \n");
printf("1. Whole number (E.g. 3). \n");
printf("2. Number name (E.g. Three). \n");
printf("3. Floating-point (E.g. 3.14). \n");
printf("4. Double (E.g. 3.14159265358). \n");

    char typeArr[numOfElts];    // Store datatype of number (int, string,
float or double)

    for (int i = 0; i<numOfElts; i++) {
printf("\nEnter option: ");
scanf("%d%*c", &option);

        switch(option) {
            case 1:
printf("Enter whole number: ");
scanf("%d%*c", &num1[i].num);
typeArr[i] = 'i';    // 'i' -> Integer
                break;
```

```
            case 2:
printf("Enter number name: ");
                gets(num1[i].number);
typeArr[i] = 's';    // 's' -> String
                break;

            case 3:
printf("Enter floating-point number: ");
scanf("%f%*c", &num1[i].fnum);
typeArr[i] = 'f';    // 'f' -> Floating-point
                break;

            case 4:
printf("Enter double-type number: ");
scanf("%lf%*c", &num1[i].dnum);
typeArr[i] = 'd';    // 'd' -> Double
                break;

            default:
printf("Invalid option. Please enter a number between 0 - 4. \n");
                break;
        }
    }

    // Output - Retrieve numbers
printf("\n*** Printing Array of Numbers *** \n");

printf("{");
    for (int i = 0; i<numOfElts - 1; i++) {
        if (typeArr[i] == 'i') {
printf("%d, ", num1[i].num);
        }
        else if (typeArr[i] == 's') {
printf("\"%s\", ", num1[i].number);
        }
        else if (typeArr[i] == 'f') {
printf("%f, ", num1[i].fnum);
        }
        else if (typeArr[i] == 'd') {
printf("%lf, ", num1[i].dnum);
        }
    }

    if (typeArr[numOfElts - 1] == 'i') {
printf("%d} \n", num1[numOfElts - 1].num);
    }
    else if (typeArr[numOfElts - 1] == 's') {
printf("\"%s\"}\n", num1[numOfElts - 1].number);
    }
```

```
    else if (typeArr[numOfElts - 1] == 'f') {
printf("%f} \n", num1[numOfElts - 1].fnum);
    }
    else if (typeArr[numOfElts - 1] == 'd') {
printf("%lf} \n", num1[numOfElts - 1].dnum);
    }

printf("\nProgram ended. \n");

    return 0;
}
```

RESULT:

Concepts related to programs using structures and unions were studied.

INPUT / OUTPUT :

```
Enter number of elements : 5

*** Input Datatype Menu ***
1. Whole number (E.g. 3).
2. Number name (E.g. Three).
3. Floating-point (E.g. 3.14).
4. Double (Eg. 3.14159).

Enter option : 1
Enter whole number : 2

Enter option : 2
Enter number name : Five

Enter option : 3
Enter floating-point number : 4.7

Enter option : 4
Enter double-type number : 3.14159

Enter option : 2
Enter number name : Eight hundred

*** Printing Array of Numbers ***
{ 2, "Five", 4.700000, 3.141590, "Eight hundred" }

Program ended.
```

Ex. No. 12.	FILE HANDLING.
27/12/21	

AIM:

To learn file handling in C.

QUESTION 1:

Write a program to implement all the file handling functions.

ALGORITHM:

#1 Start.

#2 Create a file handle.

#3 Use 'fprintF()' to write class and section in file.

#4 Use 'putc()' to write a newline into the file.

#5 Get number of names as input from user.

#6 Input names into file using 'fprintF()'.

#7 Use 'fseek()' to move cursor to the first name in the list.

#8 Use 'fscanf()' to print the first name.

#9 Use 'rewind()' to seek the cursor to the beginning.

#10 Print class and section.

#11 Move cursor to the end of the file. Use 'ftell()' to print the position of the cursor.

#12 Use 'fclose()' to close the file.

#13 End.

QUESTION 2:

Write a program to print itself.

ALGORITHM:

#1 Start.

#2 Create a file handle.

#3 Set 'charCount' to 0.

#4 ~~Use 'rewind(~~ While character in text is not EOF

| increment charCount by 1.

#5 Use 'rewind()' to move cursor to starting of file.

#6 For character 'i' from 0 to 'charCount'

| Get character from file and print character.

#7 End.

Program 1:

```
/*
Program to implement all the file handling functions.
*/

#include <stdio.h>

int main() {
    FILE *fptr;
fptr = fopen("/Users/sheriffabdullah/College @ Shiv Nadar/OneDrive - SSN
Trust/CLAB/attendance.txt", "w+");

    // Class name
fprintf(fptr, "AI & DS - A");
fputc('\n', fptr);

    // Student names
    int numOfNames;
printf("Enter number of names: ");
scanf("%d%*c", &numOfNames);
    char name[30];

    for (int i = 0; i<numOfNames; i++) {
printf("Enter name [%d]: ", i + 1);
        gets(name);
fprintf(fptr, "%d. %s \n", i + 1, name);
    }

    char string[100];

    // Print first student's name
fseek(fptr, 12, SEEK_SET);    // Move cursor 'in front' by '20' characters
from 'beginning' (SEEK_SET)
fscanf(fptr, "%*d. %s", string);
printf("\nFirst student's name is %s. \n", string);

    // Print class name
printf("Class name: ");
    rewind(fptr);    // Bring cursor to beginning of file

    while (fgetc(fptr) != '\n') {
fseek(fptr, -1, SEEK_CUR);
printf("%c", fgetc(fptr));
    }
printf("\n");

    // Print position of cursor
fseek(fptr, 0, SEEK_END);    // Bring cursor to end of file
```

```
    long int curPos = ftell(fptr);
printf("Cursor position: %ld \n", curPos);

fclose(fptr);
printf("\nProgram ended. \n");

    return 0;
}

/*
File handling functions:

1. fopen("<file_name>.<extension", "<file_mode>")                          -
opens new or existing file.
2. fprintf(<file_ptr>, "<text/format_specifier>"[, <arg1>, <arg2>, ...]) -
write data into the file.
3. fscanf(<file_ptr>, "<text/format_specifier>"[, <arg1>, <arg2>, ...])  -
reads data from the file.
4. fputc(<char>, <file_ptr>)                                               -
writes a character into the file.
5. fgetc(<file_ptr>)                                                       -
reads & returns a character from file
6. fclose(<file_ptr>)                                                      -
closes the file.
7. fseek(<file_ptr>, <num_of_bytes>, SEEK_SET/SEEK_CUR/SEEK_END)           -
sets the file pointer to given position.
8. fputw()                                                                 -
writes an integer to file [Legacy].
9. fgetw()                                                                 -
reads an integer from file [Legacy].
10. ftell(<file_ptr>)                                                      -
returns current position.
11. rewind(<file_ptr>)                                                     -
sets the file pointer to the beginning of the file.

Legacy - used in older C compilers. Not in use anymore.
*/
```

INPUT / OUTPUT :

```
Enter number of names : 5
Enter name [1] : warning, this program uses gets(),
                 which is unsafe.

Abdullah
Enter name [2] : Abhiramee
Enter name [3] : Adithan
Enter name [4] : Manasa
Enter name [5] : Akhshan

First student's name is Abdullah.
Class name : AI & DS - [redacted]
Cursor position : 67

Program ended.
```

Program 2:

```
/*
Program that prints itself
*/

#include <stdio.h>

int main() {
    // File handle
    FILE *filePtr;
filePtr = fopen("/Users/sheriffabdullah/College @ Shiv Nadar/OneDrive - SSN
Trust/CLAB/exp113_programThatPrintsItselfUsingFileHandlingFunctions.c", "r");

    char character;
    int charCount = 0;

    // Number of characters in file
    while (getc(filePtr) != EOF) {
charCount ++;
    }

    rewind(filePtr);    // Move cursor to beginning

    // Print file
    for (int i = 0; i<charCount; i++) {
        character = fgetc(filePtr);
printf("%c", character);
    }

    return 0;
}

/*
File modes:
r  - Read                  [Cursor: Beginning]
r+ - Read & write          [Cursor: Beginning]
w  - Write                 [Cursor: Beginning]
w+ - Write & read          [Cursor: Beginning]
a  - Append = Write        [Cursor: End]
a+ - Append & read         [Cursor: End]

Note:
w, w+, a & a+ - If file does not exist -> Creates a new file
w, w+ - Truncates (erases) all existing data in file
*/
```

INPUT / OUTPUT :

```
/*
Program that prints itself
*/

#include <stdio.h>

int main () {
        // File handle
        FILE *filePtr;
        filePtr = Fopen("/Users/sheriffabdullah/College @ Shiv N-
                        adar/OneDrive - SSN Trust/CLAB/exp
                        113_program thatPrintsItselfUsing
                        FileHandling Functions.c", "r");

        char character;
        int charCount = 0;

        // Number of characters in file
        while (getc(filePtr) != EOF) {
                charCount++;
        }
        rewind(filePtr);   // Move cursor to beginning

        // Print file
        for (int i=0; i < charCount; i++) {
                character = fgetc(filePtr);
                printf("%c", character);
        }
```

RESULT :

Concepts related to file handling were studied.

```
    return 0;
}

/*
File modes:

r   - Read                  [Cursor: Beginning]
r+  - Read & write          [Cursor: Beginning]
w   - Write                 [Cursor: Beginning]
w+  - Write & read          [Cursor: Beginning]
a   - Append = Write        [Cursor: End]
a+  - Append & read         [Cursor: End]

Note:
w, w+, a & a+ - If file does not exist -> Creates a
                new file

w, w+ - Truncates (erases) all existing data in file
*/
```