MODULE 4: Structures and File Handling

1. What is structure data type? Explain.

Definition of Structures:

A structure is a collection of one or more variables of similar or dissimilar data types, grouped together under a single name i.e. A structure is a derived data type containing multiple variable of homogeneous or heterogeneous data types. The structure is defined using the keyword struct followed by an identifier. This identifier is called struct_tag.

Definition of Derived Data type: it's a data type created by the using primitive data types or fundamental data types.

Examples: arrays, structures, pointers, union, etc.

The syntax and example is as follows:

Syntax for Declaring and Defining Structure: the declaration of structure defines a template for structure and its members. No memory allocation is done for the declared structure.

```
struct struct_tag
{
        type  var_1;
        type  var_2;
        .
        .
        .
        type var_n;
};
```

Here,

- The word struct is a keyword in C
- The item identified as struct_tag is called a tag and identifies the structure later in the program
- Each type1, type2, …, typen can be any valid C data type including structure type
- The item var_1,var_2,…,var_n are the members or fields of the structure

Example:

```
struct  student
{
        char   name[20];
        int      roll_number;
        float   average_marks;
};
```

Declaring Structure Variables: declaring structure variables is nothing but creating instances of structures. Hence, actual memory allocation is done after declaring variables of structure type. Until and unless we create structure variables, it is not possible to store, process and access the members of the structure.

Total memory allocated for one variable of defined structure is sum of size of each member of the structure.

The syntax of declaring structure variables is shown below:

struct struct_tag v1,v2,v3,…,vn;

Example: struct student s1, s2, s3;

In the above statement, three variables s1, s2 and s3 are created of type struct student. Hence, separate memory allocation is done for each variable.

Therefore, total memory allocated for s1 = 20bytes (size of member **name**) + 4bytes (size of member **roll_number**) + 8bytes(size of **average_marks**) => 32bytes.

Similarly, for s2 another 32bytes of memory is allocated and for s3 also another 32bytes of memory is allocated.

The complete structure definition along with structure declaration is as shown below:

```
struct student
{
        char  name[10];
        int     roll_number;
        float   average_marks;
};
struct   student   cse, ise;          //creates   two   variables   of
```

structure type

Structure Initialization: Assigning the values to the structure member fields is known as structure initialization. The syntax of initializing structure variables is similar to that of arrays i.e all the elements are enclosed within braces i.e { and } and are separated by commas. The appropriate values for each member of the structure must be provided in sequence. Partial initialization is also allowed but programmer must not omit initializers for middle members of the structure.

The syntax is as shown below:

struct struct_tag variable = { v1,v2,v3,…,vn};

Example : struct student cse = { “Raju”,18, 87.5};

Accessing Members of Structures (Using The Dot (.) Operator):The member of structure is identified and accessed using the dot operator (.). The dot operator connects the member name to the name of its containing structure.

The general syntax is:

structure_variablename●membername;

Example: E1.name, here E1 is structure variable of structure employee and name is one of the member of structure employee.

Structure assignment (using assignment operator): the variable of one structure can be assigned to another variable of same structure. Every member value of one variable is copied to corresponding member of another variable.

Example:

struct student cse = { "Raju",18, 87.5}; // creates a variable cse and initialized with values as specified in curly braces.

struct student ise; //creates another variable ise of structure student type.

ise = cse;

in this statement, the value of name, roll_number and average_marks of cse are copied to name, roll_number and average_marks of ise respectively.

2. How structure is different from an array? Explain declaration of a structure with an example.

Both Structure and arrays are derived data types.

The major differences between structure and array are illustrated below:

Arrays	Structures
It is a collection of homogeneous elements i.e All elements of same data type	1. It is a collection of homogeneous or heterogeneous elements i.e Members of a structure can belong to different data types
For arrays, the default parameter passing mechanism is call-by-address/reference technique.	For structure, the default parameter passing mechanism is call-by-value technique.

Example : int a[10]; char x[10];	Example:
	struct complex
	{
	float real;
	int imaginary;
	};

Syntax for Declaring and Defining Structure: the declaration of structure defines a template for structure and its members. No memory allocation is done for the declared structure.

```
struct struct_tag
{
        type  var_1;
        type  var_2;
        .
        .
        .
        type var_n;
};
```

Here,

- The word struct is a keyword in C
- The item identified as struct_tag is called a tag and identifies the structure later in the program
- Each type1, type2, …, typen can be any valid C data type including structure type

- The item var_1,var_2,…,var_n are the members or fields of the structure

Example:

```
struct  student
{
        char   name[20];
        int      roll_number;
        float   average_marks;
};
```

Declaring Structure Variables: declaring structure variables is nothing but creating instances of structures. Hence, actual memory allocation is done after declaring variables of structure type. Until and unless we create structure variables, it is not possible to store, process and access the members of the structure.

Total memory allocated for one variable of defined structure is sum of size of each member of the structure.

The syntax of declaring structure variables is shown below:

struct struct_tag v1,v2,v3,…,vn;

Example: struct student s1, s2, s3;

In the above statement, three variables s1, s2 and s3 are created of type struct student. Hence, separate memory allocation is done for each variable.

Therefore, total memory allocated for s1 = 20bytes (size of member **name**) + 4bytes (size of member **roll_number**) + 8bytes(size of **average_marks**) => 32bytes.

Similarly, for s2 another 32bytes of memory is allocated and for s3 also another 32bytes of memory is allocated.

The complete structure definition along with structure declaration is as shown below:

```
struct student
{
        char  name[10];
        int     roll_number;
        float   average_marks;
};
struct    student    cse, ise;
```

3. What is a structure? Explain the C syntax of structure declaration and initialization with an example.

Definition of Structures:

A structure is a collection of one or more variables of similar or dissimilar data types, grouped together under a single name i.e. A structure is a derived data type containing multiple variable of homogeneous or heterogeneous data types. The structure is defined using the keyword struct followed by an identifier. This identifier is called struct_tag.

The syntax and example is as follows:

Syntax for Declaring and Defining Structure: the declaration of structure defines a template for structure and its members. No memory allocation is done for the declared structure.

```
struct struct_tag
{
        type  var_1;
        type  var_2;
        .
        .
        .
        type var_n;
};
```

Here,

- The word struct is a keyword in C
- The item identified as struct_tag is called a tag and identifies the structure later in the program
- Each type1, type2, …, typen can be any valid C data type including structure type
- The item var_1,var_2,…,var_n are the members or fields of the structure

Example:

```
struct  student
{
        char   name[20];
        int      roll_number;
        float   average_marks;
};
```

Declaring Structure Variables: declaring structure variables is nothing but creating instances of structures. Hence, actual memory allocation is done after declaring variables of structure type. Until and unless we create structure variables, it is not possible to store, process and access the members of the structure.

Total memory allocated for one variable of defined structure is sum of size of each member of the structure.

The syntax of declaring structure variables is shown below:

```
struct  struct_tag v1,v2,v3,…,vn;
```

Example: `struct student s1, s2, s3;`

In the above statement, three variables s1, s2 and s3 are created of type struct student. Hence, separate memory allocation is done for each variable.

Therefore, total memory allocated for s1 = 20bytes (size of member **name**) + 4bytes (size of member **roll_number**) + 8bytes(size of **average_marks**) => 32bytes.

Similarly, for s2 another 32bytes of memory is allocated and for s3 also another 32bytes of memory is allocated.

Structure Initialization: Assigning the values to the structure member fields is known as structure initialization. The syntax of initializing structure variables is

similar to that of arrays i.e all the elements are enclosed within braces i.e { and } and are separated by commas. The appropriate values for each member of the structure must be provided in sequence. Partial initialization is also allowed but programmer must not omit initializers for middle members of the structure.

The syntax is as shown below:

```
struct  struct_tag  variable  = { v1,v2,v3,…,vn};
```

Example : `struct student cse = { "Raju",18, 87.5};`

4. What are typedefinitions? What are the advantages of typedef? Explain with an example, how to create a structure using 'typedef'.

Type definition: The typedef is a keyword using which the programmer can create a new data type name for an already existing data type name. So the purpose of typedef is to redefine or rename the name of an existing data type.

Syntax:

```
typedef  data_type  newname1, newname2,…, newnamen;
```

Example1: `typedef int NUMBER;`

Example2: `typedef float SALARY;`

The new names that are given by the user for the already existing data types are called user defined data types. In the above example NUMBER and SALARY are user defined data types.

The user defined data types can be used to declare variables as illustrated below:

Example1:
```
typedef int NUMBER;
NUMBER N1,N2,N3;
```

Example2:
```
typedef float SALARY;
SALARY s1,s2,s3,s4,s5;
```

Advantages of typedef

a. User-defined names can be defined for existing data types

Example:
```
typedef     int     integer;
typedef     char    character;
integer     n1, n2;
character   c1;
```

From the above example, we can understand that, beginners of users of C can understand the statement *integer n1, n2* as n1 and n2 are variables of integer type and similarly, c1 is of character type.

b. Based on context or application, new names can be defined for data types.

Example1: if we are developing program to represent various whole numbers then int can be renamed as below

```
typedef     int     WHOLE_NUMBER;
WHOLE_NUMBER        w1, w2;
```

From the above two lines, we can understand that w1 and w2 will be used to represent whole numbers as the name of the data type indicates its purpose.

Example2: if we are developing program to represent percentage of three semesters of student then float can be renamed as below

```
typedef     float   percentage;
percentage  sem_1, sem_2, sem_3;
```

From the above two lines, it can be understood that sem_1, sem_2 and sem_3 represents the percentage of three semesters.

Hence, based on kind of application we are building, we can rename the existing data types as required which will be more beneficial and easy to understand.

c. Long names of existing data types can be shortened

Example: typedef unsigned long int ULINT;

In the above example, the data type name unsigned long int is renamed as ULINT it is shorter than original. So instead of using long name of the data type shortened name can be used which reduce time required to type from the keyboard.

Creating a Structure using typedef:

The structure defined using the keyword typedef is called type defined structure.

There are two different ways to create structures using typedef

1) Renaming structure type after declaration of structure using typedef.

Example:

```
struct student
{
    char name[20];
    char usn[11];
    float marks;
};
```

Once, the structure has been declared, the new data type i.e. struct student can be renamed using typedef as given below.

```
typedef struct student STUDENT;
```

Now onwards, the new name for struct student type is STUDENT. We can create variable of this structure type by using the name STUDENT.

For example: STUDENT s; // s is variable of type struct student

2) Renaming structure type at the time of declaring structure using typedef.

The syntax of the typedefined structure along with example is as given below:

Syntax:

```
typdef  struct  [tagname]
{
    data_type1 member1;
    data_type2  member2;
    .
    .
    .
    data_typen  membern;
} struct_ID ;
```

Example:

```
typedef      struct  student  ------>  Optional
{
        char name[20];
        char usn[11];
        float marks;

}STUDENT;
```

NOTE: as shown with diagram and arrow mark, the tag name i.e. student which is used to name the structure is optional.

In this example, the new data type struct student has been declared as well it is been renamed as STUDENT.

Programming Example: Program to simulate the addition of two complex numbers using the typedef structures is illustrated below:

```
#include<stdio.h>
typdef  struct
{
        int  real_part;
        int  imaginary_part;
}complex;
int main ()
{
        complex   c1,c2, sc;
        printf("\n Enter real and imaginary part of first complex number);
        scanf("%d %d",&c1.real_part,&c1.imaginary_part);
        printf("\n Enter real and imaginary part of second complex number);
        scanf("%d %d",&c2.real_part,&c2.imaginary_part);
        sc.real_part = c1.real_part + c2.real_part;
        sc.imaginary_part = c1.imaginary_part +c2.imaginary_part;
        printf("The real and imaginary part of resultant complex number is \n");
        printf("%d   %d  \n", sc.real_part, sc.imaginary_part);
        return 0;
}
```

5. Explain structure within a structure (Nested Structures) with examples.

A structure can be a member of another structure. A structure which includes another structure is called as nested structure. There is no limit for number of structures that can be nested.

Example:

```
struct dob
{
        int dd;
```

```
        int mm;
        int yyyy;
};
struct student_detail
{
        int id;
        char name[20];
        float percentage;
        // structure within structure
        struct dob DOB;
}stu_data;
```

In this example, "dob' structure is declared inside "student_detail" structure. Both structure variables are normal structure variables.

To access members of the main structure and nested structure we have to use the following syntax.

To access dd of nested structure: stu_data**.**DOB.dd

To access mm of nested structure: stu_data**.**DOB.mm

To access yyyy of nested structure: stu_data**.**DOB.yyyy

This program explains how to use structure within structure in C. "student_college_detail' structure is declared inside "student_detail" structure in this program. Both structure variables are normal structure variables.

Please note that members of "student_college_detail" structure are accessed by 2 dot(.) operator and members of "student_detail" structure are accessed by single dot(.) operator.

```
#include <stdio.h>
#include <string.h>
struct   student_college_detail
{
        int college_id;
        char college_name[50];
};
struct   student_detail
{
        int id;
        char name[20];
        float percentage;
        // structure within structure
        struct   student_college_detail          clg_data;
};
int main()
{
        struct student_detail stu_data = {1, "xyz", 90.5, 32, "REC Hulkoti"};
        printf(" Id is: %d \n", stu_data.id);
```

```
        printf(" Name is: %s \n", stu_data.name);
        printf(" Percentage is: %f \n\n", stu_data.percentage);
        printf(" College Id is: %d \n", stu_data.clg_data.college_id);
        printf("College Name is: %s \n", stu_data.clg_data.college_name);
        return 0;
}
```

6. Explain the concept of array of structures, with suitable C program.

Solution:

In many situations, we need to create an array of structures. To name one example, we would use an array of students to work with a group of students stored in a structure. By putting the data in an array, we can quickly and easily work with the data.

Like arrays, Array of Structure can be initialized at compile time.

Way1 : **Initializing After Declaring Structure Array**:

```
struct Book
{
    char bname[20];
    int pages;
    char author[20];
    float price;
}b1[3] = { {"Let us C", 700, "YPK", 300.00}, {"Wings of Fire", 500, "APJ Abdul Kalam", 350.00}, {"Complete C", 1200, "Herbert Schildt", 450.00}};
```

Explanation: As soon as after declaration of structure we initialize structure with the pre-defined values. For each structure variable we specify set of values in curly braces. Suppose we have 3 Array Elements then we have to initialize each array element individually and all individual sets are combined to form single set.

{"Let us C",700,"YPK",300.00}

Above set of values are used to initialize first element of the array.

Similarly - {"Wings of Fire",500,"APJ Abdul Kalam",350.00} is used to initialize second element of the array.

Way 2: **Initializing in Main**

```
struct Book
{
    char bname[20];
    int pages;
    char author[20];
    float price;
};
int main()
{
    struct Book b1[3] = { {"Let us C",700,"YPK",300.00},
                          {"Wings of Fire",500,"Abdul Kalam",350.00},
                          {"Complete C",1200,"Herbt Schildt",450.00}
```

```
                                    };
}
```

Some Observations and Important Points :

Tip #1 : All Structure Members need not be initialized

```
#include<stdio.h>
struct Book
{
    char bname[20];
    int pages;
    char author[20];
    float price;
}b1[3] = {{"Book1",700,"YPK"}, {"Book2",500,"AAK",350.00},
{"Book3",120,"HST",450.00}};
int main()
{
    printf("\nBook Name    : %s",b1[0].bname);
    printf("\nBook Pages: %d",b1[0].pages);
    printf("\nBook Author: %s",b1[0].author);
    printf("\nBook Price   : %f",b1[0].price);
}
```

Output :

```
Book Name    : Book1
Book Pages   : 700
Book Author  : YPK
Book Price   : 0.000000
```

Explanation: In this example, While initializing first element of the array we have not specified the price of book 1. It is not mandatory to provide initialization for all the values. Suppose we have 5 structure elements and we provide initial values for first two element then we cannot provide initial values to remaining elements.

```
{"Book1", 700, ,90.00}
```

above initialization is illegal and can cause compile time error.

Tip #2 : Default Initial Value

```
struct Book
{
    char bname[20];
    int pages;
    char author[20];
    float price;
}b1[3] = { {},{"Book2",500,"AAK",350.00}, {"Book3",120,"HST",450.00} };
```

Output :

```
Book Name    :  Book
Pages   : 0
Book Author  :
Book Price   : 0.000000
```

It is clear from above output. Default values for different data types.

C program: to illustrate array of structures.

```
#include<stdio.h>
typedef struct
{
        char  name[20];          // Name
        char usn[11];                   //usn
        char     subname[30];           //subject name
        int  m1, m2, m3;                //marks
}student;
int main()
{
        student s[100];
        int       i, N;
        printf("Enter N value\n");
        scanf("%d", &N);
        //Reading student details
        printf("Enter %d student details\n", N);
        for(i=0;i<N;i++)
        {
                printf("Enter details of %d student\n" , i+1);
                printf("\nEnter name of a student :");
                scanf("%s" , s[i].name);
                printf("\n Enter USN:");
                scanf("%s" , s[i].usn);
                printf("\n Enter subject name:");
                scanf("%s" , s[i].subname);
                printf("\n Enter first, second and third IA marks :");
                scanf(" %d %d %d" , &s[i].m1, &s[i].m2, &s[i].m3);
        }
        // Printing student details
        printf("\n Name \t USN \t Subject Name \t Marks1 \t Marks2 \tMarks3\n");
        for(i=0;i<N;i++)
                printf("%s \t %s \t %s \t %d \t %d \t %d \n" , s[i].name, s[i].usn,
    s[i].subname, s[i].m1, s[i].m2, s[i].m3);
        return 0;
    }
```

Explanation: in this program a structure which include name, usn, subject name and IA marks of student is declared then an array of students is declared by using student s[100] statement which defines array s with capable of storing 100 students information. Each student information is accessed by subscripting name of the array s with an index i as s[i]. In this example, N student details are inputted from the keyboard and same details are displayed on screen.

7. Show how a structure variable is passed as a parameter to a function, with an example.

Solution:

In C, structure can be passed to functions by two methods:

1. Pass by value (passing actual value of structure variable as argument)
2. Pass by reference (passing address of structure variable as argument)

Passing structure using call-by-value: A structure variable can be passed to the function as an argument as normal variable. If structure is passed by value, change made in structure variable in function definition does not reflect in original structure variable in calling function.

Write a C program to create a structure student, containing name and roll number. Ask user the name and roll number of a student in main function. Pass this structure to a function and display the information in that function.

```
#include <stdio.h>
struct student
{
    char name[50];
    int rollno;
};
void display(struct student stu);  /* function prototype should be below to the structure declaration otherwise compiler shows error */
int main()
{
    struct student s1;
    printf("Enter student's name: ");
    scanf("%s" , s1.name);
    printf("Enter roll number:");
    scanf("%d" , &s1.rollno);
    display(s1);     // passing structure variable s1 as argument
    return 0;
}
void display(struct student stu)
{
    printf("Entered Name: %s" , stu.name);
    printf("\nRoll Number: %d" , stu.rollno);
}
```

Output

```
Enter student's name: xyz
Enter roll number: 14
Entered Name: xyz
Roll Number: 14
```

Passing structure using call-by-reference: The address location of structure variable is passed to a function using pass by reference technique. If structure is

passed by reference, change made in structure variable in function definition reflects in original structure variable in the calling function.

Write a C program to add two distances (feet-inch system) entered by user.

To solve this program, make a structure. Pass two structure variable (containing distance in feet and inch) to add function by pass-by-reference and display the result in main function without returning it.

```
#include <stdio.h>
struct distance
{
    int feet;
    float inch;
};
void add(struct distance d1,struct distance d2, struct distance *d3);
int main()
{
    struct distance dist1, dist2, dist3;
    printf("First distance\n");
    printf("Enter feet: ");
    scanf("%d" , &dist1.feet);
    printf("Enter inch: ");
    scanf("%f",&dist1.inch);
    printf("Second distance\n");
    printf("Enter feet: ");
    scanf("%d",&dist2.feet);
    printf("Enter inch: ");
    scanf("%f",&dist2.inch);
    add(dist1, dist2, &dist3);    /*passing structure variables dist1 and dist2 by value whereas passing structure variable dist3 by reference */
    printf("\nSum of distances = %d \t %f\",dist3.feet, dist3.inch);
    return 0;
}
void add(struct distance d1,struct distance d2, struct distance *d3)
{
    /* Adding distances d1 and d2 and storing it in d3 */
    d3->feet = d1.feet + d2.feet;
    d3->inch = d1.inch + d2.inch;
    if (d3->inch>=12)
    {
            d3->inch -= 12;
            ++d3->feet;
    }
}
```

Output

First distance

Enter feet: 12
Enter inch: 6.8
Second distance
Enter feet: 5
Enter inch: 7.5
Sum of distances = 18 - 2.3

Explaination: In this program, structure variables dist1 and dist2 are passed by value (because value of dist1 and dist2 does not need to be displayed in main function) and dist3 is passed by reference, i.e, address of dist3 (&dist3) is passed as an argument. Thus, the structure pointer variable d3 points to the address of dist3. If any change is made in d3 variable, effect of it is seen in dist3 variable in main function.

8. Write a C program to store Name, USN, subject name and IA Marks of a student using structure. Print the same.

Program:

```
#include<stdio.h>
struct student
{
        char  name[20];         // Name
        char usn[11];           //usn
        char    subname[30];            //subject name
        float  m1,m2,m3;                //marks
};
int main()
{
        struct   student s;
        //Reading student details
        printf("\nEnter name of a student :");
        scanf("%s" , s.name);
        printf("\n Enter USN:");
        scanf("%s" , s.usn);
        printf("\n Enter subject name:");
        scanf("%s" , s.subname);
        printf("\n Enter first, second and third IA marks :");
        scanf(" %f %f %f" , &s.m1, &s.m2, &s.m3);
        // Printing student details
        printf("\n Name \t USN \t Subject Name \t Marks1 \t Marks2 \tMarks3\n");
        printf("%s \t %s \t %s \t %f \t %f \t %f \n" , s.name, s.usn, s.subname, s.m1,
    s.m2, s.m3);
        return 0;
}
```

9. Write a C program to store the following details of 'N' students using structure: Name: string, USN: integer, subject name: string and IA Marks: integer. Output the same details.

```
#include<stdio.h>
struct student
{
        char  name[20];         // Name
        int usn;                //usn
        char    subname[30];            //subject name
        int  m1, m2, m3;                //marks
};
int main()
{
        struct   student s[100];
        int      i, N;
        printf("Enter N value\n");
        scanf("%d", &N);
        //Reading student details
        printf("Enter %d student details\n", N);
        for(i=0;i<N;i++)
        {
                printf("Enter details of %d student\n" , i+1);
                printf("\nEnter name of a student :");
                scanf("%s" , s[i].name);
                printf("\n Enter USN:");
                scanf("%d" , &s[i].usn);
                printf("\n Enter subject name:");
                scanf("%s" , s[i].subname);
                printf("\n Enter first, second and third IA marks :");
                scanf(" %d %d %d" , &s[i].m1, &s[i].m2, &s[i].m3);
        }
        // Printing student details
        printf("\n Name \t USN \t Subject Name \t Marks1 \t Marks2 \tMarks3\n");
        for(i=0;i<N;i++)
                printf("%s \t %d \t %s \t %d \t %d \t %d \n" , s[i].name, s[i].usn,
    s[i].subname, s[i].m1, s[i].m2, s[i].m3);
        return 0;
    }
```

10. Write a C program to input the following details of 'N' students using structure:
Roll No: integer, Name: string, Marks: float, Grade: char
Print the names of the students with marks>=70.0%.

Program:

```
#include<stdio.h>
struct student
{
        int     rno;            // Roll No
        char  name[20];         // Name
        float    marks;         //Marks
        char  grade;            //Grade
};
int main()
{
        int      i, n, found=0;
        struct   student s[20];
        printf("\nHow many student details ?);
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\n Type in %d student detail \n",i+1);
                printf("\n Roll no :");
                scanf("%d" , &s[i].rno);
                printf("\n Name:");
                scanf("%s" , s[i].name);
                printf("\n Marks:");
                scanf("%f" , &s[i].marks);
                printf("\n Grade :");
                scanf(" %c" , &s[i].grade);
        }
        printf("\n The Names of the students with marks >=70.0%\n");
        for(i=0;i<n;i++)
        {
                if(s[i].marks>=70.0))
                {
                        printf("\n %s\n" , s[i].name);
                        found =1;
                }
        }
        if(found ==0)
                printf("\n There is no student with marks >=70.0%\n");
        return 0;
}
```

11. Write a C program to maintain a record of "n" student details using an array of structures with four fields (Roll number, Name, Marks, and Grade). Each field is of an appropriate data type. Print the marks of the student given student name as input.

Program:

```
#include<stdio.h>
#include<string.h>
struct student
{
        int     rno;            // Roll No
        char  name[20];         // Name
        float    marks;         //Marks
        char  grade;            //Grade
};
int main()
{
        int      i, N;
        struct   student s[20];
        char key[20];
        printf("\nHow many student details ?);
        scanf("%d",&N);
        for(i=0;i<N;i++)
        {
                printf("\n Type in %d student detail \n",i+1);
                printf("\n Roll no :");
                scanf("%d" , &s[i].rno);
                printf("\n Name:");
                scanf("%s" , s[i].name);
                printf("\n Marks:");
                scanf("%f" , &s[i].marks);
                printf("\n Grade :");
                scanf("%c" , &s[i].grade);
        }
        printf("Enter the name to be searched\n");
        scanf("%s", key);
        for(i=0; i<N; i++)
        {
                if(strcmp(key, s[i].name))
                {
                        printf("\n Marks obtained by %s = %d\n" , key, s[i].marks);
                        return 0;
                }
        }
        printf("\n No student found with name = %s \n", key);
```

```
    return 0;
}
```

File Handling

12. What is a file? What are the different modes of opening a file? Explain.

Solution: Abstractly, a file is a collection of data stored on a secondary storage device, which is generally a disk. The collection of data may be interpreted, for example, as characters, words, lines, paragraphs and pages from a textual document; fields and records belonging to a database; or pixels from a graphical image.

Files can be classified as input files and output files. Input files are file which contains various data that will be provided as input to the program. Output files are files which contains the output produced from the program.

Working with file: While working with file, you need to declare a pointer of type FILE. This declaration is needed for communication between file and program.

```
FILE *ptr;
```

Here, ptr is pointer to file called as file pointer which will be used to address or reference the file.

Different Modes of file opening in Standard I/O

In C, the files are opened using fopen function. Which accepts two parameters; first parameter is name of the file to be opened and second parameter is mode of file open.

General Syntax of fopen call:

```
FILE *fopen(char filename[], char mode[]);
```

On successful opening of specified file, fopen function returns a file pointer. On failure, fopen function returns NULL.

File Access Mode	Meaning of Mode	If file doesn't exist
"r"	Open for reading	If the file does not exist, fopen() returns NULL.
"w"	Open for writing	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
"a"	Open for appending i.e data is added from end of file	If the file doesn't exist it will be created
"r+"	Open for both reading and writing	If the file does not exist, fopen() returns NULL.
"w+"	Open for both reading and writing	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
"a+"	Open for both reading and appending	If the file does not exists, it will be created.

13. Explain how the input is accepted from a file and displayed.

Solution:

Accepting input from a file and displaying can be done in different ways.

a. Using fscanf() and fprint()

fscanf function is used to accept input from a file. Which accepts three parameters; first parameter is source of input which can be either file or standard input device, second parameter defines the type of data to read and third parameter defines address of variable where the data is to be stored. Once the data is stored in variables then it can be displayed on standard output device i.e., monitor or can be stored in a file using fprintf().

This program reads a string of text from a file using fscanf() and fprintf().

/* Source Code to read a text of string and display the same from a file. */

```
#include <stdio.h>
#include<stdlib.h>          // for exit function
int main()
{
    char data[1000];
    FILE *fptr;
    if ((fptr=fopen("program.txt","r"))==NULL)
    {
            printf("Error! opening file");
            exit(1);        /* Program exits if file pointer returns NULL. */
    }
    fscanf(fptr,"%[^\n]",data);     //read every character that is not '\n'
    fprintf(stdout,"Data from file:\n%s",data);
    fclose(fptr);
    return 0;
}
```

This program reads the content of "program.txt" file and stores it in a string data. Then content of this data is displayed on screen using fprintf(). If there is no file named program.txt then, this program displays Error! opening file and program will be terminated.

b. Using fgetc and fputc function

fgetc() function is used to read one character at a time from a specified file. This procedure can be repeated until end of file.

fputc() function is used to write one character at a time into a specified file. This procedure can be repeated until all the characters are written to a file.

/* program to read text from file and store it in another file using fgetc and fputc */

```
#include <stdio.h>
#include<stdlib.h>          // for exit function
int main()
{
    char ch;
```

```
FILE *fptr1,*fptr2;
if ((fptr1=fopen("read.txt","r"))==NULL)
{
        printf("Error! opening file");
        exit(1);        /* Program exits if file pointer returns NULL. */
}
if ((fptr2=fopen("write.txt","w"))==NULL)
{
        printf("Error! opening file");
        exit(1);        /* Program exits if file pointer returns NULL. */
}
while((ch = fgetc(fptr1)) != EOF)      // Read a Character until end of file
        fputc(ch,fptr2);
fclose(fptr1);
fclose(fptr2);
return 0;
}
```

c. Using fgets and fputs functions

These are useful for reading and writing entire lines of data to/from a file. If *buffer* is a pointer to a character array and *n* is the maximum number of characters to be stored, then

fgets (buffer, n, input_file);

will read an entire line of text (max chars = n) into *buffer* until the newline character or n=max, whichever occurs first. The function places a NULL character after the last character in the buffer. The fgets function returns NULL when no more data to read.

fputs (buffer, output_file);

writes the characters in *buffer* until a NULL is found. The NULL character is not written to the *output_file*.

NOTE: fgets does not store the newline into the buffer, fputs will append a newline to the line written to the output file.

/* program to read text from file and store it in another file using fgets and fputs */

```
#include <stdio.h>
#include<stdlib.h>          // for exit function
int main()
{
    char data[101];
    FILE *fptr1,*fptr2;
    if ((fptr1=fopen("read.txt","r"))==NULL)
    {
            printf("Error! opening file");
            exit(1);        /* Program exits if file pointer returns NULL. */
    }
    if ((fptr2=fopen("write.txt","w"))==NULL)
```

```
{
        printf("Error! opening file");
        exit(1);        /* Program exits if file pointer returns NULL. */
}
/* Read 100 Characters or until end of file whichever occurs first from read.txt
file and write it to write.txt file.   */
while((fgets(data,100,fptr1)) != NULL)
        fputs(data,fptr2);
fclose(fptr1);
fclose(fptr2);
return 0;
}
```

14. Explain the following file operations along with syntax and examples:

a) fopen() b) fclose() c) fscanf() d) fprintf() e) feof()

Solution:

a) **fopen()** : Opening a file is done by a call to the function fopen( ) which tells the operating system the name of the file and whether the file is to be opened for reading or for writing or for appending.

General form of call to fopen:

FILE *fopen(char filename[], const char mode[]);

OR

filepointer = fopen ("Filename" , "mode");

The function fopen takes two parameters both of which are strings.

The parameter **filename** is a C string which must contain the name of the disk file to be opened. If the file is not in the default directory a full path name must be provided.

Parameter **mode** is a C string which defines the way the file is to be opened. The mode is a string not characters and must be enclosed in double quotation marks.

The various modes are shown below.

Access Mode	Description
"r"	Open a file for **reading**. Specified file must exist.
"w"	Creates an empty file for **writing**. If the specified file doesn't exist then, it creates a new file by specified name and opens for writing. If the file already exist then, its content is erased and opens as new empty file for writing.
"a"	Open a file for **appending**. If the file doesn't exist then, it creates new file by specified name and opens it for appending.
"r+"	Open a file for both reading and writing. Specified file must exist.
"w+"	Creates an empty file for both reading and writing.
"a+"	Opens a file for both reading and appending.

Example1: **Creating and opening three different files in different modes**

```
FILE    *fp1,*fp2,*fp3;
```

```
fp1  = fopen("student.txt","r");        // opens a file for reading
fp2  =fopen("One.txt","w");             // opens a file for writing
fp3  = fopen("newfile.txt","a");        // opens a file for appending
```

Note:

1. One must include the complete path of the file if the file is not stored in the default directory.
   Example:
   ```
   fp1 = fopen("C:\\foldername\\xyz.txt" , "r");
   ```
2. The data file may be named with extension .dat or .txt

ii) **fclose()** : After file has been used it must be closed. This is done by a call to the function fclose(). The fclose() function breaks the connection between the stream and the file.

General form of a call to the fclose() is

```
fclose(file_pointer);
```

The function fclose takes one parameter which is pointer to a file of opened file. If there is an error, such as trying to close a file that is not opened, the function returns EOF; otherwise it returns 0. The return value is usually not checked.

Example :

```
File  *fp1,*fp2 ;
fclose(fp1) ;          // closes a file referenced by fp1
fclose(fp2);           // closes a file referenced by fp2
```

Opening and closing the File : Example program

```
int main()
{
    FILE *fp;
    char ch;
    fp = fopen("INPUT.txt","r");          // Open file in Read mode
    while(1)
    {
        ch = fgetc(fp);          // Read a Character
        if(ch == EOF )           // Check for End of File
            break ;
        printf("%c",ch);
    }
    fclose(fp);                  // Close File after Reading
}
```

iii) **fscanf()**

The C library function fscanf() reads formatted input from a file or from the standard input device i.e. keyboard.

General syntax of fscanf() is

```
fscanf(filepointer, "format string", list of address of variables);
```

fscanf() returns number of data items successfully read.

Here,

- **filepointer** − This is the pointer to a FILE object that identifies the opened file or it can be **stdin** if we want to read data from keyboard.
- **format string** − This is the C string that contains one or more format specifiers which defines type of data to read from the file or from the keyboard.
- **List of address of variables –** this contains one or more address of variables where the data will be stored.

```
// Example program which reads the data from file using fscanf()
#include <stdio.h>
#include <stdlib.h>
int main()
{
  char str1[10], str2[10], str3[10];
  int year;
  FILE * fp;
  fp = fopen ("file.txt", "w+");
  fputs("We are in 2012", fp);
  rewind(fp);
  fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);     // reads three strings and one integer data
  printf("Read String1 |%s|\n", str1 );
  printf("Read String2 |%s|\n", str2 );
  printf("Read String3 |%s|\n", str3 );
  printf("Read Integer |%d|\n", year );
  fclose(fp);
  return(0);
}
```

iv) fprintf()

The C library function fprintf() prints formatted output to a file, printer or to the standard output device i.e. screen.

General syntax of fprintf() is

fprintf(filepointer, “format string”, list of variables);

fprintf() returns number of data items successfully written.

Here,

- **filepointer** − This is the pointer to a FILE object that identifies the opened file or it can be **stdout** if we want to write data to screen.
- **format string** − This is the C string that contains one or more format specifiers which defines type of data to be written to the file or standard output device.

- **List of variables –** this contains one or more variables of which the data will be written.

```
/* example program to write data to a file */
#include <stdio.h>
#include <stdlib.h>
int main()
{
  FILE * fp;
  fp = fopen ("file.txt", "w+");
  fprintf(fp, "%s %s %s %d", "We", "are", "in", 2012); // writes three strings followed by an integer separated by whitespace to the file file.txt
  fclose(fp);
  return(0);
}
```

v) **feof()**

The C library function **int feof(FILE *filepointer)** tests the end-of-file indicator for the given file. Where, filepointer is the pointer to file object which identifies the file to be tested.

This function returns a non-zero value when End-of-File is encountered, else zero is returned.

```
// use of feof() in program
#include <stdio.h>
int main ()
{
  FILE *fp;
  int c;
  if((fp = fopen("file.txt","r"))==NULL)
  {
    printf("Error in opening file");
    return (-1);
  }
  while(!feof(fp))  //feof() tests the End-of-File, if EOF then jumps out of loop.
  {
       c = fgetc(fp);
       printf("%c", c);
  }
  fclose(fp);
  return(0);
}
```

15. Explain the following file operations along with syntax and examples:
a) fputc( ) b) fputs( ) c) fgetc( ) d) fgets()

a) fputc()

The C library function **int fputc(int character, FILE *filepointer)** writes a character (an unsigned char) specified by the argument **char** to the specified file and advances the position indicator for the file.

Here,

- **character** - This is the character to be written. This is passed as its int promotion.
- **filepointer** -- This is the pointer to a FILE object that identifies the file where the character is to be written.

If there are no errors, the same character that has been written is returned. If an error occurs, EOF is returned and the error indicator is set.

// example program to illustrate use of fputc()

```
#include <stdio.h>
int main ()
{
  FILE *fp;
  int ch;
  fp = fopen("file.txt", "w+");
  for( ch = 'A' ; ch <= 'Z'; ch++ )
    fputc(ch, fp);        // writes a character stored in ch to file pointed by fp
  fclose(fp);
  return(0);
}
```

b) fputs()

The C library function **int fputs(const char *str, FILE *filepointer)** writes a string to the specified file up to but not including the null character.

Here,

- **str** - This is an array containing the null-terminated sequence of characters to be written.
- **filepointer** - This is the pointer to a FILE object that identifies the file where the string is to be written.

This function returns a non-negative value on success or EOF on error.

```
// program to illustrate use of fputs()
#include <stdio.h>
int main ()
{
  FILE *fp;
  fp = fopen("file.txt", "w+");
//writes this is c programming to file pointed by fp
  fputs("This is c programming.", fp);
//writes This is a system programming language. to the file pointed by fp.
  fputs("This is a system programming language.", fp);
  fclose(fp);
```

```
  return 0;
}
```

c) fgetc()

The C library function **int fgetc(FILE *filepointer)** gets the next character (an unsigned char) from the specified file and advances the position indicator for the file.

Where,

- **filepointer** - This is the pointer to a FILE object that identifies the file on which the operation is to be performed.

This function returns the character read as an unsigned char cast to an int or EOF on end of file or error.

```
// program to illustrate use of fputc()
#include <stdio.h>
int main ()
{
  FILE *fp;
  int c;
  int n = 0;
  fp = fopen("file.txt","r");
  if(fp == NULL)
  {
    printf("Error in opening file");
    return (-1);
  }
  while(!feof(fp))       //read until end of file
  {
        c = fgetc(fp);   // reads an unsigned character and converts to integer.
        printf("%c", c);
  }
  fclose(fp);
  return(0);
}
```

d) fgets()

The C library function **char *fgets(char *str, int n, FILE *filepointer)** reads a line from the specified stream and stores it into the string pointed to by **str**.

It stops when either **n-1** characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.

Where,

- **str** -- This is the pointer to an array of characters where the string read is stored.

- **n** -- This is the maximum number of characters to be read (including the final null-character). Usually, the length of the array is passed.
- **filepointer** -- This is the pointer to a FILE object that identifies the file where characters are read from.

On success, the function returns the same str parameter. If the End-of-File is encountered and no characters have been read, the contents of str remain unchanged and a null pointer is returned.

If an error occurs, a null pointer is returned.

```
// example program to illustrate use of fgets()
#include <stdio.h>
int main()
{
  FILE *fp;
  char str[60];
  /* opening file for reading */
  fp = fopen("file.txt" , "r");
  if(fp == NULL)
  {
    printf("Error opening file");
    return (-1);
  }
// fgets() reads maximum of 60 characters from file pointed by fp and
stores in str
  if( fgets (str, 60, fp)!=NULL )
  {
    /* writing content to stdout */
    puts(str);
  }
  fclose(fp);
  return(0);
}
```

16. Write a C program to read and display a text from the file.

```
/* Source Code to read a text of string and display the same from a file. */
#include <stdio.h>
#include<stdlib.h>          // for exit function
int main()
{
    char ch;
    FILE *fptr;
    if ((fptr=fopen("program.txt","r"))==NULL)
    {
            printf("Error! opening file");
```

```
        exit(1);        /* Program exits if file pointer returns NULL. */
    }
    printf("Content of file is:\n");
    while((ch=fgetc(fptr))!=EOF)
        fputc(ch, stdout);
    fclose(fptr);
    return 0;
}
```

This program reads the content of "program.txt" file one character at a time using fgetc function and stores it in a variable ch. Then content of this variable ch is displayed on screen using fputc(). If there is no file named program.txt then, this program displays Error! opening file and program will be terminated.

16. **Write a C program to read the contents from the file called 'abc.txt', count the number of characters, number of lines and number of whitespaces and output the same.**

Program:

```
#include<stdio.h>
int main()
{
    FILE    *fp;
    char   ch;
    int   cc =0;      /* number of characters */
    int   bc=0;       /* number of blanks */
    int   tc =0;      /* number of tabs*/
    int   lc=0;       /* number of lines */
    int  wc=0;        /* number of words */
    fp = fopen("abc.txt" , "r");
    if(fp==NULL)
    {
        printf("Error in opening the file \n");
        return -1;
    }
    while((ch=fgetc(fp))!=EOF)
    {
        cc++;
        if (ch == "")
            bc++;
        if(ch == "\n")
            lc++;
        if(ch == "\t")
            tc++;
    }
    fclose(fp);
```

```
    wc = bc + lc;
    printf("\n Number of characters = %d\n" , cc);
    printf("\n Number of tabs = %d\n" , tc);
    printf("\n Number of lines = %d\n", lc);
    printf("\n Number of blanks = %d\n" , bc);
    printf("\n Number of words count = %d\n" , wc);
    return 0;
}
```

17. Given two text documentary files "Ramayana.in" and "Mahabharatha.in". Write a C program to create a new file "Karnataka.in" that appends the content of the file "Ramayana.in" to the file "Mahabharatha.in". Also calculate the number of words and new lines in the output file.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char buff;
    FILE *fd1,*fd2,*fd3;
    int line=0,word=0;
    fd1=fopen("Mahabharatha.in","r");
    if(fd1==NULL)
    {
            printf("Error: opening Mahabharatha.in file");
            exit(1);
    }
    fd2=fopen("Ramayana.in","r");
    if(fd2==NULL)
    {
            printf("Error: opening Ramayana.in file");
            exit(1);
    }
    fd3=fopen("Karnataka.in","w");
    if(fd2==NULL)
    {
            printf("Error: Opening Karnataka.in file");
            exit(1);
    }
    //write the content of Mahabharatha.in to the file Karnataka.in
    while((buff=fgetc(fd1))!=EOF)
            fputc(buff,fd3);  //write into the file
    //Append the content of Ramayana.in to the file Karnataka.in
    while((buff=fgetc(fd2))!=EOF)
            fputc(buff,fd3);
```

```
    fclose(fd1);
    fclose(fd2);
    fclose(fd3);
    printf("Content of Karnataka.in file is:\n");
    fd1=fopen("Karnataka.in","r");
    while((buff=fgetc(fd1))!=EOF)
    {
        if(buff=='\n')
            line++;
        if(isspace(buff)||buff=='\t'||buff=='\n')
            word++;
        putchar(buff);
    }
    fclose(fd1);
    printf("\n no of lines=%d\n",line);
    printf("no of words=%d\n",word);
    return 0;
}
```

18. Given two university information files "studentname.txt" and "usn.txt" that contains students Name and USN respectively. Write a C program to create a new file called "output.txt" and copy the content of files "studentname.txt" and "usn.txt" into output file in the sequence shown below. Display the contents of output file "output.txt" on to the screen.

Student Name	USN
Name1	**USN1**
Name2	**USN2**
…	**…**
…	**…**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char buff;
    FILE *fd1,*fd2,*fd3;
    fd1=fopen("USN.txt","r");
    if(fd1==NULL)
    {
        printf("Error: opening USN.txt file");
        exit(1);
    }
    fd2=fopen("OUTPUT.txt","w");
    if(fd2==NULL)
```

```
    {
        printf("Error: Opening OUTPUT.txt file");
        exit(1);
    }
    fd3=fopen("studentname.txt","r");
    if(fd3==NULL)
    {
        printf("Error: opening studentname.txt file");
        exit(1);
    }
    fprintf(fd2,"%s\t%s\n","Student Name","USN");
    do
    {
        buff=fgetc(fd3);        //read content from file studentname.txt
        if((buff!=EOF) && (buff!='\n'))
            fputc(buff,fd2);  //write into the file output.txt
        if(buff=='\n')
        {
            fputc('\t',fd2);
            do
            {
                buff=fgetc(fd1);  //read content from USN.txt file
                if(buff!=EOF)
                    fputc(buff,fd2);  //write into the file output.txt
                if(buff=='\n')
                    break;
            }while(1);  //until end-of-file for USN.txt
        }
    }while(buff!=EOF);    //until end-of-file for NAME.txt
    fclose(fd1);
    fclose(fd2);
    fclose(fd3);
    printf("Content of output.txt file is:\n");
    fd1=fopen("output.txt","r");
    while((buff=fgetc(fd1))!=EOF)
        putchar(buff);
    fclose(fd1);
    return 0;
}
```