UNIT V
Structures, Unions and Files

Introduction to Structures:

Structure Definition: Structure is a user defined data type, used to group no. of elements of different data types with different names. A structure is defined with the keyword "struct".

Syntax :

```
struct <structure_name>
{
        Member variables;
} ;
```

Ex:

```
struct emp
{
        int no, sal;
        char name[20];
} ;
```

In above declaration, "struct emp" is data type. A variable has to be declared for this data type to make use of its members. The structure variable is declared as normal variable with syntax data_type var_name; . As here "struct emp" is data type, the variable is declared as "struct emp e;" .

In above declaration the "e" is called structure variable and it is allocated with memory for its members no, sal and name sequentially. The structure members are accessed with structure variable with a DOT " . " operator called PERIOD.

Ex:

```
        e,no=5;
        strcpy(e.name,"your_name");
        e.sal=5000;
```

---

q. Explain how to define a structure and how to access its members.

Nested structures:

We can declare a structure variable as member within another structure, like a normal other variables, and it is called nested structures.

In this case, the access the nested structure's members, the period operator is used twice as "struct_var . member_struct_var. member_var"

```
struct address
{
        char city[20];
        int pin;
};
struct employee
{
        int no;
        char name[20];
        struct address add;
};

void main ()
{
        struct  employee e;
        printff("Enter number:");
        scanf("%d",&e.no);
        printff("Enter name:");
        scanf("%s",&e.name);
        printff("Enter City:");
        scanf("%s",&e.add.city);
        printff("Enter PIN code:");
        scanf("%d",&e.add.pin);

        printf("%d  %s  %s  %d",e.no,e.name,e.add.city,e.add.pin);

}
```

Q. Write about nested structures

---

array of structures:

The structures can be created with arrays of structure variables. In that case, a loop is used to refer the index of structure variable's array.

Ex:

```
struct emp{
        int no,sal;
        char name[20];
} ;

void main()
{
        struct emp e[5];
        int i;
        for(i=0;i<5;i++)
        {
                printf("Enter 5 employee data");
                scanf("%d%s%d",&e[i].no,&e[i].name, &e[i].sal);
        }

        printf("Employee detailes are \n");
        for(i=0;i<5;i++)
                printf("%d  %s  %d\n",e[i].no, e[i].name,e[i].sal);
}
```

Q. Explain about array of structure variables.

---

structure and functions:

The structure can be declared either as local to a function or global. If a structure is defined as a local structure, it can be created with variable only within that function. If it is declared globally, it can be created with variable in any function. It is generally preferred to declare structures as global.

The structure variables can also be declared as global or as local. If the structure variable is declared as local variable, then to access it in other functions, it need to be passes as arguments from function call into function definition.

Ex:

```
struct emp{
        int no,sal;
        char name[20];
} ;
void disp(struct emp);

void main()
{
        struct emp e;
        printf("Enter  employee data");
        scanf("%d%s%d",&e.no,&e.name, &e.sal);
        disp(e);
}
void disp(struct emp e)
{
        printf("%d  %s  %d",e.no,e.name.e.sal);
}
```

Q. Explain structures and functions.

---

structures and pointers.

We can also declare a pointer to structure, if we declare a pointer to structure, then we use an operator - > called QUALIFIER to access members of the structure instead of PERIOD operator.

Ex:

```
struct emp
{
        int no, sal;
        char name[20];
} ;
void main ()
{
```

```
        struct  employee *e=(struct employee*) malloc(sizeof(structu
employee));
        printff("Enter number:");
        scanf("%d",&e->no);
        printff("Enter name:");
        scanf("%s",&e->name);
        printff("Enter salary:");
        scanf("%d",&e->sal);

        printf("%d  %s  %d", e-> no, e->name, e-> sal);
}
```

Q. Write about pointers to structures.

Unions -introduction to unions, Union definition, accessing members, difference between Structures and Unions.

Unions are similar to structures but declared with keyword "union". Similar to structures, the unions are created with variables to make use of its members. But when a union variable is declared, the largest member of the union is allocated with memory and shared by all other members.

For example:

```
union abc
{
        int x;
        float y;
} ;
```

If a union is declared as above and created with a variable, "union abc a;", then the union variable "a" is allocated with 4 bytes memory and shared by both members "x" and "y". the first 2 bytes is called "x" and all 4 bytes together is called "y". thus unions provide better memory management.

But there is a restriction on unions. As the members share same memory, programmer can use only one member at a time. In above case, if a value stored into member "x" as "a.x=5;", then "a.y" cannot be used. And if a.y is stored

with a value, then a.x cannot be used. At a given time, user can use only one member of the union variable.

Programmer has to apply his own logic while using unions like in below example.

```
union person{
    int pensioner_id, employee_id;
};
void main(){
    union person p;

}
```

In above case, the union variable “p” is assigned with only 2 bytes memory and the same is called pensioner_id and the same itself is called employee_id. A person can have either employee_id or pensioner_id. In this case, as there is a requirement of using only one at a time, the programmers can prefer unions instead of structures. Similar to structures, the unions can also be declared with array of variables. We can declare nested structures or nested unions.

Q. Explain differences between structures and unions.

Working with text files - modes: opening, reading, writing and closing text files: When a program is developed, the code is divided into three parts viz., designing user interface, process and data storage.

The user interface in C is developed with IO functions and the process is managed with operators, branching, loops, functions etc.

The data storage is managed with variables, arrays, pointers, structures, unions etc. but all these are managed in RAM, which is volatile and stores data temporarily. To use data in future, the data can be stored onto hard disk (secondary memory) in the form of a file.

Through a C program, the programmer can create a new file, write data to it, read data from it, append data to existing file etc.

Modes of operation:

Read mode : **“r”**
Write mode : **“w”**
Append mode : **“a”**

If a file is opened in “w” mode, it creates a new file. If opened in “a” mode, if a file exists, then data is appended at end of existing file and if no file exists, the a new file is created.

A file opened in a specific mode can be operated for that purpose only. As the file exists in hard disk and the program runs in RAM, the C provides a facility of using a pointer in program to point to the file by storing its address. The library stdio.h has a predefined typedef structure FILE, to which a pointer is created and used for file manipulations.

Functions related with file processing:

fopen() : this function opens given file in specified mode and returns its address.
Syntax : fopen(“file name and path”,”mode”);
Ex: FILE *fp;
fp=fopen(“c: / college /data.txt”,”w”); this code creates a file data.txt in college folder for data writing purpose.
fclose() : closes the file whose pointer is given. A file must be closed explicitly once processing is done, unless which the file gets corrupted. when a file is closed with fclose(), the C Runtime places a null value at end of file for identification called EOF (End Of File).
Ex:

```
fclose(fp);
```

fseek() : used to place the file pointer to a specific byte in the file
ftell() : returns current byte number of pointer in the file.
foef() : this function is used to identify where the file pointer reached EOF or not.

Q. Explain briefly text files in C
Q. Explain how to open and close files in C
Q. Explain file handling in C

Writing data to files:
The following functions are used to write different data values to a file after opening the file.

fputc() : used to write single character to a file
fputw() : used to write one integer to a file
fputs() : used to read a string from file
fprintf() : used to write formatted data to a file
fwrite() : used to write different data type values to a file

Ex:

```
void main()
{
        FILE *fp;
        int no,sal;
        char name[20];
        fp=fopen("data.txt", "w");
        printf("Enter no,name and salary");
        scanf("%d%s%d",&no,name,&sal);
        fprintf(fp, "%d  %s  %d\n",no,name,sal);
        fclose(fp);
}
```

Reading data from files:
The following functions are used to read data from a file after opening the file.

fgetc(): used to read single character from file
fgetw() : used to read one integer value from file
fgets() : used to write a string to a file
fscanf() : used to read formatted data from a file
fread() : used to read different type of data values from a file.

Ex:

```
void main()
{
        FILE *fp;
```

```
        int no,sal;
        char name[20];
        fp=fopen("data.txt", "r");
        fscanf(fp,"%d%s%d",&no,name,&sal);
        while(!feof(fp))
        {
                printf( "%d  %s  %d",no,name,sal);
                fscanf(fp,"%d%s%d",&no,name,&sal);
        }
        fclose(fp);
}
```

The EOF can be detected in two ways. Either by checking whether the character read through pointer has EOF or with the function feof().

```
        ch=fgetc(fp);
                if( ch==EOF)
                        exit(1);
        OR


        if( !feof( fp) )
                        exit(1);
```

"Error handling" during file processing:

There can be some common errors that raise while doing file processing. They are…..

- When trying to read a file beyond indicator.
- When trying to read a file that does not exist.
- When trying to use a file that has not been opened.
- When trying to use a file in an appropriate mode i.e., writing data to a file that has been opened for reading.
- When writing to a file that is write-protected i.e., trying to write to a read-only file.

Q. what are different file reading and file writing functions and explain with example.
