MODULE 4

Syllabus

CO4:Construct user defined data types using structure, union and files.

M4.01	Explain the definition, declaration and processing of structure data type	Understanding
M4.02	Develop programs using structure to solve problems	Applying
M4.03	Illustrate the array of structure with examples	Understanding
M4.04	Illustrate passing of structure as parameters to a function.	Understanding
M4.05	Utilize pointers to process structure data type.	Applying
M4.06	Explain features of union data type, enumerations	Understanding
M4.07	Illustratethe use of file as data storage, input and output to programs.	Understanding
M4.08	Illustrate command line arguments	Understanding

Contents:
Structure – declaration, definition and initialization of structure variables, Accessing of structure elements – Array of structure – Structure and Pointer – Structure and Function – Union - enumerations.
File – Defining, opening, closing a file - input and output operations on sequential files - Command Line arguments.

STRUCTURE

Definition:A structure is a user defined data type in C. A structure creates a data type that can be used to group items of possibly different types or heterogenous data types into a single type. It means the data types may or may not be of same type.
Each variable in the structure is known as a **member** of the structure.

Declaration of structure

You can create a structure by using the **struct** keyword and declare each of its members inside curly braces:
Consider we want to create the structure of a person with following variables name, age and address. Then such a structure can be created as

```
struct Person
{
char name[30];
int age;
char addr[50];
};
```
The general format of a structure definition is as follows

struct structure_name{
data_type member1;
data_type member2;

};

In defining a structure you may note the following syntax:

- The template is terminated with a semicolon.
- While the entire definition is considered as a statement, each member is declared independently for its name and type in a separate statement inside the template.
  Example:
  struct Person p1,p2,p3; // created structure variable for person Structure.

Initialization of structure variable

you can initialize a structure to some default value during its variable declaration.
Example:

```
// Declare and initialize structure variable
struct student stu1 = { "Pankaj", 12, 79.5f };
```

The values for the value initialized structure should match the order in which structure members are declared.

Accessing of structure variable

Members of a structure is accessed as structure variable followed by **dot(.)** operator (also called period or member access operator.)

Write a program to create a structure employee with member variables name, age, bs, da, hra and tsalary. Total Salary is calculated by the equation tsalary= (1+da+hra)* bs. Read the values of an employee and display it.

```
#include<stdio.h>
struct Employee{
char name[30];
int age;
float bs;
float da;
float hra;
float tsalary;
```

```
};
void main()
{
struct Employee e;
printf("Enter the name:");
scanf("%s",e.name);
printf("Enter the age:");
scanf("%d",&e.age);
printf("Enter the basic salary:");
scanf("%f",&e.bs);
printf("Enter the da:");
scanf("%f",&e.da);
printf("Enter the hra:");
scanf("%f",&e.hra);
e.tsalary=(1+e.da+e.hra)*e.bs;
printf("Name=%s\n",e.name);
printf("Age=%d\n",e.age);
printf("Basic Salary=%.2f\n",e.bs);
printf("DA=%.2f\n",e.da);
printf("HRA=%.2f\n",e.hra);
printf("Total Salary=%.2f\n",e.tsalary);
}
```

OUTPUT

```
Enter the name:John
Enter the age:31
Enter the basic salary:10000
Enter the da:12
Enter the hra:7.5
Name=John
Age=31
Basic Salary=10000.00
DA=12.00
HRA=7.50
Total Salary=205000.00
```

Array of structures

Structure is used to store the information of one particular object but if we need to store the information of many such objects then Array of Structure is used.

Example :

```
struct Bookinfo
{
char[20] bname;
int pages;
int price;
}Book[100];
```

Example:
Declare a structure namely Student to store the details (roll number, name, mark_for_C) of a student. Then, write a program in C to find the average mark obtained by the students in a class for the subject Programming in C (using the field mark_for_C). Use array of structures to store the required data.

```
#include<stdio.h>
struct Student{
char name[30];
int rollnum;
int mark_for_C;
};
void main(){
struct Students[30];
int i,n,sum=0;
float avg;
printf("Enter the no of Student:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the Student name:");
scanf("%s",s[i].name);
printf("Enter the Student rollnum:");
scanf("%d",&s[i].rollnum);
printf("Enter the Student Mark for C:");
scanf("%d",&s[i].mark_for_C);
}
printf("Name\tRoll Number\tMark for C\n");
for(i=0;i<n;i++)
{
        printf("%s\t%d\t%d\n",s[i].name,s[i].rollnum,s[i].mark_for_C);
sum = sum + s[i].mark_for_C;
}
avg = (float)sum / n;
printf("Average Mark=%.2f\n",avg);
}
```

OUTPUT

```
Enter the no of Student:3
Enter the Student name:John
Enter the Student rollnum:27
Enter the Student Mark for C:35
Enter the Student name:Miya
Enter the Student rollnum:24
Enter the Student Mark for C:40
Enter the Student name:Anu
Enter the Student rollnum:26
Enter the Student Mark for C:45
Name Roll Number Mark for C
```

John 27 35
Miya 24 40
Anu 26 45
Average Mark=40.00

Structure and pointers

Here's how you can create pointers to structs.

```
struct name {
   member1;
   member2;
   .
   .
};

int main()
{
   struct name *ptr;
}
```

Here, ptr is a pointer to struct.

To access members of a structure using pointers, we use the **->** operator.

Example:

```
#include <stdio.h>
struct person
{
  int age;
  float weight;
};

int main()
{
   struct person *personPtr, person1;
   personPtr = &person1;

   printf("Enter age: ");
   scanf("%d", &personPtr->age);

   printf("Enter weight: ");
   scanf("%f", &personPtr->weight);

   printf("Displaying:\n");
   printf("Age: %d\n", personPtr->age);
   printf("weight: %f", personPtr->weight);

   return 0;
}
```

In this example, the address of person1 is stored in the personPtr pointer using

personPtr = &person1;.
Now, you can access the members of person1 using the personPtr pointer.
By the way,

- personPtr->age is equivalent to (*personPtr).age
- personPtr->weight is equivalent to (*personPtr).weight

Structure and functions

Example:

```
#include <stdio.h>
struct student {
  char name[50];
  int age;
};

// function prototype
void display(struct student s);

int main() {
  struct student s1;

  printf("Enter name: ");

  // read string input from the user until \n is entered
  // \n is discarded
  scanf("%[^\n]%*c", s1.name);

  printf("Enter age: ");
  scanf("%d", &s1.age);

  display(s1); // passing struct as an argument

  return 0;
}

void display(struct student s) {
  printf("\nDisplaying information\n");
  printf("Name: %s", s.name);
  printf("\nAge: %d", s.age);
}
```

Output

```
Enter name: Bond
Enter age: 13

Displaying information
Name: Bond
Age: 13
```

Here, a struct variable s1 of type struct student is created. The variable is passed to the display() function using display(s1); statement.

Return struct from a function

Example:

```
#include <stdio.h>
struct student
{
    char name[50];
    int age;
};

// function prototype
struct student getInformation();

int main()
{
    struct student s;

    s = getInformation();

    printf("\nDisplaying information\n");
    printf("Name: %s", s.name);
    printf("\nRoll: %d", s.age);

    return 0;
}
struct student getInformation()
{
  struct student s1;

  printf("Enter name: ");
  scanf ("%[^\n]%*c", s1.name);

  printf("Enter age: ");
  scanf("%d", &s1.age);

  return s1;
}
```

Here, the getInformation() function is called using s = getInformation(); statement. The function returns a structure of type struct student. The returned structure is displayed from the main() function.
Notice that, the return type of getInformation() is also struct student.

Union

A union is a user-defined type similar to structure in C programming. We use the keyword ‘union’ to define unions. When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.
Example of Employee Union creation and declaration

```
union Employee
{
char name[30];
int age;
double salary;
} ;
union Employee e;
```

Structure	Union
struct keyword is used to define a structure	**union keyword is used to define a union**
Members do not share memory in a structure.	**Members share the memory space in a union**
Any member can be retrieved at any time in a structure	**Only one member can be accessed at a time in a union.**
Several members of a structure can be initialized at once.	**Only the first member can be initialized.**
Size of the structure is equal to the sum of size of the each member.	**Size of the union is equal to the size of the largest member.**

Enumeration datatype in C

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.
Example

```
enum State {Working = 1, Failed = 0};
```

By default, the values of the constants are as follows:
constant1 = 0, constant2 = 1, constant3 = 2 and so on.

```
enum flag{constant1, constant2, constant3, ..... };
```

```
// Another example program to demonstrate working of enum in C
#include<stdio.h>

enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,Aug, Sep, Oct, Nov, Dec};

int main()
{
int i;
for (i=Jan; i<=Dec; i++)
        printf("%d ", i);

return 0;
}
```

Output:

0 1 2 3 4 5 6 7 8 9 10 11

In this example, the for loop will run from i = 0 to i = 11, as initially the value of i is Jan which is 0 and the value of Dec is 11(If we do not explicitly assign values to enum names, the compiler by default assigns values starting from 0.)

File in C

File – Defining, opening, closing a file - input and output operations on sequential files - Command Line arguments.

A file is a collection of related data that a computer treats as a single unit. Computers store files to secondary storage so that the contents of files remain intact when a computer shuts down. When a computer reads a file, it copies the file from the storage device to memory; when it writes to a file, it transfers data from memory to the storage device. When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates. It is easy to move the data from one computer to another without any changes. C uses a structure called FILE (defined in stdio.h) to store the attributes of a file. When working with files, you need to declare a pointer of type FILE. This declaration is needed for communication between the file and the program.

```
FILE *fp;
```

Types of Files

There are two types of files

Text files

Binary files

1. Text files

Text files are the normal .txt files. You can easily create text files using any simple text editors such as Notepad. When you open those files, you'll see all the contents within the file as plain text. It is easy to edit or delete the contents. They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

2. Binary files

Binary files are mostly the .bin files in the computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold a higher amount of data, are not readable easily, and provides better security than text files.

FILE OPERATIONS

1) Opening a file

A file must be “opened” before it can be used. Opening a file is performed using the fopen() function defined in the stdio.h header file.
The syntax for opening a file in standard I/O is:

```
FILE *fp
fp = fopen("filename","mode");
```

– fp is declared as a pointer to the data type FILE.(C has a special "data type" for handling files which is defined in the standard library 'stdio.h '.)
It is called the file pointer and has the syntax FILE*.
– filename is a string - specifies the name of the file.
– fopen returns a pointer to the file which is used in all subsequent file operations.

File Opening Mode

1. **r** Opens an existing text file for reading purpose.
2. **w** Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
3. **a** Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.
4. **r+** Opens a text file for both reading and writing.
5. **w+** Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
6. **a+** Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

2) Closing a file

The file (both text and binary) should be closed after reading/writing. Closing a file is performed using the fclose() function.

```
fclose(fp);
```

Here, fp is a file pointer associated with the file to be closed.
– fclose() function closes the file and returns zero on success, or EOF if there is an error in closing the file.
– This **EOF** is a constant defined in the header file stdio.h.

3) Reading and writing to a file

- **fgetc()** To read a character from a file
  ch = fgetc(fp)
- **fputc()** To write a character to a file
  fputc(ch,fp)
- **fscanf()** To read numbers, strings from a file
  fscanf(fp,"%d",&n)
- **fprintf()** To write numbers, strings to a file
  fprintf(fp,"%d",n)

RANDOM ACCESS TO A FILE

1) rewind()
The rewind() function sets the file pointer at the beginning of the stream. It is useful if you have to use stream many times.

rewind(file pointer)

2) fseek()
If you have many records inside a file and need to access a record at a specific position, you need to loop through all the records before it , to get the record. This will waste a lot of memory and operation time. An easier way to get to the required data can be achieved using fseek().

fseek(FILE * stream, long int offset, int pos);

The first parameter stream is the pointer to the file. The second parameter is the position of the record to be found, and the third parameter specifies the location where the offset starts.

- **Different positions in fseek()**
  SEEK_SET Starts the offset from the beginning of the file.
  SEEK_END Starts the offset from the end of the file.
  SEEK_CUR Starts the offset from the current location of the cursor in the file.

3) ftell()
ftell() in C is used to find out the position of file pointer in the file with respect to starting of the file.
Syntax of ftell() is:

ftell(FILE *pointer)

Difference between sequential and random access

Sequential file access is the method employed in tape drives where the files are accessed in a sequential manner. So if you have to get a file in the end of the tape you have

to start from the beginning till it reaches the beginning of the file.
Random access files are similar to the one in Hard Disks and Optical drives, wherever the files is placed it will go to that particular place and retrieve it.
Accessing data sequentially is much faster than accessing it randomly because of the way in which the disk hardware works.
Write a program to display the content of a file.

```
#include<stdio.h>
void main()
{
FILE *fp;
char ch;
fp = fopen("test.txt","r");
while(feof(fp) == 0)
{
ch=fgetc(fp);
printf("%c",ch);
}
fclose(fp);
}
```

Content of test.txt
Hello, Welcome to C Programming Lectures.

OUTPUT

Hello, Welcome to C Programming Lectures.

Command line arguments

To pass command line arguments, we typically define main() with two arguments : first argument is the **number of command line arguments** and second is **list of command-line arguments.**

```
int main(int argc, char *argv[])
```

- argc (ARGument Count) is int and stores number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)
- The value of argc should be non negative.
- argv(ARGument Vector) is array of character pointers listing all the arguments.
- If argc is greater than zero,the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- Argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.
- **Example**

```
#include <stdio.h>
int main( int argc, char *argv[] )  {
  if( argc == 2 ) {
    printf("The argument supplied is %s\n", argv[1]);
```

```
   }
   else if( argc > 2 ) {
      printf("Too many arguments supplied.\n");
   }
   else {
      printf("One argument expected.\n");
   }
}
```

When the above code is compiled and executed with single argument, it produces the following result.

$./a.out testing

The argument supplied is testing

When the above code is compiled and executed with a two arguments, it produces the following result.

$./a.out testing1 testing2

Too many arguments supplied.

When the above code is compiled and executed without passing any argument, it produces the following result.

$./a.out

One argument expected

It should be noted that argv[0] holds the name of the program itself and argv[1] is a pointer to the first command line argument supplied, and *argv[n] is the last argument. If no arguments are supplied, argc will be one, and if you pass one argument then argc is set at 2.

You pass all the command line arguments separated by a space