PROGRAMMING IN C

UNIT-4

Structure:

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures ca; simulate the use of classes and templates as it can store various information

The, struct keyword is used to define the structure.

Syntax:

```
struct structure_name
{
data_type member1;
data_type member2;
……..
……..
data_type memeberN;
};
```


Example:

```
struct employee
{   int id;
char name[20];
float salary;
};
```

Declaring structure variable:

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

```
struct employee
{   int id;
    char name[50];
    float salary;
};
```

Now write given code inside the main() function.

```
struct employee e1, e2;
```

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in C++ and Java.

2. By declaring a variable at the time of defining the structure.

Let's see another way to declare variable at the time of defining the structure.

```
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2;
```


Accessing members of the structure:

There are two ways to access structure members:

1. Member or dot operator(.):

The dot (.) operator is used for direct member selection via object name. In other words, it is used to access the child object.

2. structure pointer operator(->):

To access the members of the structure referenced using the pointer we use the operator “->”. This operator is called as arrow operator. Using this we can access all the members of the structure and we can further do all operations on them.

Structure Initialization:

Generally, the initialization of the structure variable is done after the structure declaration. Just after structure declaration put the braces (i.e. {}) and inside it an equal sign (=) followed by the values must be in the order of members specified also each value must be separated by commas. The example below will show how to initialize structure variable in C programming.

Example:

Program : To demonstrate initialization of structure in C.

```
#include<stdio.h>

#include<conio.h>

void main(void)

{

struct language

{

char name;

int year;

};

struct language

dr={'C', 1972};

printf("%c%d\n", dr.name, dr. year);

getch();
```

OUTPUT:

C1972

Operations performed on structure variable:

The only operation that is allowed on structure variables is the assignment operation. Two variables of the same structure can be copied similar to ordinary variables.

If P1 and P2 are variables of struct P, then P1 values can be assigned to P2 as

P2=P1;

where values of P1 will be assigned to P2 member by member.

Operation on Individual Members of Structures

All operations are valid on individual members of structures.

Example:

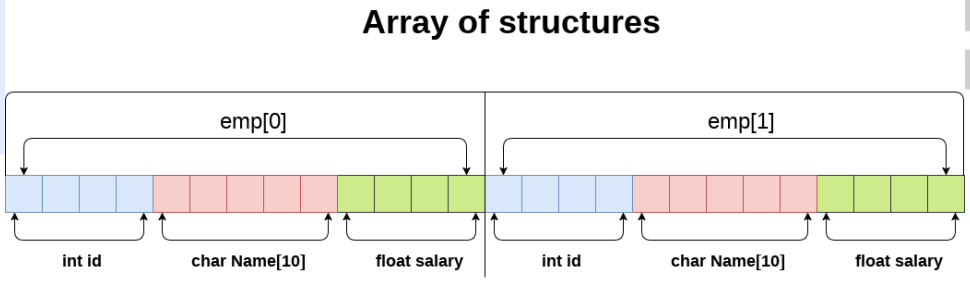
```
#include<stdio.h>
#include<string.h>
struct student
{
int rno;
char name[10];
int marks,age;
};
void main()
{
int m;
//assigning values to structure variable s1 using initalization
struct student s1={2,"Gandhi",89,18};
struct student s2;
s2=s1;
printf("\nDetails of student 1:\n");
printf("\n roll number: %d",s1.rno);
printf("\n name :%s",s1.name);
printf("\n marks: %d",s1.marks);
printf("\n age: %d",s1.age);
printf("\n\n");
printf("Details of student 2:\n");
printf("\n roll number: %d",s2.rno);
printf("\n name :%s",s2.name);
printf("\n marks: %d",s2.marks);
printf("\n age: %d",s2.age);
//comparsion of two student details.
m=((s1.rno==s2.rno)&&(s1.marks==s2.marks))?1:0;
if(m==1)
printf("\n both the details are same");
else
printf("\n both the details are not same");
}
```

Output:

```
Details of student 1:

 roll number: 2
 name :Gandhi
 marks: 89
 age: 18

Details of student 2:

 roll number: 2
 name :Gandhi
 marks: 89
 age: 18
 both the details are same

...Program finished with exit code 0
Press ENTER to exit console.
```

Array of Structures:

An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.




```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

sizeof (emp) = 4 + 5 + 4 = 13 bytes

sizeof (emp[2]) = 26 bytes

Let's see an example of an array of structures that stores information of 5 students and prints it.

```
#include<stdio.h>

#include <string.h>

struct student{

int rollno;
```

```
char name[10];
};
int main(){
int i;
struct student st[5];
printf("Enter Records of 5 students");
for(i=0;i<5;i++){
printf("\nEnter Rollno:");
scanf("%d",&st[i].rollno);
printf("\nEnter Name:");
scanf("%s",&st[i].name);
}
printf("\nStudent Information List:");
for(i=0;i<5;i++){
printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
  return 0;
}
```


Output:

```
Enter Records of 5 students
Enter Rollno:1
Enter Name:Sonoo
Enter Rollno:2
Enter Name:Ratan
Enter Rollno:3
Enter Name:Vimal
Enter Rollno:4
Enter Name:James
Enter Rollno:5
Enter Name:Sarfraz

Student Information List:
```

Rollno:1, Name:Sonoo
Rollno:2, Name:Ratan
Rollno:3, Name:Vimal
Rollno:4, Name:James
Rollno:5, Name:Sarfraz

Union:

Union can be defined as a user-defined data type which is a collection of different variables of different data types in the same memory location. The union can also be defined as many members, but only one member can contain a value at a particular point in time.

Union is a user-defined data type, but unlike structures, they share the same memory location.

The size of the union is based on the size of the largest member of the union.

Let's understand this through an example.

```
struct abc
{
  int a;
  char b;
}
```


When we define the union, then we found that union is defined in the same way as the structure is defined but the difference is that union keyword is used for defining the union data type, whereas the struct keyword is used for defining the structure. The union contains the data members, i.e., 'a' and 'b', when we check the addresses of both the variables then we found that both have the same addresses. It means that the union members share the same memory location.

Accessing members of union using pointers

We can access the members of the union through pointers by using the (->) arrow operator.

Let's understand through an example.

```
#include <stdio.h>
union abc
{
  int a;
  char b;
```

```
};
int main()
{
   union abc *ptr; // pointer variable declaration
   union abc var;
   var.a= 90;
   ptr = &var;
   printf("The value of a is : %d", ptr->a);
   return 0;
}
```

File Handling:

In programming, we may require some specific input data to be generated several numbers of times. Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling in C.

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

- Creation of the new file
- Opening an existing file
- Reading from the file
- Writing to the file
- Deleting the file

Functions for file handling:

There are many functions in the C library to open, read, write, search and close the file. A list of file functions are given below:

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file

6	fclose()	closes the file
7	fseek()	sets the file pointer to given position
8	fputw()	writes an integer to file
9	fgetw()	reads an integer from file
10	ftell()	returns current position
11	rewind()	sets the file pointer to the beginning of the file

Opening File: fopen():

We must open a file before it can be read, write, or update. The fopen() function is used to open a file.

Syntax:

```
FILE *fopen( const char * filename, const char * mode );
```

The fopen() function accepts two parameters:

1. The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name can be like **"c://some_folder/some_file.ext"**.
2. The mode in which the file is to be opened. It is a string.

The fopen function works in the following way.

1. Firstly, it searches the file to be opened.
2. Then, it loads the file from the disk and place it into the buffer. The buffer is used to provide efficiency for the read operations.
3. It sets up a character pointer which points to the first character of the file.

Closing File: fclose()

The fclose() function is used to close a file. The file must be closed after performing all the operations on it.

Syntax:

```
int fclose( FILE *fp );
```

Need of File Handling:

There are times when the output generated out of a program after its compilation and running do not serve our intended purpose. In such cases, we might want to check the program's output various times. Now, compiling and running the very same program multiple times becomes a tedious task for any programmer. It is exactly where file handling becomes useful.

Let us look at a few reasons why file handling makes programming easier for all:

Reusability: File handling allows us to preserve the information/data generated after we run the program.

Saves Time: Some programs might require a large amount of input from their users. In such cases, file handling allows you to easily access a part of a code using individual commands.

Commendable storage capacity: When storing data in files, you can leave behind the worry of storing all the info in bulk in any program.

Portability: The contents available in any file can be transferred to another one without any data loss in the computer system. This saves a lot of effort and minimises the risk of flawed coding.

Input/Output Operations on Files:

1. fprintf() function:

The fprintf() function is used to write set of characters into file. It sends formatted output to a stream.

Syntax:

```
int fprintf(FILE *stream, const char *format [, argument, ...])
```


Example:

```
#include <stdio.h>
main(){
  FILE *fp;
  fp = fopen("file.txt", "w");//opening file
  fprintf(fp, "Hello file by fprintf...\n");//writing data into file
  fclose(fp);//closing file
}
```

2. fscanf() function:

The fscanf() function is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.

Syntax:

```
int fscanf(FILE *stream, const char *format [, argument, ...])
```

Example:

```
#include <stdio.h>
main(){
  FILE *fp;
  char buff[255];//creating char array to store data of file
  fp = fopen("file.txt", "r");
  while(fscanf(fp, "%s", buff)!=EOF){
  printf("%s ", buff );
  }
  fclose(fp);
}
```

Output:

Hello file by fprintf...

3. fputc() function

The fputc() function is used to write a single character into file. It outputs a character to a stream.

Syntax:

```
int fputc(int c, FILE *stream)
```

Example:

```
#include <stdio.h>
main(){
  FILE *fp;
  fp = fopen("file1.txt", "w");//opening file
  fputc('a',fp);//writing single character into file
  fclose(fp);//closing file
}
```

4. fgetc() function:

The fgetc() function returns a single character from the file. It gets a character from the stream. It returns EOF at the end of file.

Syntax:

int fgetc(**FILE** *stream)

Example:

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
char c;
clrscr();
fp=fopen("myfile.txt","r");

while((c=fgetc(fp))!=EOF){
printf("%c",c);
}
fclose(fp);
getch();
}
```