Structure

Structure is a user defined data type which hold or store heterogeneous/different types data item or element in a single variable. It is a Combination of primitive and derived data type.
Or
A structure is a collection of one or more data items of different data types, grouped together under a single name.
Variables inside the structure are called members of structure.
Each element of a structure is called a member.
struct keyword is used to define/create a structure. struct define a new data type which is a collection of different type of data.

Syntax

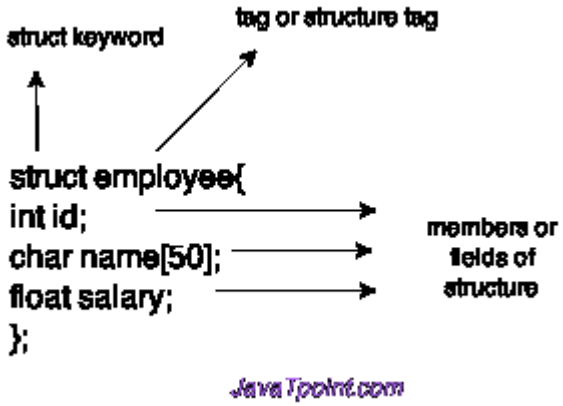
```
struct structure_name /tag name
{
data_type member1;
data_type member2;
data_type member n;
};
```

Note: Don't forget the semicolon }; in the ending line.

Example

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

Here, struct is the keyword, employee is the tag name of
structure; id, name and salary are the members or fields of the structure. Let's
understand it by the diagram given below:



Syntax to create structure variable

```
struct tagname/structure_name variable;
```

Declaring structure variable

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

1. By struct keyword within main() function/ Declaring Structure variables separately
2. By declaring variable at the time of defining structure/ Declaring Structure Variables with Structure definition

1st way:

Let's see the example to declare 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;
```

2nd way:

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

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

Which approach is good

But if no. of variable are not fixed, use 1st approach. It provides you flexibility to declare the structure variable many times.

If no. of variables are fixed, use 2nd approach. It saves your code to declare variable in main() function.

Structure Initialization

structure variable can also be initialized at compile time.

```
struct Patient
{
float height;
int weight;
int age;
};
struct Patient p1 = { 180.75 , 73, 23 }; //initialization
or
struct patient p1;
p1.height = 180.75; //initialization of each member separately
p1.weight = 73;
p1.age = 23;
```

Accessing Structures/ Accessing members of structure

There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)
When the variable is normal type then go for struct to member operator.
Any member of a structure can be accessed as:

structure_variable_name.member_name
Example
```
struct book
{
char name[20];
char author[20];
int pages;
}; struct book b1;
```
for accessing the structure members from the above example

b1.name, b1.author, b1.pages:
Example
```
struct emp
{
int id;
char name[36];
int sal;
};
sizeof(struct emp) // --> 40 byte (2byte+36byte+2byte)
```

Example of Structure in C
```
#include<stdio.h>
#include<conio.h>
struct emp
{
int id;
char name[36];
float sal;
};
void main()
{
struct emp e;
clrscr();
printf("Enter employee Id, Name, Salary: ");
scanf("%d",&e.id);
scanf("%s",&e.name);
scanf("%f",&e.sal);
printf("Id: %d",e.id);
printf("\nName: %s",e.name);
printf("\nSalary: %f",e.sal);
getch();
}
```
Output

Output: Enter employee Id, Name, Salary: 5 Spidy 45000
Id : 05
Name: Spidy
Salary: 45000.00
Example

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

struct employee
{ int id;
char name[50];
}e1; //declaring e1 variable for structure
int main( )
{
//store first employee information
e1.id=101;
strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
//printing first employee information printf(
"employee 1 id : %d\n", e1.id);
printf( "employee 1 name : %s\n", e1.name);
return 0;
}
```

Output:

```
employee
1 id : 101
employee
1 name :
Sonoo
Jaiswal
```

Unions

A union is a special data type available in C that allows to store different data types in the same memory location.Unions are conceptually similar to structures. The syntax of union is also similar to that of structure. The only difference is in terms of storage. In structure each member has its own storage location, whereas all members of union use a single shared memory location which is equal to the size of its largest data member.

We can access only one member of union at a time. We can‟t access all member values at the same time in union. But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.

Difference between Structure and Union

Struct	Union
The struct keyword is used to define a structure.	The union keyword is used to define union.
When the variables are declared in a structure, the compiler allocates memory to each variables member. The size of a structure is equal or greater to the sum of the sizes of each data member.	When the variable is declared in the union, the compiler allocates memory to the largest size variable member. The size of a union is equal to the size of its largest data member size.
Each variable member occupied a unique memory space.	Variables members share the memory space of the largest size variable.
Changing the value of a member will not affect other variables members.	Changing the value of one member will also affect other variables members.
Each variable member will be assessed at a time.	Only one variable member will be assessed at a time.
We can initialize multiple variables of a structure at a time.	In union, only the first data member can be initialized.
All variable members store some value at any point in the program.	Exactly only one data member stores a value at any particular instance in the program.
The structure allows initializing multiple variable members at once.	Union allows initializing only one variable member at once.
It is used to store different data type values.	It is used for storing one at a time from different data type values.
It allows accessing and retrieving any data member at a time.	It allows accessing and retrieving any one data member at a time.

Advantages of Structure

- Structure stores more than one data type of the same object together.
- It is helpful when you want to store data of different or similar data types such as name, address, phone, etc., of the same object.
- It makes it very easy to maintain the entire record as we represent complete records using a single name.
- The structure allows passing a whole set of records to any function with the help of a single parameter.
- An array of structures can also be created to store multiple data of similar data types.

Disadvantages of Structure

- If the complexity of the project goes increases, it becomes hard to manage all your data members.
- Making changes to one data structure in a program makes it necessary to change at several other places. So it becomes difficult to track all changes.
- The structure requires more storage space as it allocates memory to all the data members, and even it is slower.
- The structure takes more storage space as it gives memory to all the different data members, whereas union takes only the memory size required by the largest data size parameters, and the same memory is shared with other data members.

Comparative advantages and disadvantages of union

Below is the list of some advantages and disadvantages of using union:

Advantages of Union

- Union takes less memory space as compared to the structure.
- Only the largest size data member can be directly accessed while using a union.
- It is used when you want to use less (same) memory for different data members.
- It allocates memory size to all its data members to the size of its largest data member.

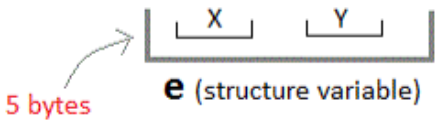
Disadvantages of Union

- It allows access to only one data member at a time.
- Union allocates one single common memory space to all its data members, which are shared between all of them.

- Not all the union data members are initialized, and they are used by interchanging values at a time.

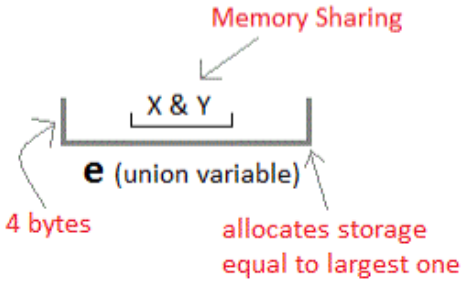
Structure

```
struct Emp
{
 char X ;     // size 1 byte
 float Y ;    // size 4 byte
} e ;
```



Unions

```
union Emp
{
 char X ;
 float Y ;
} e ;
```



Structure

```
struct Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 7 byte
```

size of e1= 1 + 2 + 4 = 7

Union

```
union Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 4 byte
```

size of e1= 4 (maximum size of 1 element)

syntax

```
union union_name
{
data_type member1;
data_type member2;
.
.
data_type memeberN;
};
```

Example

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

```
#include <stdio.h>
#include <string.h>
union employee
{ int id;
char name[50];
}e1; //declaring e1 variable for union
int main( )
{
//store first employee information
e1.id=101;
strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
//printing first employee information
printf( "employee 1 id : %d\n", e1.id);
printf( "employee 1 name : %s\n", e1.name);
return 0;
}
```

Output:

```
employee 1 id : 1869508435
employee 1 name : Sonoo Jaiswal
```

As you can see, id gets garbage value because name has large memory size. So only name will have actual value.

Example

```
#include <stdio.h>
#include <conio.h>
union item
{
int a;
float b;
char ch;
```

```
};
int main( )
{
union item it;
it.a = 12;
it.b = 20.2;
it.ch='z';
clrscr();
printf("%d\n",it.a);
printf("%f\n",it.b);
printf("%c\n",it.ch);
getch();
return 0;
}
```

Output

```
-26426
20.1999
z
```

As you can see here, the values of a and b get corrupted and only variable c prints the expected result. Because in union, the only member whose value is currently stored will have the memory

Array of Structures

Consider a case, where we need to store the data of 5 students. We can store it by using the structure as given below.

```
#include<stdio.h>
stuct student
{
   char name[20];
   int id;
   float marks;
};
void main()
{
   struct student s1,s2,s3;
   int dummy;
   printf("Enter the name, id, and marks of student 1 ");
   scanf("%s %d %f",s1.name,&s1.id,&s1.marks);
   scanf("%c",&dummy);
   printf("Enter the name, id, and marks of student 2 ");
   scanf("%s %d %f",s2.name,&s2.id,&s2.marks);
```

```
    scanf("%c",&dummy);
    printf("Enter the name, id, and marks of student 3 ");
    scanf("%s %d %f",s3.name,&s3.id,&s3.marks);
    scanf("%c",&dummy);
    printf("Printing the details....\n");
    printf("%s %d %f\n",s1.name,s1.id,s1.marks);
    printf("%s %d %f\n",s2.name,s2.id,s2.marks);
    printf("%s %d %f\n",s3.name,s3.id,s3.marks);
}
```

Output

```
Enter the name, id, and marks of student 1 James 90 90
Enter the name, id, and marks of student 2 Adoms 90 90
Enter the name, id, and marks of student 3 Nick 90 90
Printing the details....
James 90 90.000000
Adoms 90 90.000000
Nick 90 90.000000
```

In the above program, we have stored data of 3 students in the structure. However, the complexity of the program will be increased if there are 20 students. In that case, we will have to declare 20 different structure variables and store them one by one. This will always be tough since we will have to declare a variable every time we add a student. Remembering the name of all the variables is also a very tricky task. However, c enables us to declare an array of structures by using which, we can avoid declaring the different structure variables; instead we can make a collection containing all the structures that store the information of different entities.

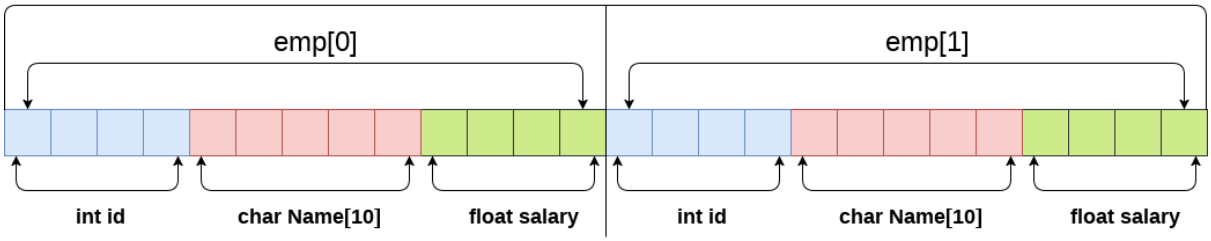
Array of Structures in C

An array of structres 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.

Array 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

Nested Structure in C

C provides us the feature of nesting one structure within another structure by using which, complex data types are created. For example, we may need to store the address of an entity employee in a structure. The attribute address may also have the subparts as street number, city, state, and pin code. Hence, to store the address of the employee, we need to store the address of the employee into a separate structure and nest the structure address into the structure employee. Consider the following program.

```c
#include<stdio.h>
struct address
{
   char city[20];
   int pin;
   char phone[14];
};
struct employee
{
   char name[20];
   struct address add;
};
void main ()
{
   struct employee emp;
   printf("Enter employee information?\n");
   scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
   printf("Printing the employee information....\n");
   printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s",emp.name,emp.add.city,emp.add.pin,emp.ad
}
```

Output

```
Enter employee information?

Arun

Delhi

110001

1234567890

Printing the employee information....

name: Arun

City: Delhi

Pincode: 110001

Phone: 1234567890
```

Structure Pointer

The **structure pointer** points to the address of a memory block where the Structure is being stored. Like a pointer that tells the address of another variable of any data type (int, char, float) in memory. And here, we use a structure pointer which tells the address of a structure in memory by pointing pointer variable **ptr** to the structure variable.

Declare a Structure Pointer

The declaration of a structure pointer is similar to the declaration of the structure variable. So, we can declare the structure pointer and variable inside and outside of the main() function. To declare a pointer variable in C, we use the asterisk (*) symbol before the variable's name

struct structure_name *ptr;

After defining the structure pointer, we need to initialize it, as the code is shown:

Initialization of the Structure Pointer

ptr = &structure_variable;

We can also initialize a Structure Pointer directly during the declaration of a pointer.

struct structure_name *ptr = &structure_variable;

As we can see, a pointer **ptr** is pointing to the address structure_variable of the Structure.

Access Structure member using pointer:

There are two ways to access the member of the structure using Structure pointer:

1. Using (*) asterisk or indirection operator and dot (.) operator.
2. Using arrow (->) operator or membership operator.

```
#include <stdio.h>

// create a structure Subject using the struct keyword
struct Subject
```

```
{
   // declare the member of the Course structure
   char sub_name[30];
   int sub_id;
   char sub_duration[50];
   char sub_type[50];
};

int main()
{
   struct Subject sub; // declare the Subject variable
   struct Subject *ptr; // create a pointer variable (*ptr)
   ptr = ⊂ /* ptr variable pointing to the address of the structure variable sub */

   strcpy (sub.sub_name, " Computer Science");
   sub.sub_id = 1201;
   strcpy (sub.sub_duration, "6 Months");
   strcpy (sub.sub_type, " Multiple Choice Question");

   // print the details of the Subject;
   printf (" Subject Name: %s\t ", (*ptr).sub_name);
         printf (" \n Subject Id: %d\t ", (*ptr).sub_id);
      printf (" \n Duration of the Subject: %s\t ", (*ptr).sub_duration);
         printf (" \n Type of the Subject: %s\t ", (*ptr).sub_type);

   return 0;

}
```

Output:

```
Subject Name:  Computer Science
 Subject Id: 1201
 Duration of the Subject: 6 Months
 Type of the Subject:  Multiple Choice Question
```

In the above program, we have created the **Subject** structure that contains different data elements like sub_name (char), sub_id (int), sub_duration (char), and sub_type (char). In this, the **sub** is the structure variable, and ptr is the structure pointer

variable that points to the address of the sub variable like ptr = &sub. In this way, each *ptr is accessing the address of the Subject structure's member.

```
#include <stdio.h>

// create Employee structure
struct Employee
{
    // define the member of the structure
    char name[30];
    int id;
    int age;
    char gender[30];
    char city[40];
};

// define the variables of the Structure with pointers
struct Employee emp1, emp2, *ptr1, *ptr2;

int main()
{
    // store the address of the emp1 and emp2 structure variable
    ptr1 = &emp1;
    ptr2 = &emp2;

    printf (" Enter the name of the Employee (emp1): ");
    scanf (" %s", &ptr1->name);

    printf (" Enter the id of the Employee (emp1): ");
    scanf (" %d", &ptr1->id);
    printf (" Enter the age of the Employee (emp1): ");
    scanf (" %d", &ptr1->age);
    printf (" Enter the gender of the Employee (emp1): ");
    scanf (" %s", &ptr1->gender);
    printf (" Enter the city of the Employee (emp1): ");
    scanf (" %s", &ptr1->city);
```

```
    printf (" \n Second Employee: \n");
          printf (" Enter the name of the Employee (emp2): ");
    scanf (" %s", &ptr2->name);

    printf (" Enter the id of the Employee (emp2): ");
    scanf (" %d", &ptr2->id);
    printf (" Enter the age of the Employee (emp2): ");
    scanf (" %d", &ptr2->age);
    printf (" Enter the gender of the Employee (emp2): ");
    scanf (" %s", &ptr2->gender);
    printf (" Enter the city of the Employee (emp2): ");
    scanf (" %s", &ptr2->city);

    printf ("\n Display the Details of the Employee using Structure Pointer");
    printf ("\n Details of the Employee (emp1) \n");
    printf(" Name: %s\n", ptr1->name);
    printf(" Id: %d\n", ptr1->id);
    printf(" Age: %d\n", ptr1->age);
    printf(" Gender: %s\n", ptr1->gender);
    printf(" City: %s\n", ptr1->city);

    printf ("\n Details of the Employee (emp2) \n");
    printf(" Name: %s\n", ptr2->name);
    printf(" Id: %d\n", ptr2->id);
    printf(" Age: %d\n", ptr2->age);
    printf(" Gender: %s\n", ptr2->gender);
    printf(" City: %s\n", ptr2->city);
    return 0;
}
```

Output:

```
Enter the name of the Employee (emp1): John
 Enter the id of the Employee (emp1): 1099
 Enter the age of the Employee (emp1): 28
 Enter the gender of the Employee (emp1): Male
 Enter the city of the Employee (emp1): California

 Second Employee:
 Enter the name of the Employee (emp2): Maria
 Enter the id of the Employee (emp2): 1109
```

```
Enter the age of the Employee (emp2): 23
Enter the gender of the Employee (emp2): Female
Enter the city of the Employee (emp2): Los Angeles

Display the Details of the Employee using Structure Pointer
Details of the Employee (emp1)
Name: John
Id: 1099
Age: 28
Gender: Male
City: California

Details of the Employee (emp2) Name: Maria
Id: 1109
Age: 23
Gender: Female
City: Los Angeles
```

In the above program, we have created an **Employee** structure containing two structure variables **emp1** and **emp2**, with the pointer variables ***ptr1** and ***ptr2**. The structure Employee is having the name, id, age, gender, and city as the member. All the Employee structure members take their respective values from the user one by one using the pointer variable and arrow operator that determine their space in memory.

What is File?

A file a container in a computer system that stores data, information, settings, or commands, which are used with a computer program. In graphical user interface (GUI), such as Microsoft operating systems, represent the files as icons, which associate to the program that opens the file. For instance, the picture is shown as an icon; it is related to Microsoft Word

. If your computer contains this file and you double-click on the icon, it will open in Microsoft Word installed on the computer.

There are several types of files available such as directory files, data files, text files, binary and graphic files, and these several kinds of files contain different types of information. In the computer system, files are stored on hard drives, optical drives, discs, or other storage devices.

In most of the operating systems, a file must be saved with a unique name within a given file directory. However, certain characters cannot be used during creating a file as they are considered illegal. A filename is consisted of with a file extension that is also called a suffix. The file extension contains two to four characters that follow the complete filename, and it helps to recognize the file format, type of file, and the attributes related to the file.

Most modern computer systems have the ability to protect files from file corruption or damage. The file can be contained the data from system-generated information to user-specified information. File management is done manually at times with the help of the user or done with the help of third-party tools and operating systems.

File Handling in C

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. The syntax of the fopen() is given below.

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

The fopen() function accepts two parameters:

- 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"**.
- The mode in which the file is to be opened. It is a string.

We can use one of the following modes in the fopen() function.

Mode	Description
R	opens a text file in read mode
W	opens a text file in write mode
A	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode

The fopen function works in the following way.

- Firstly, It searches the file to be opened.
- 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.
- It sets up a character pointer which points to the first character of the file.

Consider the following example which opens a file in write mode.

```
#include<stdio.h>
void main( )
{
FILE *fp ;
char ch ;
fp = fopen("file_handle.c","r") ;
while ( 1 )
{
ch = fgetc ( fp ) ;
if ( ch == EOF )
break ;
printf("%c",ch) ;
}
fclose (fp ) ;
}
```

Output

The content of the file will be printed.

```
#include;
void main( )
{
FILE *fp; // file pointer
char ch;
fp = fopen("file_handle.c","r");
while ( 1 )
{
```

```
ch = fgetc ( fp ); //Each character of the file is read and stored in the
character file.
if ( ch == EOF )
break;
printf("%c",ch);
}
fclose (fp );
}
```

Closing File: fclose()

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

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

fprintf() and fscanf()

Writing File : 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, ...]) ;
```

```
#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
}
```

Reading File : 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...
```

Let's see a file handling example to store employee information as entered by user from console. We are going to store id, name and salary of the employee.

```
#include <stdio.h>
void main()
{
   FILE *fptr;
   int id;
   char name[30];
   float salary;
   fptr = fopen("emp.txt", "w+");/*  open for writing */
   if (fptr == NULL)
   {
      printf("File does not exists \n");
      return;
   }
   printf("Enter the id\n");
   scanf("%d", &id);
   fprintf(fptr, "Id= %d\n", id);
   printf("Enter the name \n");
   scanf("%s", name);
   fprintf(fptr, "Name= %s\n", name);
   printf("Enter the salary\n");
   scanf("%f", &salary);
```

```
    fprintf(fptr, "Salary= %.2f\n", salary);
    fclose(fptr);
}
```

Output:

```
Enter the id
1
Enter the name
sonoo
Enter the salary
120000
```

Now open file from current directory. For windows operating system, go to TC\bin directory, you will see emp.txt file. It will have following information.

emp.txt

```
Id= 1
Name= sonoo
Salary= 120000
```

fputc() and fgetc()

Writing File : 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
}
```

file1.txt

a

Reading File : 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();
}
```

myfile.txt

```
this is simple text message
```

fputs() and fgets()

The fputs() and fgets() in C programming are used to write and read string from stream. Let's see examples of writing and reading file using fgets() and fgets() functions.

Writing File : fputs() function

The fputs() function writes a line of characters into file. It outputs string to a stream.

Syntax:

int fputs(**const char** *s, **FILE** *stream) ;

Example:

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
clrscr();

fp=fopen("myfile2.txt","w");
fputs("hello c programming",fp);

fclose(fp);
getch();
}
```

myfile2.txt

```
hello c programming
```

Reading File : fgets() function

The fgets() function reads a line of characters from file. It gets string from a stream.

Syntax:

char* fgets(**char** *s, **int** n, **FILE** *stream) ;

Example:

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
char text[300];
```

```
clrscr();

fp=fopen("myfile2.txt","r");
printf("%s",fgets(text,200,fp));

fclose(fp);
getch();
}
```

Output:

```
hello c programming
```

This file handling C program illustrates how to write into a file using putw() function and how to read the same file using getw() function.C

```
#include <stdio.h>

int main ()
{

  FILE *fp;
  int i=1, j=2, k=3, num;
  fp = fopen ("test.c","w");
  putw(i,fp);
  putw(j,fp);
  putw(k,fp);
  fclose(fp);

  fp = fopen ("test.c","r");

 while(1)
{num=getw(fp);
if(num==EOF)
break;
```

```
    printf("Data in test.c file is %d \n", num);
  }
  fclose(fp);
  return 0;
}
```

OUTPUT:

```
Data in test.c file is
1
2
3
```

Random access to files

Random accessing of files in C language can be done with the help of the following functions −

- ftell ( )
- rewind ( )
- fseek ( )

ftell ( )

It returns the current position of the file ptr.

The syntax is as follows −

```
int n = ftell (file pointer)
```

For example,

```
void main( )
{
FILE *fp;
int n;
fp=fopen("text.txt","w");
fputs("welcome in c programming",fp);
n = ftell (fp);
printf("file pointer:%d",n);
```

```
getch();
}
```

Output:

file pointer:24

Note − ftell ( ) is used for counting the number of characters which are entered into a file.

rewind ( )

It makes file ptr move to beginning of the file.

The syntax is as follows −

```
rewind (file pointer);
```

For example,

```
void main( )
{
FILE *fp;
int n;
_fp=fopen("text.txt","w");
fputs("welcome in c programming",fp);
n = ftell (fp);
printf("file pointer:%d",n);
rewind(fp);
n=ftell(fp);
printf("file pointer %d",n);
getch();
}
```

Output:

file pointer:24

file pointer 0

Output

The output is as follows –

0 (always).

fseek ( )

The fseek() function is used to set the file pointer to the specified offset. It is used to write data into file at desired location.

Syntax:

The syntax is as follows −

fseek(file pointer, offset, position);

Offset

- The no of positions to be moved while reading or writing.
- If can be either negative (or) positive.
    - Positive - forward direction.
    - Negative – backward direction.

Position

It can have three values, which are as follows −

- 0 – Beginning of the file.
- 1 – Current position.
- 2 – End of the file.

There are 3 constants used in the fseek() function SEEK_SET, SEEK_CUR and SEEK_END.

SEEK_SET : offset is relative to beginning of the file
SEEK_CUR : offset is relative to the current position in the file
SEEK_END : offset is relative to end of the file

Example

- fseek (fp,0,2) - fp moved 0 bytes forward from the end of the file.
- fseek (fp, 0, 0) – fp moved 0 bytes forward from beginning of the file
- fseek (fp, m, 0) – fp moved m bytes forward from the beginning of the file.
- fseek (fp, -m, 2) – fp moved m bytes backward from the end of the file.

Errors

The errors related to fseek () function are as follows

- fseek (fp, -m, 0);
- fseek(fp, +m, 2);

Example:

```
#include <stdio.h>
void main(){
  FILE *fp;

  fp = fopen("myfile.txt","w+");
  fputs("This is javatpoint", fp);

  fseek( fp, 7, SEEK_SET );
  fputs("sonoo jaiswal", fp);
  fclose(fp);
}
```

myfile.txt

```
This is sonoo jaiswal
```

Types of Files in a C Program

When referring to file handling, we refer to files in the form of data files. Now, these data files are available in 2 distinct forms in the C language, namely:

- Text Files
- Binary Files

Text Files

The text files are the most basic/simplest types of files that a user can create in a C program. We create the text files using an extension *.txt* with the help of a simple text editor. In general, we can use notepads for the creation of .txt files. These files store info internally in ASCII character format, but when we open these files, the content/text opens in a human-readable form.

Text files are, thus, very easy to access as well as use. But there's one major disadvantage; it lacks security. Since a .txt file can be accessed easily, information isn't very secure in it. Added to this, text files consume a very large space in storage.

To solve these problems, we have a different type of file in C programs, known as binary files.

Binary Files

The binary files store info and data in the binary format of 0's and 1's (the binary number system). Thus, the files occupy comparatively lesser space in the storage. In simpler words, the binary files store data and info the same way a computer holds the info in its memory. Thus, it can be accessed very easily as compared to a text file.

The binary files are created with the extension .bin in a program, and it overcomes the drawback of the text files in a program since humans can't read it; only machines can. Thus, the information becomes much more secure. Thus, binary files are safest in terms of storing data files in a C program.

fread() fwrite() functions in c:

The fread() function reads the entire record at a time.

Syntax:

fread(& structure variable, size of (structure variable), no of records, file pointer);

Example:

```
struct emp{
  int eno;
  char ename [30];
  float sal;
} e;
FILE *fp;
fread (&e, sizeof (e), 1, fp);
```

The fwrite() function writes an entire record at a time.

Syntax:

fwrite(& structure variable , size of structure variable, no of records, file pointer);

Example:

```
struct emp{
  int eno:
  char ename [30];
  float sal;
} e;
FILE *fp;
fwrite (&e, sizeof(e), 1, fp);
```

Write a C program for storing the details of 3 students into a file and print the same using fread() and fwrite()

```
#include <stdio.h>
int main()
{ struct student
{int rn;
```

```
char name[10];
}s[3];
  FILE *fp;
  int i;

  fp = fopen ("student1.txt", "w");
   for (i=0; i<3; i++){
      printf ("enter details of student %d", i+1);
      printf("student number:");
      scanf("%d",&s[i].rn);
    printf("student name:");
      scanf("%s",s[i].name);
      fwrite(&s[i], sizeof(s[i]),3,fp);
   }
   fclose (fp);

fp = fopen ("student1.txt", "r");
   for (i=0; i<3; i++){
      printf ("details of student %d are", i+1);
      fread (&s[i], sizeof (s[i]) ,3,fp);
      printf("student number = %d", s[i]. rn);
      printf("student name = %s", s[i]. name);
   }
   fclose(fp);
    return 0;
}
```