UNIT-V

Structures: Basics of Structures, Nested Structures, Arrays of Structures, Arrays within structures, Structures and functions, pointers and structures, Self-referential structures, Unions**.**

Files: Introduction, Types of Files, File Access Functions, I/O on Files, Random Access to Files, Error Handling. , Command Line Arguments.

1. Define a structure? Write the syntax for structure declaration with an example.

Ans:

Definition: A structure is a collection of one or more variables of different data types, grouped together under a single name. By using structures variables, arrays, pointers etc can be grouped together.

Structures can be declared using two methods as follows:

(i) Tagged Structure:

The structure definition associated with the structure name is referred as tagged structure. It doesn't create an instance of a structure and does not allocate any memory.

The **general form or syntax of tagged structure** definition is as follows,

```
struct TAG
{
Type varaible1;
Type variable2;
……
……
Type variable-n;
};
```

Ex:-

```
struct student
{
    int htno[10];
    char name[20];
    float marks[6];
};
```

Where,

- struct is the keyword which tells the compiler that a structure is being defined.
- Tag_name is the name of the structure.
- variable1, variable2 … are called members of the structure.
- The members are declared within curly braces.
- The closing brace must end with the semicolon.

(ii) Type-defined structures:-

- The structure definition associated with the keyword **typedef** is called type-defined structure.

- This is the most powerful way of defining the structure.

The **syntax of typedefined structure** is

```
typedef struct
{
Type varaible1;
Type variable2;
……
……
Type variable-n;
}Type;
```

Ex:-

```
typedef struct
{
int htno[10];
    char name[20];
    float marks[6];
}student;
```

where

- typedef is keyword added to the beginning of the definition.
- struct is the keyword which tells the compiler that a structure is being defined.
- variable1, variable2…are called fields of the structure.
- The closing brace must end with type definition name which in turn ends with semicolon.

Variable declaration:

Memory is not reserved for the structure definition since no variables are associated with the structure definition. The members of the structure do not occupy any memory until they are associated with the structure variables.

After defining the structure, variables can be defined as follows:

For first method,

```
struct TAG v1,v2,v3….vn;
```

For second method, which is most powerful is,

```
Type v1,v2,v3,….vn;
```

Alternate way:

```
struct TAG
{
```

```
Type varaible1;

Type variable2;

  ……

  ……

Type variable-n;

} v1, v2, v3;
```

Ex:

```
struct book

{

char name[30];

int pages;

float price;

}b1,b2,b3;
```

2. **Declare the C structures for the following scenario:**

(i) College contains the following fields: College code (2characters), College Name, year of establishment, number of courses.

(ii) Each course is associated with course name (String), duration, number of students. (A College can offer 1 to 50 such courses)

Ans:

(i). Structure definition for college :-

```
struct college

{

        char code[2];

        char college_name[20];

        int year;

        int no_of_courses;

};
```

Variable declaration for structure college :-

```
void main( )
{
struct college col1,col2,col3;
….
}
```

(ii). Structure definition for course :-

```
struct course
{
        char  course_name[20];
        float duration;
        int no_of_students;
};
```

Variable declaration for structure course :-

```
void main( )
{
struct course c1,c2,c3;
….
}
```

3. How to initialize structures in 'C'? Write example.

Ans:

The rules for structure initialization are similar to the rules for array initialization. The initializers are enclosed in braces and separated by commas. They must match their corresponding types in the structure definition.

The syntax is shown below,

struct tag_name variable = {value1, value2,… value-n};

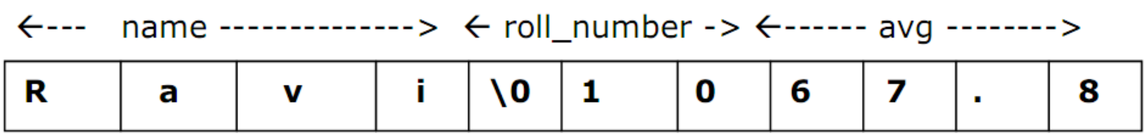
Structure initialization can be done in any one of the following ways

(i) Initialization along with Structure definition:-

Consider the structure definition for student with three fields name, roll number and average marks. The initialization of variable can be done as shown below,

```
struct student
{
char name [5];
int roll_number;
float avg;
} s1= {"Ravi", 10, 67.8};
```

The various members of the structure have the following values.



←--- name -------------->					← roll_number ->		←------ avg -------->			
R	a	v	i	\0	1	0	6	7	.	8

Figure 5.2 Initial Value of S1

(ii) Initialization during Structure declaration:-

Consider the structure definition for student with three fields name, roll number and average marks. The initialization of variable can be done as shown below,

```
typedef struct
{
int x;
int y;
float t;
char u;
} SAMPLE;
```

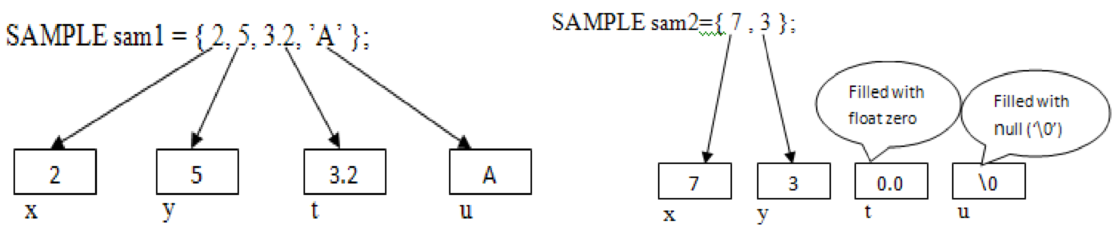


Figure Initializing Structures

The figure shows two examples of structure in sequence. The first example demonstrates what happens when not all fields are initialized. As we saw with arrays, when one or more initializers are missing, the structure elements will be assigned null values, zero for integers and floating-point numbers, and null ('\0') for characters and strings.

4. **Define a structure type *personal*, that would contain person name, date of joining and salary. Write a program to initialize one person data and display the same.**

Ans:

```
struct personal

{

        char name[20];

        int day;

        char month[10];

        int year;

        float salary;

};

void main( )

{

struct personal person = { "RAMU", 10 JUNE 1998 20000};

printf("Output values are:\n");

printf("%s%d%s%d%f",person.name,person.day,person.month,person.year,person.salary );

}
```

Output:- RAMU 10 JUNE 1998 20000

5. **How to access the data for structure variables using member operator('.')?Explain with an example.**

Ans:

We know that variables can be accessed and manipulated using expressions and operators. On the similar lines, the structure members can be accessed and manipulated. The members of a structure can be accessed by using dot(.) operator.

dot (.) operator

Structures use a **dot (.) operator**(also called **period operator** or **member operator**) to refer its elements. Before dot, there must always be a structure variable. After the dot, there must always be a structure element.

The **syntax** to access the structure members as follows,

structure_variable_name **.** structure_member_name

Consider the example as shown below,

```
struct student

{

char name [5];

int roll_number;

float avg;

};

struct student s1= {“Ravi”, 10, 67.8};
```

The members can be accessed using the variables as shown below,

```
s1.name --> refers the string “ravi”

s1.roll_number --> refers the roll_number 10

s1.avg     --> refers avg 67.8
```

6. Define a structure type *book*, that would contain book name, author, pages and price. Write a program to read this data using member operator (‘.’) and display the same.

Ans:

```
struct book

{

        char name[20];

        int day;

        char month[10];

        int year;

        float salary;

};
```

```
void main( )

{

        struct book b1;

        printf("Input values are:\n");

        scanf("%s %s %d %f", b1.title, b1.author,b1.pages,b1.price);

        printf("Output values are:\n");

        printf("%s\n %s\n %d\n %f\n", b1.title, b1.author,b1.pages,b1.price);

}
```

Output:-

```
Input values are:   C& DATA STRUCTURES    KAMTHANE      609      350.00

Output values are:

   C& DATA STRUCTURES

   KAMTHANE

   609

   350.00
```

7. How to pass a structure member as an argument of a function? Write a program to explain it.

Ans:

Structures are more useful if we are able to pass them to functions and return them.

By passing individual members of structure

This method is to pass each member of the structure as an actual argument of the function call. The actual arguments are treated independently like ordinary variables. This is the most elementary method and becomes unmanageable and inefficient when the structure size is large.

Program:

```
#include<stdio.h>
main ( )
{
float arriers (char *s, int n, float m);
typedef  struct emp
{
```

```
char name[15];
int emp_no;
float salary;
}record;
record e1 = {"smith",2,20000.25};
e1.salary = arriers(e1.name,e1.emp_no,e1.salary);
}
float arriers(char *s, int n, float m)
{
m = m + 2000;
printf("\n%s %d %f ",s, n, m);
return m;
}
```

Output

```
smith 2      22000.250000
```

8. How to pass an entire structure as an argument of a function? (or)

Write a program to pass entire structure as an argument of a structure.

Ans:

Structures are more useful if we are able to pass them to functions and return them.

Passing Whole Structure

This method involves passing a copy of the entire structure to the called function. Any changes to structure members within the function are not reflected in the original structure. It is therefore, necessary for the function to return the entire structure back to the calling function.

The general format of sending a copy of a structure to the called function is:

```
return_type  function_name (structure_variable_name);
```

The called function takes the following form:

```
data_type  function_name(struct_typetag_name)

{

………

………

return(expression);

}
```

The called function must be declared for its type, appropriate to the data type it is expected to return.

The structure variable used as the actual argument and the corresponding formal argument in the called function must be of the same struct type.

The return statement is necessary only when the function is returning some data back to the calling function. The expression may be any simple variable or structure variable or an expression using simple variables.

When a function returns a structure, it must be assigned to a structure of identical type in the calling function. The called functions must be declared in the calling function appropriately.

Program:

```
#include <stdio.h>

#include <string.h>

struct student

{

        int id;

        char name[20];

        float percentage;

};

void func(struct student record);

int main()

{

struct student record;

 record.id=1;

strcpy(record.name, "Raju");

record.percentage = 86.5;

return 0;

}

void func(struct student record)

{
        printf(" Id is: %d \n", record.id);

        printf(" Name is: %s \n", record.name);

        printf(" Percentage is: %f \n", record.percentage);

}
```

Output:

```
Id is: 1

Name is: Raju

Percentage is: 86.500000
```

9. Write a program for illustrating a function returning a structure.

Ans:

```
typedef  struct
{
char name [15];
int emp_no;
float salary;
} record;
#include<stdio.h.>
#include<string.h>
void main ( )
{
record change (record);
record e1 = {"Smith", 2, 20000.25};
printf ("\nBefore Change %s %d %f ",e1.name,e1.emp_no,e1.salary);
e1 = change(e1);
printf ("\nAfter Change %s %d %f ",e1.name,e1.emp_no,e1.salary);
}
record change (record e2)
{
strcpy (e2.name, "Jones");
e2.emp_no = 16;
e2.salary = 9999;
return e2;
}
```

Output:

```
Smith 2 20000.25
Jones 16 9999.99
```

10. What is an array of structure? Declare a variable as array of structure and initialize it? (or)When the array of structures is used? Write syntax for array of structure.

Ans:

An array is a collection of elements of same data type that are stored in contiguous memory locations. A structure is a collection of members of different data types stored in contiguous memory locations. An array of structures is an array in which each element is a structure. This concept is very helpful in representing multiple records of a file, where each record is a collection of dissimilar data items.

As we have an array of integers, we can have an array of structures also. For example, suppose we want to store the information of class of students, consisting of name, roll_number and marks, A better approach would be to use an array of structures.

Array of structures can be declared as follows,

```
struct  tag_name  arrayofstructure[size];
```

Let’s take an example, to store the information of 3 students, we can have the following structure definition and declaration,

```
struct student
{
char name[10];
int rno;
float avg;
};
struct student  s[3];
```

Defines an array called s, which contains three elements. Each element is defined to be of type struct student.

For the student details, array of structures can be initialized as follows,

```
        struct student s[3]={{“ABC”,1,56.7},{“xyz”,2,65.8},{“pqr”,3,82.4}};
```

Ex: An array of structures for structure employee can be declared as

```
                struct employee emp[5];
```

Let’s take an example, to store the information of 5 employees, we can have the following structure definition and declaration,

```
struct employee
{
int empid;
char name[10];
float salary;
};
struct employee  emp[5];
```

Defines array called emp, which contains five elements. Each element is defined to be of type struct employee.

For the employee details, array of structures can be initialized as follows,

```
struct employee emp[5] = {{1,”ramu”25,20000},
        {2,”ravi”,65000},
        {3,”tarun”,82000},
        {4,”rupa”,5000},
        {5,”deepa”,27000}};
```

11. Write a C program to calculate student-wise total marks for three students using array of structure.

Ans:

```
#include<stdio.h>

struct student

{

char rollno[10];

char name[20];

float sub[3];

float total;

};

void main( )

{

int i, j, total = 0;

struct student s[3];

printf("\t\t\t\t Enter 3 students details");

for(i=0; i<3; i++)

{

printf("\n Enter Roll number of %d Student:",i);
gets(s[i].rollno);
printf(" Enter the name:");
gets(s[i].name);
printf(" Enter 3 subjects marks of each student:");
total=0;
for(j=0; j<3; j++)
{
scanf("%d",&s[i].sub[j]);
total = total + s[i].sub[j];
}
}
printf("\n******************************************");
printf("\n\t\t\t Student details:");
printf("\n******************************************");
for(i=0; i<n; i++)
{
printf ("\n Student %d:",i+1);
printf ("\n Roll number:%s\n  Name:%s",s[i].rollno,s[i].name);
printf ("\nTotal marks =%f", s[i].total);
}
}
```

12. Write a C program using array of structure to create employee records with the following fields: emp-id, name, designation, address, salary and display it.

Ans:

```
#include<stdio.h>
struct employee
{
```

```
int emp_id;
char name[20];
char designation[10];
char address[20];
float salary;
}emp[3];
void main( )
{
int i;
printf("\t\t\t\t Enter 3 employees details");
for(i=0; i<3; i++)
{
scanf("%d",&emp[i].emp_id);
gets(emp[i].name);
gets(emp[i].designation);
gets(emp[i].address);
scanf("%f",&emp[i].salary);
}
printf("\n*******************************************");
printf("\n\t\t\t Employee details:");
printf("\n*******************************************");
for(i=0; i<3; i++)
{
printf("%d",emp[i].emp_id);
puts(emp[i].name);
puts(emp[i].designation);
puts(emp[i].address);
printf("%f",emp[i].salary);
}
}
```

13. What is structure within structure? Give an example for it.(or)

Write a C program to illustrate the concept of structure within structure(or) Explain about nested structures.

Ans:

Nested Structure (Structure within structure)

A structure which includes another structure is called nested structure or structure within structure. i.e a structure can be used as a member of another structure. There are two methods for declaration of nested structures.

(i) The syntax for the nesting of the structure is as follows

```
struct tag_name1
{
type1 member1;
 …….
…….
};
struct tag_name2
{
type1 member1;
……
……
struct tag_name1 var;
……
};
```

The syntax for accessing members of a nested structure as follows,

outer_structure_variable . inner_structure_variable.member_name

(ii) The syntax of another method for the nesting of the structure is as follows

```
struct structure_nm

{
        <data-type> element 1;
        <data-type> element 2;
        - - - - - - - - - - -
        - - - - - - - - - - -
        <data-type> element n;

        struct structure_nm
        {
                <data-type> element 1;
                <data-type> element 2;
                - - - - - - - - - - -
                - - - - - - - - - - -
                <data-type> element n;
        }inner_struct_var;
}outer_struct_var;
```

Example :

```
struct stud_Res
{
        int rno;
        char nm[50];
        char std[10];
        struct stud_subj
        {
                char subjnm[30];
                int marks;
        }subj;
}result;
```

In above example, the structure stud_Res consists of stud_subj which itself is a structure with two members. Structure stud_Res is called as 'outer structure' while stud_subj is called as 'inner structure.'

The members which are inside the inner structure can be accessed as follow :

```
result.subj.subjnm
result.subj.marks
```

Program to demonstrate nested structures.

```
#include <stdio.h>
#include <conio.h>
struct stud_Res
{
```

```
            int rno;
            char std[10];
            struct stud_Marks
            {
                        char subj_nm[30];
                        int subj_mark;
            }marks;
}result;
void main()
{
            printf("\n\t Enter Roll Number : ");
            scanf("%d",&result.rno);
            printf("\n\t Enter Standard : ");
            scanf("%s",result.std);
            printf("\n\t Enter Subject Code : ");
            scanf("%s",result.marks.subj_nm);
            printf("\n\t Enter Marks : ");
            scanf("%d",&result.marks.subj_mark);
            printf("\n\n\t Roll Number : %d",result.rno);
            printf("\n\n\t Standard : %s",result.std);
            printf("\nSubject Code : %s",result.marks.subj_nm);
            printf("\n\n\t Marks : %d",result.marks.subj_mark);
}
```

Output:

```
Enter roll number : 1
Enter standard :Btech
Enter subject code : GR11002
Enter marks : 63
Roll number :1
Standard :Btech
Subject code : GR11002
Marks : 63
```

14. Write a C program using nested structures to read 3 employees details with the following fields; emp-id, name, designation, address, da ,hra and calculate gross salary of each employee.

Ans:

```
#include<stdio.h>

struct employee

{
```

```
int emp_id;
char name[20];
char designation[10];
char address[20];
struct salary
{
float da;
float hra;
}sal;
}emp[3];
void main( )
{
int i;
printf("\t\t\t\t Enter 3 employees details");
grosssalary = 0;
for(i=0; i<3; i++)
{
scanf("%d",&emp[i].emp_id);
gets(emp[i].name);
gets(emp[i].designation);
gets(emp[i].address);
scanf("%f",&emp[i].sal.da);
scanf("%f",&emp[i].sal.hra);
grosssalary = grosssalary + emp[i].sal.da + emp[i].sal.hra;
}
printf("\n********************************************");
printf("\n\t\t\t Employee details with gross salary:");
printf("\n*******************************************");
for(i=0; i<3; i++)
{
printf("%d",emp[i].emp_id);
puts(emp[i].name);
puts(emp[i].designation);
puts(emp[i].address);
printf("%f",emp[i].sal.da);
printf("%f",emp[i].sal.hra);
printf("%f",grosssalary);
}
}
```

15. Distinguish between Arrays within Structures and Array of Structure with examples.

Ans:

Arrays within structure

It is also possible to declare an array as a member of structure, like declaring ordinary variables. For example to store marks of a student in three subjects then we can have the following definition of a structure.

```
struct student
{
char name [5];
int roll_number;
int marks [3];
float avg;
};
```

Then the initialization of the array marks done as follows,

```
struct student s1= {"ravi", 34, {60,70,80}};
```

The values of the member marks array are referred as follows, s1.marks [0] --> will refer the 0th element in the marks

s1.marks [1] --> will refer the 1st element in the marks

s1.marks [2] --> will refer the 2ndt element in the marks

Array of structure

An array is a collection of elements of same data type that are stored in contiguous memory locations. A structure is a collection of members of different data types stored in contiguous memory locations. An array of structures is an array in which each element is a structure. This concept is very helpful in representing multiple records of a file, where each record is a collection of dissimilar data items.

Ex:

An array of structures for structure *employee* can be declared as

```
        struct employee emp[5];
```

Let's take an example, to store the information of 5 employees, we can have the following structure definition and declaration,

```
struct employee
{
int empid;
char name[10];
float salary;
};
struct employee  emp[5];
```

Defines array called emp, which contains five elements. Each element is defined to be of type struct student.

For the student details, array of structures can be initialized as follows,

```
struct employee emp[5] = {{1,”ramu”25,20000},
      {2,”ravi”,65000},
      {3,”tarun”,82000},
      {4,”rupa”,5000},
      {5,”deepa”,27000}};
```

16. Write a C program using structure to create a library catalogue with the following fields: Access number, author's name, Title of the book, year of publication, publisher's name, and price.

Ans:

```
struct library

{       int acc_no;

        char author[20];
        char title[10];

        int year_pub;

        char name_pub[20];

        float price;

};

void main( )

{

      struct  library  lib;

      printf("Input values are:\n");

      scanf("%d  %s  %s  %d  %s%f", &lib.acc_no,   lib.author,  lib.title,  &lib.year_pub,
      lib.name_pub, &lib.price);

      printf("Output values are:\n\n");

      printf("Access number = %d\n Author name = %s\n

      Book Title = %s\n  Year of publication =  %d \n Name of publication = %s\n

       Price = %f ", lib.acc_no,  lib.author,  lib.title, lib.year_pub,  lib.name_pub,  lib.price);

}
```

17. Explain in brief about pointers and structures.

Ans: **Pointers to structures**:

We have pointers pointing to int, float, arrays etc., We also have pointers pointing to structures. They are called as structure pointers.

To access the members of a structure using pointers we need to perform

The following operations:

- Declare the structure variable
- Declare a pointer to a structure
- Assign address of structure variable to pointer
- Access members of structure using (.) operator or using (->)member selection operator or arrow operator.

Syntax:

struct tagname

```
{
datatype member1;
datatype member2;
…
};
struct taganme var;
struct tagname *ptr;
ptr=*var;
```

to access the members

(*ptr).member1; or ptr->member1;

The parentheses around *ptr are necessary because the member operator '.' has a higher precedence than the operator '*'

Example:

```
struct student
{
int rno;
char name[20];
};
struct student s1;
struct student *ptr;
ptr=&s1;
to access
(*ptr).rno;
(*ptr).name;
(or)
ptr->rno;
ptr->name;
```

Program to read and display student details using pointers to structures

```
struct student
{
  int HTNO;
char NAME[20];
float AVG;
};
void main()
{
 struct student s1;
struct student *ptr;
ptr=&s1;
printf("Enter student details");
scanf("%d%s%f",&ptr->HTNO,ptr->NAME,&ptr->AVG);
printf("HTNO=%d",ptr->HTNO);
printf("NAME=%s",ptr->NAME);
printf("AVERAGE MARKS=%f",ptr->AVG);
}
```

18. What is self- referential structure? Explain through example.

Ans:

Self-referential structure

A structure definition which includes at least one member as a pointer to the same structure is known as self-referential structure.

It can be linked together to form useful data structures such as lists, queues, stacks and trees.

It is terminated with a NULL pointer .

The syntax for using the self referential structure is as follows,

```
struct tag_name
{
```

```
Type1 member1;

Type2 member2;

……….

struct tag_name *next;

};
```

Ex:-

```
struct node

{

int data;

struct node *next;

} n1, n2;
```

19. Explain unions in C language? Differentiate structures and unions.

Ans:

A union is one of the derived data types. Union is a collection of variables referred under a single name. The syntax, declaration and use of union is similar to the structure but its functionality is different.

The general format or syntax of a union definition is as follows,

Syntax:

```
union union_name

{

        <data-type> element 1;

        <data-type> element 2;

        ………………

}union_variable;
```

Example:

union techno

{

int comp_id;

char nm;

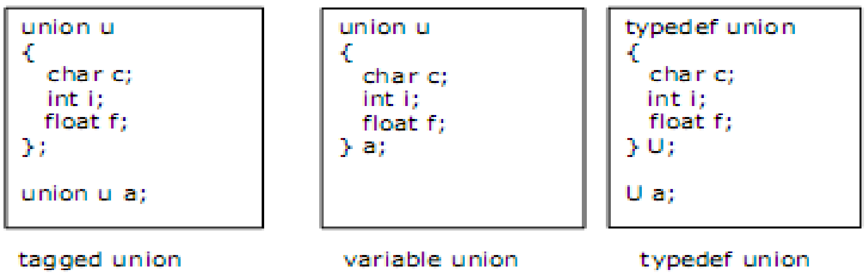
float sal;

}tch;

A union variable can be declared in the same way as structure variable.

union tag_name var1, var2...;

A union definition and variable declaration can be done by using any one of the following



```
union u
{
   char c;
   int i;
   float f;
};

union u a;
```

tagged union

```
union u
{
   char c;
   int i;
   float f;
} a;
```

variable union

```
typedef union
{
   char c;
   int i;
   float f;
} U;

U a;
```

typedef union

Figure 5.7 Types of Union Definitions

We can access various members of the union as mentioned: a.c   a.i a.f

and memory organization is shown below,

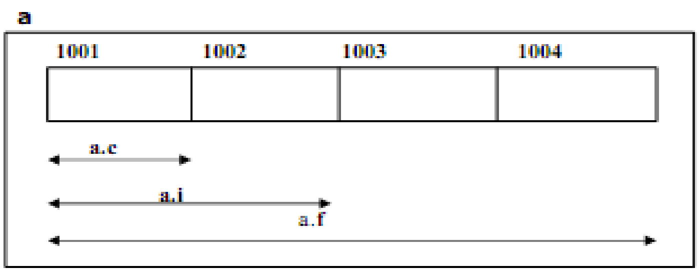


Figure 5.8: Memory Organization Union

In the above declaration, the member **f** requires 4 bytes which is the largest among all the members. Figure 5.8 shows how all the three variables share the same address. The size of the union here is 4 bytes.

A union creates a storage location that can be used by any one of its members at a time. When a different member is assigned a new value, the new value supersedes the previous members' value.

Difference between structure and union:-

	Structure	**Union**
(i) Keyword	Struct	Union
(ii) Definition	A structure is a collection of logically related elements, possibly of different types, having a single name.	A union is a collection of logically related elements, possibly of different types, having a single name, shares single memory location.

(iii) Declaration	struct tag_name
	{
	type1 member1;
	type1 member2;
	……………
	};
	struct tag_name var;
		union tag_name
		{
		type1 member1;
		type1 member2;
		……………
		};
		union tag_name var;

(iv) Initialization	Same.	Same.
(v) Accessing	Accessed by specifying structure_ variable_name.member_ name	Accessed by specifying union_ variable_name.member_name
(vi)Memory Allocation	Each member of the structure occupies unique location, stored in contiguous locations.	Memory is allocated by considering the size of the largest member. All the members

		share the common location
(vii) Size	Size of the structure depends on the type of members, adding size of all the members. sizeof (st_var);	Size is given by the size of the largest member sizeof(un_variable)
(viii) Using pointers	Structure members can be accessed by using dereferencing operator dot and selection operator(->)	same as structure.
	We can have arrays as a member of structures. All members can be accessed at a time.	We can have array as a member of union. Only one member can be accessed at a time.
	Nesting of structures is possible.	same.
	It is possible structure may contain union as a member.	It is possible union may contain structure as a member

20. Define file and explain about the types of files with examples.

Ans :

- ❖ A file is an external collection of related data treated as a unit. A file is a place on a disk where a group of related data is stored and retrieved whenever necessary without destroying data
- ❖ The primary purpose of a file is to keep record of data. Record is a group of related fields. Field is a group of characters which convey meaning.
- ❖ Files are stored in auxiliary or secondary storage devices. The two common forms of secondary storage are disks (hard disk, CD and DVD) and tapes.
- ❖ Each file ends with an end of file (EOF) at a specified byte number, recorded in file structure.

❖ A file must first be opened properly before it can be accessed for reading or writing. When a file is opened an object (buffer) is created and a stream is associated with the object.

It is advantageous to use files in the following circumstances.

1. When large volume of data are handled by the program and
2. When the data need to be stored permanently without getting destroyed when program is terminated.

There are two kinds of files depending upon the format in which data is stored:

1) Text files 2) Binary files

1) Text files:

A text file stores textual information like alphabets, numbers, special symbols, etc. actually the ASCII code of textual characters its stored in text files. Examples of some text files include c, java, c++ source code files and files with .txt extensions . The text file contains the characters in sequence. The computer process the text files sequentially and in forward direction. One can perform file reading, writing and appending operations. These operations are performed with the help of inbuilt functions of c.

2) Binary files:

Text mode is inefficient for storing large amount of numerical data because it occupies large space. Only solution to this is to open a file in binary mode, which takes lesser space than the text mode. These files contain the set of bytes which stores the information in binary form. One main drawback of binary files is data is stored in human unreadable form. Examples of binary files are .exe files, video stream files, image files etc. C language supports binary file operations with the help of various inbuilt functions ..

21. Explain different modes of opening files with syntax and example?

Ans:

To store data in a file three things have to be specified for operating system.

They include

1. FILENAME :

It is a string of characters that make up a valid file name which may contain two parts,a primary name and an optional period with the extension.

Prog1.c

Myfirst.java

Data.txt

store

2. DATA STRUCTURE:

It is defined as FILE in the library of standard I/O function definitions. Therefore all files are declared as type FILE.FILE is a define data type.

FILE *fp;

3. PURPOSE:

It defines the reason for which a file is opened and the mode does this job.

fp=fopen(“filename”,”mode”);

The different modes of opening files are :

“r” (read) mode:

The read mode (r) opens an existing file for reading. When a file is opened in this mode, the file marker or pointer is positioned at the beginning of the file (first character).The file must already exist: if it does not exist a NULL is returned as an error. If we try to write a file opened in read mode, an error occurs.

Syntax: **fp=fopen (“filename”,”r”);**

“w” (write) mode

The write mode (w) opens a file for writing. If the file doesn’t exist, it is created. If it already exists, it is opened and all its data is erased; the file pointer is positioned at the beginning of the file It gives an error if we try to read from a file opened in write mode.

Syntax: **fp=fopen (“filename”,”w”);**

“a” (append) mode

The append mode (a) opens an existing file for writing instead of creating a new file. However, the writing starts after the last character in the existing file ,that is new data is added, or appended, at the end of the file. If the file doesn’t exist, new file is created and opened. In this case, the writing will start at the beginning of the file.

Syntax: **fp=fopen (“filename”,”a”);**

“r+” (read and write) mode

In this mode file is opened for both reading and writing the data. If a file does not exist then NULL, is returned.

Syntax: **fp=fopen ("filename","r+");**

"w+" (read and write) mode

In this mode file is opened for both writing and reading the data. If a file already exists its contents are erased and if a file does not exist then a new file is created.

Syntax: **fp=fopen ("filename","w+");**

"a+" (append and read) mode

In this mode file is opened for reading the data as well as data can be added at the end.

Syntax: **fp=fopen ("filename", "a+");**

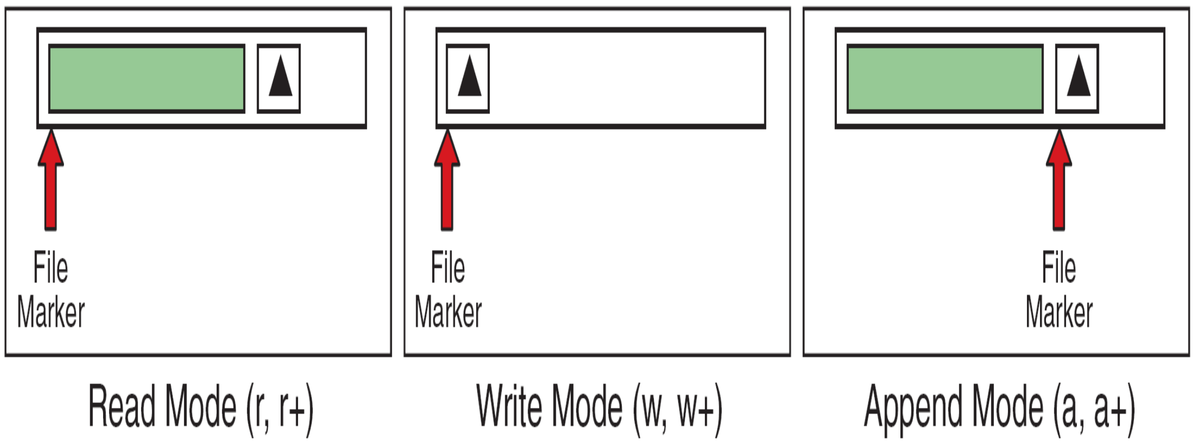


Figure : File Opening Modes

NOTE: To perform operations on binary files the following modes are applicable with an extension b like rb,wb,ab,r+b,w+b,a+b,which has the same menaing but allows to perform operations on binary files.

22. Write about the basic operations on files?

Ans:

Basic operations on file:

1. Naming a file
2. Opening a file
3. Reading data from file
4. Writing data into file
5. Closing a File

In order to perform the basic file operations C supports a number of functions .Some of the important file handling functions available in the C library are as follows :

FUNCTION NAME	**OPERATION**
fopen()	Creates a new file for use Opens an existing file for use
fclose()	Closes a file which has been opened for use
fcloseall()	Closes all files which are opened
getc()/fgetc()	Reads a character from a file
putc()/fputc()	Writes a character to a file
fprintf()	Writes a set of data values to files
fscanf()	Reads a set of data values from files
getw()	Reads an integer from file
putw()	Writes an integer to a file
gets()	Reads a string from a file
puts()	Writes a string to a file
fseek()	Sets the position to a desired point in afile
ftell()	Gives the current position in the file
rewind()	Sets the position to the beginning of the file

Naming and opening a file:

A name is given to the file used to store data. The file name is a string of characters that make up a valid file name for operating system. It contains two parts. A primary name and an optional period with the extension.

Examples: Student.dat, file1.txt, marks.doc, palin.c.

The general format of declaring and opening a file is

```
FILE *fp;                       //declaration
fp=fopen (“filename”,”mode”);     //statement to open file.
```

Here FILE is a data structure defined for files. fp is a pointer to data type FILE. filename is the name of the file. mode tells the purpose of opening this file.

Reading data from file :

Input functions used are (Input operations on files)

a) getc(); It is used to read characters from file that has been opened for read operation.

Syntax: c=getc (file pointer)

This statement reads a character from file pointed to by file pointer and assign to c.

It returns an end-of-file marker EOF, when end of file has been reached

b) fscanf();

This function is similar to that of scanf function except that it works on files.

Syntax: fscanf (fp, "control string", list);

Example fscanf(f1,"%s%d",str,&num);

The above statement reads string type data and integer type data from file.

c) **getw();** This function reads an integer from file.

Syntax: getw (file pointer);

d)**fgets():** This function reads a string from a file pointed by a file pointer. It also copies the string to a memory location referred by an array.

Syntax: fgets(string,no of bytes,filepointer);

e)**fread():** This function is used for reading an entire structure block from a given file.

Syntax: fread(&struct_name,sizeof(struct_name),1,filepointer);

Writing data to a file:

To write into a file, following C functions are used

a. **putc():**This function writes a character to a file that has been opened in write mode.

Syntax: putc(c,fp);

This statement writes the character contained in character variable c into a file whose pointer is fp.

b. **fprintf():**This function performs function, similar to that of printf.

Syntax: fprintf(f1,"%s,%d",str,num);

c. putw(): It writes an integer to a file.

Syntax: putw (variable, fp);

d. **fputs():** This function writes a string into a file pointed by a file pointer.

Synatax: fputs(string, filepointer);

e. **fwrite():** This function is used for writing an entire block structure to a given file.

Syntax: fwrite(&struct_name, sizeof(struct_name),1,filepointer);

Closing a file:

A file is closed as soon as all operations on it have been completed.Closing a file ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken. Another instance where we have to close a file is to reopen the same file in a different mode. Library function for closing a file is fclose(file pointer);

Example: fclose(fp);

Where fp is the file pointer returned by the call to fopen(). fclose() returns 0 on success (or) -1 on error. Once a file is closed, its file pointer can be reused for another file.

Note: fcloseall() can be used to close all the opened files at once.

23. What are the file I/O functions in C. Give a brief note about the task performed by each function.

Ans:

In order to perform the file operations in C we must use the high level I/O functions which are in C standard I/O library. They are

i) getc() and putc() functions:

getc()/fgetc() : It is used to read a character from a file that has been opened in a read mode. It reads a character from the file whose file pointer is fp. The file pointer moves by one character for every operation of getc(). The getc() will return an end-of –marker EOF, when an end of file has been reached.

Syntax: getc(fp);

Ex: char ch;

ch=getc(fp);

putc()/fputc() -: It is used to write a character contained in the character variable to the file associated with the FILE pointer fp. fputc() also returns an end-of –marker EOF, when an end of file has been reached**.**

Syntax: **putc(c,fp);**

Example: char c;

putc(c,fp);

Program using fgetc() and fputc():

```
#include<stdio.h>
void main()
{
FILE *fp;
char ch;
fp=fopen("input1.txt","w");
printf("\n enter some text hereand press cntrl D or Z to stop :\n");
while((ch=getchar())!=EOF)
fputc(ch,fp);
fclose(fp);
fp=fopen("input1.txt","r");
printf("\n  The entered text is : \n");
while((ch=fgetc(fp))!=EOF)
putchar(ch);
fclose(fp);
}
```

ii) fprintf() and fscanf():

In order to handle a group of mixed data simultaneously there are two functions that are fprintf() and fscanf().These two functions are identical to printf and scanf functions,except that they work on files. The first argument of these functions is a file pointer which specifies the file to be used.

fprintf(): The general form of fprintf() is

Syntax: **fprintf(fp,"control string",list);**

where fp is a file pointer associated with a file that has been opened for writing . The control string contains output specifications for the items in the list. .

Example: **fprintf(fp,"%s%d",name,age);**

fscanf() : It is used to read a number of values from a file.

Syntax: **fscanf(fp,"control string",list);**

Example: **fscanf(fp2,"%s %d",item,&quantity);**

like scanf , fscanf also returns the number of items that are successfully read. when the end of file is reached it returns the value EOF.

Program using fscanf() and fprintf():

```
#include<stdio.h>
void main()
{
int a=22,b;
char s1[20]="Welocme_to_c",s2[20];
float c=12.34,d;
FILE *f3;
f3=fopen("mynew3","w");
fprintf(f3,"%d %s %f",a,s1,c);
fclose(f3);
```

```
        f3=fopen("mynew3","r");
        fscanf(f3,"%d %s %f",&b,s2,&d);
        printf("\n a=%d \t s1=%s \t c=%f \n b=%d \t s2=%s \t d=%f",a,s1,c,b,s2,d);
        fclose(f3);
        }
```

iii) getw() and putw():

The getw() and putw()are integer oriented functions .They are similar to the getc() and putc() functions and are used to read and write integer values . These functions would be useful when we deal with only integer data. The general form of getw() and putw() are

Syntax: putw(integer,fp);

Syntax: getw(fp);

Program using getw() and putw():

```
/*Printing odd numbers in odd file and even numbers in even file*/
#include<stdio.h>
void main()
{
int x,i;
FILE *f1,*fo,*fe;//creating a file pointer
f1=fopen("anil.txt","w");   //opening a file
printf("\n enter some numbers into file or -1 to stop \n");
for(i=0;i<20;i++)
{
scanf("%d",&x);
if(x== -1)break;
putw(x,f1);  //writing read number into anil.txt file one at a time
}
fclose(f1);    //closing a file opened for  writing input
printf("\n OUTPUT DATA\n");
f1=fopen("anil.txt","r");//open anil in read mode to read data
fo=fopen("odd3f","w");
fe=fopen("even3f","w");
while((x=getw(f1))!=EOF)
{
printf("%d\t",x);
if(x%2==0)
putw(x,fe);
else
putw(x,fo);
}
fcloseall();
fo=fopen("odd3f","r");
printf("\n  contents of odd file are :\n");
while((x=getw(fo) )!= EOF)
printf(" %d\t",x);
fe=fopen("even3f","r");
printf("\n  contents of even  file are :\n");
while((x=getw(fe)) != EOF)
printf(" %d\t",x);
fcloseall();
}
```

iv) fputs() and fgets():

fgets(): It is used to read a string from a file pointed by file pointer. It copies the string to a memory location referred by an array.

Syntax: **fgets(string,length,filepointer);**

Example: **fgets(text,50,fp1);**

fputs(): It is used to write a string to an opened file pointed by file pointer.

Syntax: **fputs(string,filepointer);**

Example: **fputs(text,fp);**

Program using fgets() and fputs():

```
#include<stdio.h>
void main()
{
FILE *fp;
char str[50];
fp=fopen("fputget.txt","r");
printf("\n the read string is  :\n");
fgets(str,50,fp);
puts(str);
fclose(fp);
fp=fopen("fputget.txt","a+");
printf("\n  Enter string : \n");
gets(str);
fputs(str,fp);
puts(str);
fclose(fp);
}
```

Block or structures read and write:

Large amount of integers or float data require large space on disk in text mode and turns out to be inefficient .For this we prefer binary mode and the functions used are

v) fread() and fwrite():

fwrite(): It is used for writing an entire structure block to a given file in binary mode.

Syntax: **fwrite(&structure variable,sizeof(structure variable),1,filepointer);**

Example: **fwrite(&stud,sizeof(stud),1,fp);**

fread(): It is used for reading an entire structure block from a given file in binary mode.

Syntax: **fread(&structure variable,sizeof(structure variable),1,filepointer);**

Example: **fread(&emp,sizeof(emp),1,fp1);**

Program using fread() and fwrite():

```
#include<stdio.h>
struct player
{
char pname[30];
int age;
int runs;
};
void main()
{
struct player p1,p2;
FILE *f3;
f3=fopen("player.txt","w");
printf("\n Enter details of player name ,age and runs scored :\n ");
fflush(stdin);
scanf("%s %d %d",p1.pname,&p1.age,&p1.runs);
fwrite(&p1,sizeof(p1),1,f3);
fclose(f3);
f3=fopen("player.txt","r");
fread(&p2,sizeof(p2),1,f3);
fflush(stdout);
printf("\nPLAYERNAME:=%s\tAGE:=%d\tRUNS:=%d ",p2.pname,p2.age,p2.runs);
fclose(f3);
}
```

24. Explain about random access to files with example?

Ans:

At times we needed to access only a particular part of a file rather than accessing all the data sequentially, which can be achieved with the help of functions **fseek, ftell** and **rewind** available in IO library.

ftell():-

ftell takes a file pointer and returns a number of type **long,** that corresponds to the current position. This function is useful in saving the current position of the file, which can later be used in the program.

Syntax: n=**ftell(fp);**

n would give the Relative offset (In bytes) of the current position. This means that already **n** bytes have a been read or written

rewind():-

It takes a file pointer and resets the position to the start of the file.

Syntax: **rewind(fp);**

n=ftell(fp);

would assign 0 to **n** because the file position has been set to start of the file by rewind(). The first byte in the file is numbered 0, second as 1, so on. This function helps in reading the file more than once, without having to close and open the file.

Whenever a file is opened for reading or writing a rewind is done implicitly.

fseek:-

fseek function is used to move the file pointer to a desired location within the file.

Syntax: **fseek(file ptr,offset,position);**

file pointer is a pointer to the file concerned, offset is a number or variable of type long and position is an integer number which takes one of the following values. The offset specifies the number of positions(**Bytes**) to be moved from the location specified by the position which can be positive implies moving forward and negative implies moving backwards.

POSITION VALUE	VALUE CONSTANT	MEANING
0	SEEK_SET	BEGINNING OF FILE
1	SEEK_CUR	CURRENT POSITION
2	SEEK_END	END OF FILE

Example: **fseek(fp,10,0) ;**

fseek(fp,10,SEEK_SET);// file pointer is repositioned in the forward direction 10 bytes.

fseek(fp,-10,SEEK_END); // reads from backward direction from the end of file.

When the operation is successful fseek returns 0 and when we attempt to move a file beyond boundaries fseek returns -1.

Some of the Operations of fseek function are as follows:

STATEMENT	MEANING
fseek(fp,0L,0);	Go to beginning similar to rewind()
fseek(fp,0L,1);	Stay at current position
fseek(fp,0L,2);	Go to the end of file, past the last character of the file.
fseek(fp,m,0);	Move to (m+1)th byte in the file.
fseek(fp,m,1);	Go forward by m bytes
fseek(fp,-m,1);	Go backwards by m bytes from the current position
fseek(fp,-m,2);	Go backwards by m bytes from the end.(positions the file to the mth character from the end)

Program on random access to files:

```
#include<stdio.h>
void main()
{
FILE *fp;
char ch;
fp=fopen("my1.txt","r");
fseek(fp,21,SEEK_SET);
ch=fgetc(fp);
while(!feof(fp))
{
printf("%c\t",ch);
printf("%d\n",ftell(fp));
ch=fgetc(fp);
}
rewind(fp);
printf("%d\n",ftell(fp));
fclose(fp);
}
```

25. Explain about error handling in files?

Ans :

It is possible that an error may occur during I/O operations on a file. Typical error situations include:

1. Trying to read beyond the end of file mark.
2. Device overflow
3. Trying to use a file that has not been opened
4. Trying to perform an operation on a file, when the file is opened for another type of operations
5. Opening a file with an invalid filename
6. Attempting to write a write protected file

If we fail to check such read and write errors, a program may behave abnormally when an error occurs. An unchecked error may result in a premature termination of the program or incorrect output. In C we have two status - inquiry library functions **feof and ferror** that can help us detect I/O errors in the files.

a). feof(): The feof() function can be used to test for an end of file condition. It takes a FILE pointer as its only argument and returns a non zero integer value if all of the data from the specified file has been read, and returns zero otherwise. If fp is a pointer to file that has just opened for reading, then the statement

```
if(feof(fp))
printf("End of data");
```

would display the message “End of data” on reaching the end of file condition.

b). ferror(): The ferror() function reports the status of the file indicated. It also takes a file pointer as its argument and returns a nonzero integer if an error has been detected up to that point, during processing. It returns zero otherwise. The statement

```
if(ferror(fp)!=0)
printf(“an error has occurred\n”);
```

would print an error message if the reading is not successful

c). fp==null: We know that whenever a file is opened using fopen function, a file pointer is returned. If the file cannot be opened for some reason, then the function returns a null pointer. This facility can be used to test whether a file has been opened or not. Example

```
if(fp==NULL)
printf(“File could not be opened.\n”);
```

d) perror(): It is a standard library function which prints the error messages specified by the compiler. For example:

```
if(ferror(fp))
  perror(filename);
```

Program for error handling in files:

```
#include<stdio.h>
void main()
{
FILE *fp;
char ch;
fp=fopen("my1.txt","r");
if(fp==NULL)
printf(“\n file cannot be opened”);
while(!feof(fp))
{
ch=getc(fp);
if(ferror(fp))
perror(“problem in the file”);
else
putchar(ch);
}
fclose(fp);
}
```

26. Write a c program to read and display the contents of a file?

Ans: Program:

```
#include<stdio.h>
void main()
{
FILE *f1;
char ch;
f1=fopen("data.txt","w");
printf("\n enter some text here and press cntrl D or Z to stop :\n");
while((ch=getchar())!=EOF)
fputc(ch,f1);
fclose(f1);
printf("\n the contents of file  are \n:");
f1 = fopen("data.txt","r");
while( ( ch = fgetc(f1) ) != EOF )
putchar(ch);
fclose(f1);
}
```

27. Write a c program to copy the contents of one file to another?

Ans: Program:

```
#include<stdio.h>
void main()
{
FILE *f1,*f2;
char ch;
f1=fopen("mynew2.txt","w");
printf("\n enter some text here and press cntrl D or Z to stop :\n");
while((ch=getchar())!=EOF)
fputc(ch,f1);
fclose(f1);
f1=fopen("mynew2.txt","r");
f2=fopen("dupmynew2.txt","w");
while((ch=getc(f1))!=EOF)
putc(ch,f2);
fcloseall();
printf("\n the copied file contents are :");
f2 = fopen("dupmynew2.txt","r");
while( ( ch = fgetc(f2) ) != EOF )
putchar(ch);
fclose(f2);
}
```

28. Write a c program to merge two files into a third file? (Or)

Write a c program for the following .there are two input files named "first.dat" and "second.dat" .The files are to be merged. That is,copy the contents of first.dat and then second.dat to a new file named result.dat?

Ans: Program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
FILE *fs1, *fs2, *ft;
char ch, file1[20], file2[20], file3[20];
printf("Enter name of first file\n");
gets(file1);
printf("Enter name of second file\n");
gets(file2);
printf("Enter name of file which will store contents of two files\n");
gets(file3);
fs1=fopen(file1,"w");
printf("\n enter some text into file1 here and press cntrl D or Z to stop :\n");
while((ch=getchar())!=EOF)
fputc(ch,fs1);
fclose(fs1);
```

```
        fs2=fopen(file2,"w");

        printf("\n enter some text into file2 here and press cntrl D or Z to stop :\n");

        while((ch=getchar())!=EOF)

        fputc(ch,fs2);

        fclose(fs2);

        fs1 = fopen(file1,"r");
        fs2 = fopen(file2,"r");
        if( fs1 == NULL || fs2 == NULL )
        {
        perror("Error ");
        exit(1);
        }
        ft = fopen(file3,"w");
        while( ( ch = fgetc(fs1) ) != EOF )
        fputc(ch,ft);
        while( ( ch = fgetc(fs2) ) != EOF )
        fputc(ch,ft);
        printf("Two files were merged into %s file successfully.\n",file3);
        fcloseall();
        ft = fopen(file3,"r");
        printf("\n the merged file contents are :");
        while( ( ch = fgetc(fs1) ) != EOF )
        putchar(ch);
        fclose(ft);
        return 0;
        }
```

29. Write a c program to append the contents of a file ?

Ans: Program:

```
#include<stdio.h>

void main()

{

FILE *fp1;

char ch;

fp1=fopen("sunday.txt","w");

printf("\n Enter some Text into file :\n");

while((ch=getchar())!='.')

fputc(ch,fp1);
fclose(fp1);
fp1=fopen("sunday.txt","a+"); //to append
printf("\n Enter some  MORE Text into file :\n");
while((ch=getchar())!='.')
fputc(ch,fp1);
rewind(fp1);
printf("\n The complete Text in file  is :\n");
while((ch=fgetc(fp1))!=EOF)
putchar(ch);
fclose(fp1);
}
```

30. Write a c program to display the reverse the contents of a file?

Ans:

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
FILE *fp1;
char ch;
int x;
fp1=fopen("any1.txt","w");
printf("\n Enter some text into file :\n");
while((ch=getchar())!=EOF)
fputc(ch,fp1);
fclose(fp1);
fp1=fopen("any1.txt","r");
if(fp1==NULL)
{
printf("\n cannot open file:");
exit(1);
}
fseek(fp1,-1L,2);
x=ftell(fp1);
printf("\n  Th text in the  file in reverse order is : \n");
while(x>=0)
{
ch=fgetc(fp1);
printf("%c",ch);
x--;
fseek(fp1,x,0);
 }
fclose(fp1);
  }
```

31. Explain the command line arguments. What are the syntactic constructs followed in 'C'?

Ans :

Command line argument is the parameter supplied to a program when the program is invoked. This parameter may represent a file name the program should process. For example, if we want to execute a program to copy the contents of a file named X_FILE to another one name Y_FILE then we may use a command line like

```
C:> program X_FILE Y_FILE
```

Program is the file name where the executable code of the program is stored. This eliminates the need for the program to request the user to enter the file names during execution. The 'main' function can take two arguments called argc, argv and information contained in the command line is passed on to the program to these arguments, when 'main' is called up by the system.

The variable **argv** is an argument vector and represents an array of characters pointers that point to the command line arguments.

The **argc** is an argument counter that counts the number of arguments on the command line. The size of this array is equal to the value of argc. In order to access the command line arguments, we must declare the 'main' function and its parameters as follows:

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

}

Generally argv[0] represents the program name.

<u>Example:-</u> A program to copy the contents of one file into another using command line arguments.

```
#include<stdio.h>
#include<stdlib.h>
void main(int argc,char* argv[]) /* command line arguments */
 {
        FILE *ft,*fs; /* file pointers declaration*/
        int c=0;
        if(argc!=3)
        Printf("\n insufficient arguments");
        fs=fopen(argv[1],"r");
        ft=fopen(argv[2],"w");
        if (fs==NULL)
        {
        printf("\n source file opening error");
        exit(1) ;
        }
        if (ft==NULL)
        {
        printf("\n target file opening error");
        exit(1) ;
        }
         while(!feof(fs))
        {
        fputc(fgetc(fs),ft);
        c++;
        }
        printf("\n bytes copied from %s file to %s file =%d", argv[1], argv[2], c);
        c=fcloseall(); /*closing all files*/
        printf("files closed=%d",c);
}
```

32. Write a c program to add two numbers using command line arguments?

Ans:

Program:

```
#include<stdio.h>
int main(int argc, char *argv[])
{
int x, sum=0;
printf("\n Number of arguments are:%d", argc);
printf("\n The agruments are:");
for(x=0;x<argc; x++)
 {
printf("\n agrv[%d]=%s", x, argv[x]);
if(x<2)
continue;
else
sum=sum+atoi(argv[x]);
  }
printf("\n program name:%s",argv[0]);
printf("\n name is:%s",argv[1]);
printf("\n sum is:%d",sum);
return(0);
 }
```