Module 4

Structures And File Management

4.1 Introduction

- Derived data type : int,float,char,double, are the primitive data types .
- Using these primitive data types we can derive other data types and such data types derived from basic types are called **derived data types.**

4.2 Type definition

- Typedef is a keyword that allows the programmer to create new data type name for an already existing datatype.
- Syntax:

```
typedef    old datatype     newdatype ;
```

```
Example:      typedef  int  MARKS;
              typedef  float AMOUNT;
```

Example:

- Program to compute simple interest using typedef definition

```
#include<stdio.h>
typedef  float AMOUNT;
typedef  float TIME;
typedef float RATE;
void main( )
{
        AMOUNT   p,si;
        TIME  t;
        RATE  r;
        printf("enter the value of p,t,r\n");
        scanf("%f%f%f",&p,&t,&r);
        si=(p*t*r)/100;
        printf("si=%f\n",si);
}
```

Advantages of typede:

1.Provide a meaningful way of declaring variable

2.Increases readability of program

3.A complex and lengthy declaration can be reduced to short and meaningful declaration

4.Helps the programmer to understand source code easily.

4.3 Structures

- **Definition:** A *structure* is defined as a collection of variables of same data type or dissimilar datatype grouped together under a single name.

Syntax	Example
struct tagname **{** **datatype member1;** **datatype member2;** **datatype member3;** **}**	**struct** student **{** **char name[10];** **int usn;** **float marks;** **};**

where
struct is a keyword which informs the compiler that a structure is being defined
tagname: name of the structure
member1,member2: members of structure:
type1,type 2 : int,float,char,double

4.3.1 Structure Declaration

As variables are declared ,structure are also declared before they are used:

Three ways of declaring structure are:

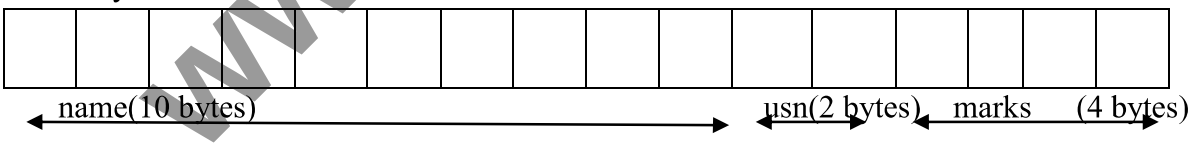
- Tagged structure
- Structure without tag
- Type defined structures

1.Tagged structure	
syntax **struct tag_name** **{** **data type member1;** **data type member2;** **----------** **----------** **};**	**Example** **struct student** **{** **char name[20];** **int usn;** **float marks;** **};**
Declaration of structure variables	
struct tagname v1,v2,v3…vn;	**struct student s1,s2,s3;**

2.structure without tagname	
syntax **struct** **{** **data type member1;** **data type member2;** **;** **----------** **----------** **}v1,v2,v3;**	**Example** **struct** **{** **char name[20];** **int usn;** **float marks;** **}s1,s2;**

3.Type defined structure	
syntax **typedef struct** **{** **data type member1;** **data type member2;** **----------** **----------** **}TYPE_ID;**	**Example** **typedef struct** **{** **char name[20];** **int usn;** **float marks;** **}STUDENT;**
Declaring structure variables	
TYPE_ID v1,v2,v3…vn;	**STUDENT s1,s2,s3;**

Memory Allocation for structure variable s1:



memory allocated for a structure variable s1=memory allotted for name+usn+marks

10+2+4

16 bytes.

4.3.2 Structure initialization

Syntax:

struct tagname variable={v1,v2….vn};

example
struct student s1={“sony”,123,24};

4.4 Accessing structures

- The members of a structure can be accessed by specifying the variable followed by dot operator followed by the name of the member.
- For example,
  consider the structure definition and initialization along with memory representation as shown below:

```
struct student
{
        char    name[20];
        int     usn;
        float   marks;

} s1;
struct student s1 = {"aditi",002,40};
```

By specifying
Variblename . membername
Example
S1.name
S1.usn
S1.marks

4.5 Structure operations

1. **Copying of structure variables**
2. **Arithmetic operations on structures**
3. **Comparision of two structures**

1. Copying of structure variables

- copying of two structure variables is achieved using assignment operator.
- Consider two structure definition of student and emplpoyee

struct student **{** **char name[20];** **int usn;** **float marks;** **};**	**struct employee** **{** **char ename[20];** **int eid;** **float salary;** **};**

struct student s1,s2,s3;	struct employee e1,e2,e3;

s1=s2; //valid
s2=s1; //valid
 	
s1=e1;//invalid	

2. **Arithmetic operations on structures**
   - Any arithmetic operation can be performed on the members of a structure Example
   - S1.marks++;
   - S2.marks++;
3. **Comparision of two structures**
   - variables of same structure definition can be compared along with member.
   - S1.usn!=s2.usn;
   - S1.marks==s2.marks
   - S1.name==s2.name

4.6 Pointers to structure

- A variable which contains address of a structure variable is called pointer to a structure.

typedef struct **{** **char name[20];** **int usn;** **float marks;** **} STUDENT;**	} Structure definition

STUDENT s1,s2,s3;
STUDENT *p;
P=&s1;
	// structure variables
	// pointer declaration
	// assign the address of structure s1 to pointer variable p

Access the members of structures using pointers

There are two methods:

1. Dereferencing operator (*)
2. Selection operator (→)

1. Dereferencing operator (*)
   - If **p is a pointer** to structure student, members can be accessed as follows:
   - (*p).name using this name can be accessed

- (*p).usn using this usn can be accessed
- (*p).marks using this marks can be accessed

2. Selection operator (→)
   - If **p is a pointer** to structure student, members can be accessed as follows:
   - **p→name** using this name can be accessed
   - **p→usn** using this usn can be accessed
   - **p→marks** using this marks can be accessed

4.7 Complex structures

1. Nested structures
2. Structure containing arrays
3. Structure containing pointers
4. Array of structures

1. Nested structures
   - A structure **which includes another structure** is called nested structure.

```
struct   subject
{
        int marks1;
        int marks1;
        int marks1;
};
typedef  struct
{
        char name[20];
        int usn;
        struct   subject PCD;
        float avg;
} STUDENT;
STUDENT   s1;
```

Structure definition with tagname subject

//Structure definition with typeid **STUDENT**

// include structure subject with a member name **PCD**

- Structure student has member called PCD,which inturn is a structure .
- Hence **STUDENT** structure is a nested structure.
- We can access various members as follows:

  S1.name;
  S1.usn;
  S1.PCD.marks1;

```
S1.PCD.marks2;
S1.PCD.marks3;
S1.avg;
```

4.8 Array of structure

- As array of integers we can have array of structures
- Suppose we want to store information of 5 students consisting of name,usn,marks,structure definition is as follows:

```
typedef  struct
{
        char name[20];
        int usn;
        float marks;
} STUDENT;
STUDENT   s[5];
```

If n=3

- We can access the first student information as follows:
  S[0].name
  S[0].usn
  S[0].marks
- We can access the second student information as follows
  S[1].name
  S[1].usn
  S[1].marks
- We can access the second student information as follows
  S[2].name
  S[2].usn
  S[2].marks

hence i ranges from 0 till 2 if n=3

```
for(i=0;i<n;i++)
{
        printf("enter the %d student details\n",i+1);
        printf("enter the roll number:\n");
        scanf("%d",&s[i].rollno);
        printf("enter the student name:\n");
        scanf("%s",s[i].name);
```

```
                printf("enter the marks:\n");
                scanf("%d",&s[i].marks);
                printf("enter the grade:\n");
                scanf("%s",s[i].grade);
        }
```

Example Programs:

1.Write a c program to store information of n students with name,usn and marks.print the name of the students whose marks is **greater than 90**.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct   student
{
        char name[20];
        int  usn;
        float marks;

};
void main()
{               int i,n;
                struct student s[10];    // array  of structure declaration
                printf("enter the number\n");
                scanf("%d",&n);
                for(i=0; i<n; i++)
                {
                        printf("enter the %d student details\n",i+1);
                        printf("enter the student name:\n");
                        scanf("%s",s[i].name);
                        printf("enter the usn:\n");
                        scanf("%d",&s[i].usn);
                        printf("enter the marks:\n");
                        scanf("%d",&s[i].marks);
                }

                for(i=0;i<n;i++)
                {
                        if(s[i].marks>90)
```

//structure definition for student

enter details of n students

//check if the marks is greater than 90

```
                    {   printf("\n marks of the student is %d \n",s[i].marks);
                        exit(0);
                    }
                }
        printf("student name NOT FOUND\n");
}
```

2.Write a C program to maintain a record of **n** student details using an array of structures with four fields (Roll number, Name, Marks, and Grade). Assume appropriate data type for each field. Print the marks of the student, given the student name as input.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct student
{       int rollno,marks;
        char name[20];
        char grade[2];
};
void main()
{               int i,n;
                struct student s[10];
                char key[20];
                clrscr();
                printf("enter the number\n");
                scanf("%d",&n);
                for(i=0;i<n;i++)
                {
                        printf("enter the %d student details\n",i+1);
                        printf("enter the roll number:\n");
                        scanf("%d",&s[i].rollno);
                        printf("enter the student name:\n");
                        scanf("%s",&s[i].name);
                        printf("enter the marks:\n");
                        scanf("%d",&s[i].marks);
                        printf("enter the grade:\n");
                        scanf("%s",&s[i].grade);
                }
        printf("enter the key student name:\n");
        scanf("%s",key);
        for(i=0;i<n;i++)
```

```
        {
                if(strcmp(s[i].name,key)==0)
                {       printf("\n marks of the student is %d \n",s[i].marks);
                        exit(0);
                }
        }
        printf("student name NOT FOUND\n");
}
```

4.9 Structure and functions

- The structure or structure members can be passed to functions in various ways as shown below:

  Various ways of passing structures to functions:

  1. By passing individual members of a structure
  2. By passing whole structure
  3. By passing structure through pointers

1. By passing individual members of a structure to a function

- A function can be called by passing individual members of a structure as actual parameters.
- Consider the below program which demonstrates the multiplication of fraction:

Program	User defined function to multiply
#include<stdio.h> **typedef struct** **{** **int n;** **int d;** **}FRACTION;** **void main()** **{** **FRACTION a,b,c;** **printf("enter fraction 1");** **scanf("%d%d", &a.n, &a.d);** **printf("enter fraction 2");** **scanf("%d%d", &b.n, &b.d);** **/* function call with numerator of a and b*/** **c.n=multiply(a.n , b.n);** **/*same as above function call with denominator of a and b*/**	***Continued….*** **int multiply(int x,int y)** **{** **return x*y;** **}**

```
        c.d=multiply(b.d, b.d);
        printf("fraction c is %d / %d", c.n, c.d);
}
```

Continued to next user defined function….

2. By passing whole structure

Program	User defined function to multiply

**#include<stdio.h>
typedef struct
{
int n;
int d;
}FRACTION;
void main()
{
FRACTION a,b,c;
printf("enter fraction 1");
scanf("%d %d", &a.n, &a.d);
printf("enter fraction 2");
scanf("%d %d", &b.n, &b.d);
/* send together numerator and denominator of a and b*/
c=multiply(a , b); //func call
printf("fraction c is %d / %d", c.n, c.d);
}**
	Continued….
	
	**FRACTION multiply(FRACTION x, FRACTION y)
	{
	FRACTION z;
	z.num=x.n*y.n;
	z.deno=x.d*y.de;
	return;
	}**

3. By passing structure through pointers

Program	User defined function to multiply

**#include<stdio.h>
typedef struct
{
int n;
int d;
}FRACTION;
void main()
{
FRACTION a,b,c;
printf("enter fraction 1");
scanf("%d %d", &a.n, &a.d);
printf("enter fraction 2");
scanf("%d %d", &b.n, &b.d);**
	Continued….
	
	**void multiply(FRACTION *x, FRACTION *y, FRACTION *z)
	{
	z→n = x→n * y→n;
	z→d = x→d * y→d;
	return;
	}**

/* send together numerator and denominator of a and b*/
multiply(&a , &b ,&c); //func call
printf("fraction c is %d / %d", c.n, c.d);
}	

Uses of structure:

- Structures are used to represent more complex data structure.
- Related data items of dissimilar data types can be grouped under a common name.
- Can be used to pass arguments so as to minimize the number of function arguments.
- When more than one data has to be returned from the function, then structures can be used.
- Extensively used in applications involving database management.

File Handling

4.10:Definitions

File: A file is defined as a collection of data stored on the secondary device such as hard disk. **FILE** is type defined structure and it is defined in a header file "stdio.h". FILE is a derived data type.FILE is not a basic data type in C language.

Input File: An input file contains the same items that might have typed in from the keyboard.

Output File: An output file contains the same information that might have been sent to the screen as the output from the program.

Text(data) File:A text file is the one where data is stored as a stream of characters that can be processed sequentially.

4.11 Steps to be perform file manipulations

1. Declare a file pointer variable
2. Open a file
3. Read the data from the file or write the data into file
4. Close the file

1.Declare a file pointer variable

- Like all the variables are declared before they are used, a file pointer variable should be declared.
- File pointer is a variable which contains starting address of file.
- It can be declared using following syntax:

FILE *fp;

Example:

```
#include <stdio.h>
void main()
{
        FILE *fp;          /* Here, fp is a pointer to a structure FILE */
        ---------------------------
        ------------------------
}
```

2.File open Function

The file should be opened before reading a file or before writing into a file.

Syntax:

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

Where,

fp is a feile pointer
fopen() function to open file
mode is **"r","w","a"**

- **fopen()** function will return the starting address of opened file and it is stored in file pointer.
- If file is not opened then fopen function returns NULL

```
if (fp == NULL)
{
        printf("Error in opening the file\n");
        exit(0);
}
```

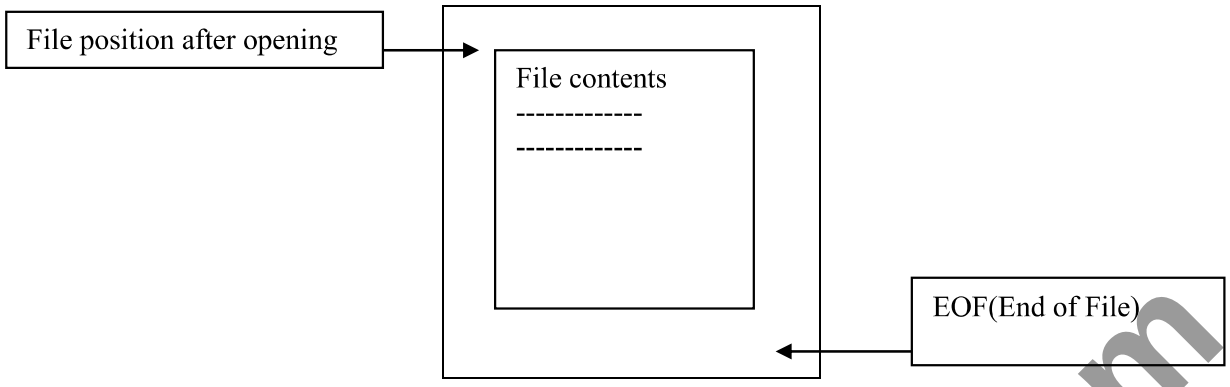
Modes of File

The various mode in which a file can be opened/created are:

Mode	Meaning
"r"	opens a text file for reading. The file must exist.
"w"	creates an text file for writing.
"a"	Append to a text file.

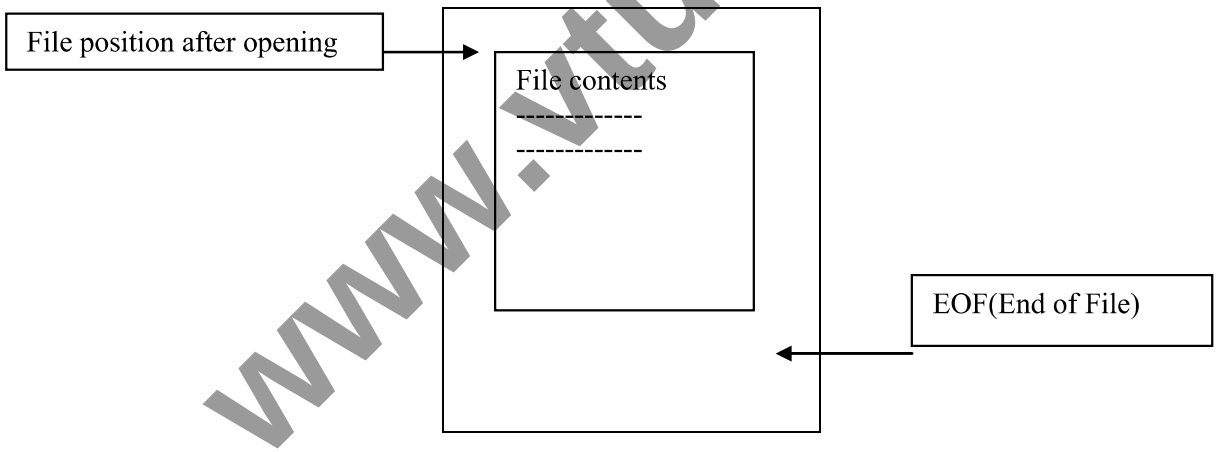
1.Read Mode("r")

- This mode is used for opening an existing file to perform read operation.
- The various features of this mode are
  - Used only for text file
  - If the file does not exist, an error is returned
  - The contents of the file are not lost
  - The file pointer points to beginning of the file.



2.Write Mode("w")

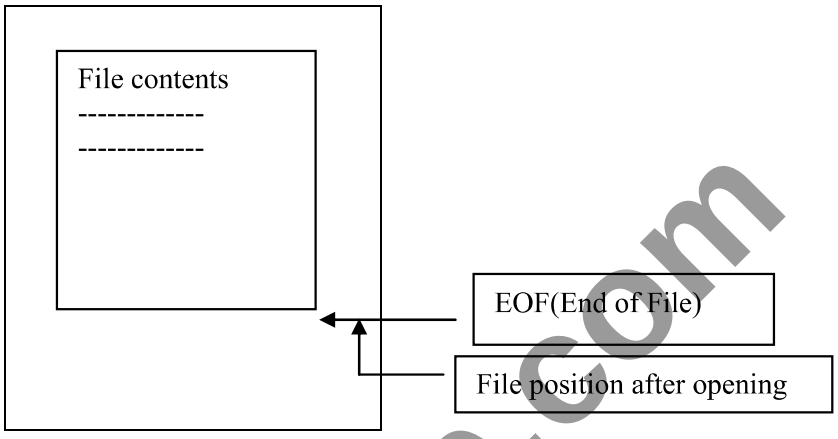
- This mode is used to create a file for writing.
- The various features of this mode are
  - If file does not exist, a new file is created.
  - If a file already exists with the same name, the contents of the file are erased and the file is considered as a new empty file.
  - The file pointer points to the beginning of the file.



3.Append Mode("a")

- This mode is used to insert the data at the end of the existing file.
- The various features of this mode are
- Used only for text file.
- If the file already exist, the file pointer points to end of the file.

- If the file is does not exist. A new file is created.
- Existing data cannot be modified.



Examples

Read Mode	Write Mode

#include<stdio.h>
 	
FILE *fp // File Pointer
fp=fopen("civil.txt","r"); //opening file civil in read mode
 	
if (fp == NULL) //file does not exist
{
printf("Error in opening the file\n");//error message
exit(0); //terminate the program
}
 	
fclose(fp); // close the file civil.txt
	#include<stdio.h>
	
	FILE *fp // File Pointer
	fp=fopen("ecb.txt","w"); //opening file ecb in write mode
	
	if (fp == NULL) //file does not exist
	{
	printf("Error in opening the file\n");//error message
	exit(0); //terminate the program
	}
	
	fclose(fp); // close the file ecb.txt

Append Mode
#include<stdio.h> FILE *fp // File Pointer

```
fp=fopen("ecb.txt","a");   //opening file ecb in write mode

if (fp == NULL)          //file does not exist
{
printf("Error in opening the file\n");//error message
exit(0);                  //terminate the program
}

fclose(fp); // close the file ecb.txt
```

3.Closing a file

- When we no longer need a file, that file should be closed.
- This is the last operation to be performed on a file.
- A file can be closed using **fclose()** function.
- If a file is closed successfully, 0 is returned, otherwise EOF is returned.

Syntax:

fclose(file pointer);

Example:

fclose(fp);

4.12 I/O(Input and Output) file functions

The three types of I/O functions to read from or write into the file

I. File I/O functions for **fscanf**() and **fprintf**()
II. File I/O functions for strings **fgets**() and fputs()
III. File I/O functions for characters fgetc() and fputc()

File I/O functions for **fscanf**() and **fprintf**()

1.fscanf():

The function **fscanf** is used to get data from the file and store it in memory.

Syntax:

fscanf(fp, "format string", address list);

where,

"fp" is a file pointer.It points to a file from where data is read.

"format String": The data is read from file and is stored in variable s specified in the list ,will take the values from the specified pointer fp by using the specification provided in format sting.
"**address list":**address list of variables

Note:fsanf() returns number of items successfully read by fscanf function.

Example:

FILE *fp
fp=fopen("name.txt","r");
fscanf("fp,"%s",name);
	FILE *fp
	fp=fopen("marks.txt","r");
	fscanf("fp,"%d%d%d",&m1,&m2,&m3);

Note:
1.If the data is read from the keyboard then use **stdin** in place of **fp**
2.If the data is read from the file then use **fp**

2.fprintf():
The function **fprintf** is used to write data into the file.

Syntax:

fprintf(fp, "format string", variable list);

where,
"fp" is a file pointer.It points to a file where data to be print.
"format String": group of format specifiers.
"**address list":**list of variables to be written into file

Note:fprinf() returns number of items successfully written by fprintf function.

Example:

FILE *fp
fp=fopen("name.txt","w");
fscanf("fp,"%s",name);
	FILE *fp
	fp=fopen("marks.txt","w");
	fscanf("fp,"%d%d%d",m1,m2,m3);

Note:
1.If the data has to be printer on output screen then use **stdout** in place of **fp**
2.If the data has to be written to the file then use **fp**

Example Program

Write a C program to read the contents of two files called as name.txt and usn.text and merge the contents to another file called as output.txt and display the contents on console using fscanf() and fprintf()

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        FILE *fp1,*fp2,*fp3;
        char name[20];
        int usn;
        fp1=fopen("name.txt","r");
        fp2=fopen("usn.txt","r");
        fp3=fopen("output.txt","w");
        for(;;)
        {
            if(fscanf(fp1,"%s",name)>0)
            {
                    if(fscanf(fp2,"%d",&usn)>0)
                    {
                        fprintf(fp3,"%s %d\n",name,usn);
                    }
                    else break;
            }
            else break;
        }
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
        fp3=fopen("output.txt","r");
        printf("NAME\tUSN\n");
        while(fscanf(fp3,"%s %d\n",name, &usn)>0)
        {
                printf("%s \t%d\n",name,usn);
        }
        fclose(fp3);
}
```

File I/O functions for fgets() and fputs()

1.fgets()

fgets() is used to read a string from file and store in memory.

Syntax:

ptr=fgets(str,n,fp);

where

fp ->file pointer which points to the file to be read

str ->string variable where read string will be stored

n ->number of characters to be read from file

ptr->If the operation is successful, it returns a pointer to the string read in.

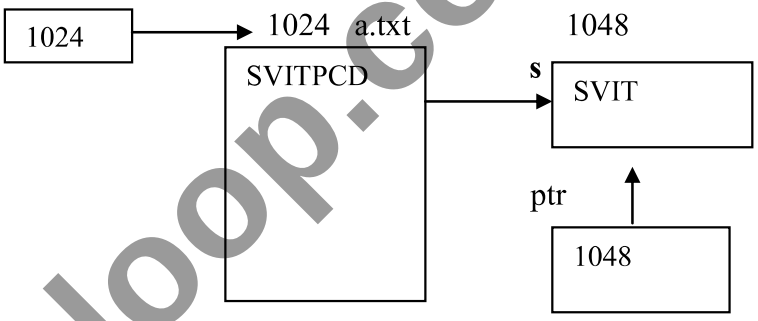
Otherwise it returns NULL.

The returned value is copied into ptr.

Example:

```
FILE *fp;
char s[10];
char *ptr;
fp=fopen("a.txt","r");

if(fp==NULL)
{
printf("file cnnnot be opened);
exit(0);
}
ptr=fgets(s,4,fp);
fclose(fp);
```



Example Programs:

1.Write a C program to read from file using function fgets.
#include<stdio.h>
void main()
{
FILE *fp;
char str[15];
char *ptr;
fp=fopen("name.txt","r"):
if(fp==NULL)
{
printf("file cannot be opened");
exit(0);
	1.Write a C program to read string from keyboard using function fgets.
	#include<stdio.h>
	void main()
	{
	char str[15];
	char *ptr;
	printf("Enter the string");
	ptr=fgets(str,10,stdin);
	if(ptr==NULL)
	{
	printf("reading is unsuccessful");
	exit(0);

```
}
ptr=fgets(str,10,fp);
if(ptr==NULL)
{
     printf("reading is unsuccessful");
      exit(0);
}
printf("string is");
puts(str);
fclose(fp);
}
```

```
}
printf("string is");
puts(str);
fclose(fp);
}
```

2.fputs()

fputs() is used to write a string into file.

Syntax:

fputs(str,fp);

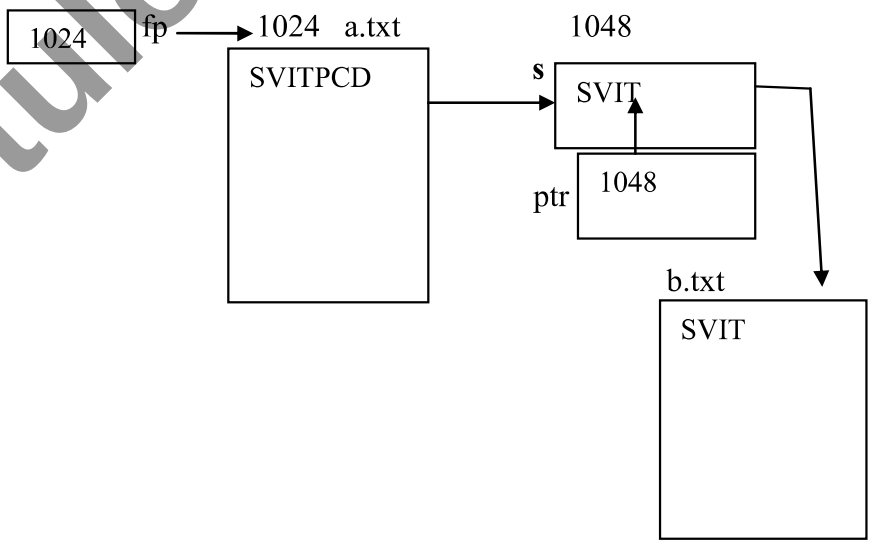
where

fp ->file pointer which points to the file to be read

str ->string variable where read string will be stored

Example:

```
FILE *fp,*fp1;
char s[10];
char *ptr;
fp=fopen("a.txt","r");
fp1=fopen("b.txt","w");
if(fp==NULL)
{
printf("file cnnnot be opened);
exit(0);
}
ptr=fgets(s,4,fp);
fputs(s,fp1);
fclose(fp);
fclose(fp1);
```



Example Program:

1.Write a C program to read from file using function fgets and print into file using fputs function.

```
#include<stdio.h>
void main()
{
```

```
 FILE *fp,fp1;
 char  str[15];
 char *ptr;
fp=fopen("name.txt","r");
fp1=fopen("output.txt","w");
if(fp==NULL)
{
        printf("file cannot be opened");
         exit(0);
}
ptr=fgets(str,10,fp);
if(ptr==NULL)
{
    printf("reading is unsuccessful");
     exit(0);
}
fputs(str,fp1);
fclose(fp);
fclose(fp1);
}
```

File I/O functions for fgetc() and fputc()

1.fgetc()

fgetc() function is used to read a character from file and store it in memory.

Syntax:

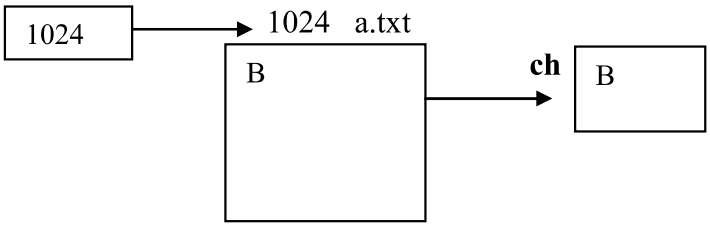
```
ch=fgetc(fp);
```

Example 1:

```
FILE *fp;
fp=fopen("sec.txt","r");
ch=fgetc(fp);
```

Example 2:

```
FILE *fp;
char ch;
fp=fopen("a.txt","r");
ch=fgets(fp);
fclose(fp);
```



3.fputc()

fgetc() function is used to write a character into a file.

Syntax:

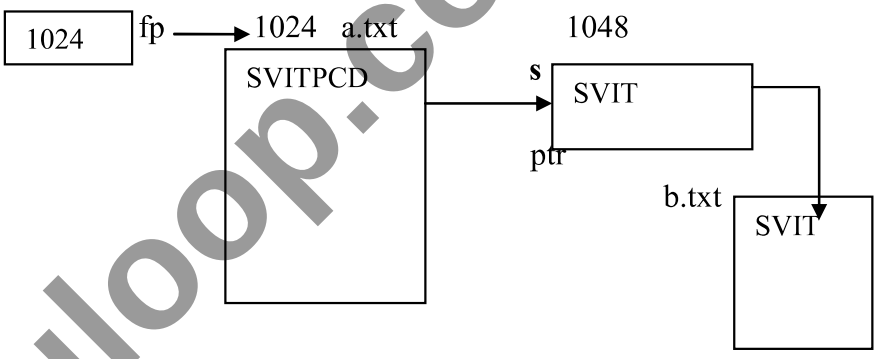
fputc(ch,fp);

Example 1:

```
FILE *fp;
fp=fopen("sec.txt","w");
fputc(ch,fp);
```

Example 2:

```
FILE *fp,*fp1;
char ch;
fp=fopen("a.txt","r");
fp1=fopen("b.txt","w");
ch=fges(fp);
fputc(ch,fp1);
fclose(fp);
fclose(fp1);
```



Example Programs

1.Write a C program to copy one file to another using fgetc() and fputc() functions.

```
#include<stdio.h>
void main()
{
FILE *fp1,*fp2;
char ch;
fp1=fopen("file1.txt","r");
fp2=fopen("file2.txt","w");
while((ch=fegtc(fp1))!=EOF)
{
fputc(ch,fp2);
}
fclose(fp1);
fclose(fp2);
```

2.Write a C program to concatenate two files using fgetc() and fputc() functions.

```
#include<stdio.h>
void main()
{
FILE *fp1,*fp2,*fp3;
char ch;
fp1=fopen("file1.txt","r");
fp2=fopen("file2.txt","r");
fp3=fopen("file3.txt","w");

while((ch=fegtc(fp1))!=EOF)
{
fputc(ch,fp3);
}
```

}	while((ch=fegtc(fp2))!=EOF)
	{
	fputc(ch,fp3);
	}
	fclose(fp1);
	fclose(fp2);
	fclose(fp3);
	}

3.Write a C program for counting the characters, blanks, tabs and lines in file.

```
#include<stdio.h>
void main()
{
FILE *fp;
char ch;
int cc=0,bc=0,tc=0,lc=0;
fp1=fopen("file1.txt","r");
while((ch=fegtc(fp1))!=EOF)
{
cc++;
if(ch==' ')      bc++;
if(ch==' \n')    lc++;
if(ch=='\t ')    tc++;
}
fclose(fp);
printf("total number of characters=%d\n",cc);
printf("total number of tabs=%d\n",tc);
printf("total number of lines=%d\n",lc);
printf("total number of blanks=%d\n",bc);
}
```


4.13 **Command Line Arguments**

- The interface which allows the user to interact with the computer by providing instructions in the form of typed commands is called command line interface.
- In the command prompt user types the commands.

Example:

In MS_DOS command prompt looks as follows:

C:\>copy T1.c T2.c

The above copy command copies contents of T1.c to T2.c. In the above line **copy**, T1.c and T2.c are called command line arguments.

Write a C program to accept a file either through command line or as specified by user during runtime and displays the contents.

```
#include<stdio.h>
#include<string.h>
void main(int argc,char *argv[])
{
FILE *fp;
char fname[10];
char ch;
if(arg==1)
{
printf("\n Enter file name\n");
scanf("%s",fname);
}
else
{
strcpy(fname, argv[1]);
}
fp=fopen(fname,"r");
if(fp==NULL)
{
printf("cannot open file");
exit(0);
}
printf("contents of file are\n");
while((ch=fgetc(fp))!=EOF)
{
printf("%c",ch);
}
}
```