MODULE-5

STRUCTURES AND FILES

LEARNING OBJECTIVES

- Explain how structures are used in C.
- Describe how structure variable and members are manipulated.
- Know the concept of File
- Discuss Files I/O functions
- Opening and closing of Files

Basics of Structure:

- **"A Structure is a user defined data type, which is used to store the values of different data types together under the same name".**

or

- **"A structure is a collection of one or more variables of same data type or dissimilar data types grouped under a single name for convenient handling".**
- Each member of a structure is assigned a separate memory location.
- A structure helps us to organize complicated data, particularly in large programs, because they permit a group of related variables to be treated as a unit instead of as separate entities.

Defining a Structure:

- Definition of structures starts with the '**struct**' keyword, followed by the name of the structure, a pair of curly braces containing declaration of a set of variables called structure members and then semicolon at the end.

Syntax:

```
struct structure_name
{
        data_type1      variable1;
        data_type2      variable2;
        ……….                          } structure members
        ……….
        data_typen      variablen;
};
```

- **Struct:** It is a keyword, and it indicates the beginning of a structure definition.
- **Structure:** It represents the name of the structure being defined.
- **Data_type1 to n:** Represent the data types the structure members.
- **Variables1 to n:** Represent the variables in the Structure members.

Examples:

1.
```
struct employee
{
    int emp_id, emp_age;
    char name[20], grade;
    float emp_salary;
};
```

2.
```
struct book
{
    char book_title[20];
    char author[20];
    float cost_of_book;
    int pages;
};
```

Declaring Structure Variables:

- We can declare variables of the structure name in the same way as we declare variables of predefined data type.

Syntax:

```
struct structure_name variable_name;
```

Example:

```
struct student
{
    int roll_no, age;
    char name[20], grade;
    float collg_fee;
};
struct student student1;      /*student1 is structure variable
```

Structure template with tag Here student is the tag

- We can also declare **multiple structure variables** using a single line of code by using following syntax:

Syntax:

```
struct structure_type variable1,variable2,…..variablen;
```

Example:

```
struct student
{
    int roll_no, age;
    char name[20], grade;
    float collg_fee;
};
struct student student1,student2;
```

- **The definition of a structure and declaration of structure variables can be done together.**

Syntax:

```
struct structure_name
{
        data_type1      variable1;
        data_type2      variable2;
        ……….
        ……….
        data_type n     variablen;

} structure_variable1, structure_variable2,……, structure_variablen;
```

Example:

```
struct student
{
        int roll_no, age;
        char name[20], grade;
        float collg_fee;
}student1,student2;
```

- Structure name is student and we have declared two variables of type student: student1 and student2.

Accessing members of a Structure:

- **A member of a structure is identified and accessed using the (.) Dot Operator. The Dot Operator connects the member name to the name of its containing Structure.**

Syntax:

```
structure_variable_name.member;
```

Example:

```
student1.roll_no;

student1.name;
```

Initializing Structure variable:

- Each structure variable contains a copy of all the members of the structure.
- Initializing a structure variable involves assigning values to its structure members.
- We can assign values to the members of the structure variables either all at once or one at a time.

i. In first approach, the declaration of the structure variable and the initialization of the structure members are done using a single line of code.

Example:

```
struct student
{
        int roll_no, age;
        char name[20], grade;
        float collg_fee;
};
struct student student1={50,18, "Rama", 'A',35500.00};
```

ii. In the second approach, we first need to declare the structure variables and then initialize each member of the structure variable one by one.

- Initializing each structure members separately requires us to access individually structure members. We can access a structure member by dot (.) operator.

Syntax :

```
structure_variable_name.member_name=value;
```

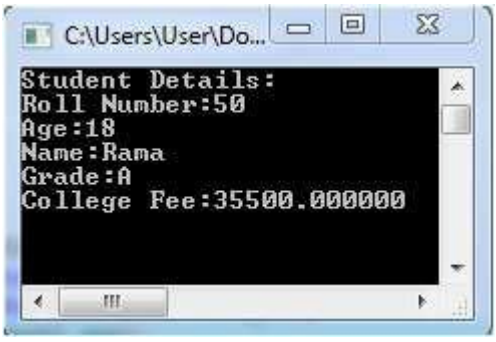
structure_name: It represents the name of the Structure variable.
member_name: It represents a member of the Structure variable.
value: It represents the value being assigned to the Structure member.

Example:

```
struct student
{
        int roll_no, age;
        char name[20],grade;
        float collg_fee;
} student1;
student1.roll_no=50;
student1.age=18;
student1.name= "Rama";
student1.name= 'A';
student1.collg_fee=35500.00;
```

Example: Write a C program to print the details of student using structure.

```
#include <stdio.h>
#include<conio.h>
struct student
{
        int roll_no, age;
        char name[20], grade;
        float collg_fee;
};
void main( )
{
      struct student student1={50,18, "Rama", 'A',35500.00};
       clrscr();
       printf("Student Details:\n");
       printf("Roll Number:%d\n",student1.roll_no);
       printf("Age:%d\n",student1.age);
       printf("Name:%s\n",student1.name);
       printf("Grade:%c\n",student1.grade);
       printf("College Fee:%f\n",student1.collg_fee);
       getch();
}
```



Structures and Functions:

- We can pass an entire structure as a parameter to a function instead of passing individual structure members.
- Structures are passed by value and enable a function to alter a parameter and return a value as structure.
- Function definition must consist of declaration of a Structure and Structure variable as its formal parameter.
- The Structure variables used in both calling function (actual parameter) and in called function (formal parameter) belong to same Structure name or Structure template.

Example: Write a C program to pass a structure as a parameter to a function.

```
#include <stdio.h>
#include<conio.h>
struct student
{
        int roll_no, age;
        char name[20], grade;
        float collg_fee;
};
void main ( )
{
        struct student st;
        printf("Enter the student Details:\n");
        printf("Enter the student Roll_no:");
        scanf("%d",&st.roll_no);
        printf("Enter the student Age:");
        scanf("%d",&st.age);
        printf("Enter the student Name:");
        scanf("%s",st.name);
        printf("Enter the student Grade:");
        scanf("%c",st.grade);
        printf("Enter the College fee:");
        scanf("%f",&st.collg_fee);
        print_details(st);
        getch();
}
void print_details(st)
{
         printf("Student details:\n");
         printf("Roll_no:%d\n Age:%d\n Name:%s \n Marks:%d\n College
         fee:%f",st.roll_no,st.marks,st.name);

}
```

OUTPUT:

```
Enter the student Details:
Enter the student Roll_no: 10
Enter the student Age: 18
Enter the student Name: Rama
Enter the student Grade: A
Enter the College fee: 35500
        Student details:
        Roll_no: 10
        Age: 18
        Name: Rama
        Grade: A
        College fee: 35500.00
```

- ✓ Here structure named 'student' is defined and then the variable of that structure st has been declared. This structure variable st is then passed as a parameter to the print_details () function.

Nested Structures:

- ✓ We can nest a structure inside another Structure.
- ✓ The structures can be nested in two ways.

1. In first approach, the complete definition of a structure is placed inside the definition of another structure.

Example:
```
struct student
{
        int roll_no,age;
        char name[20],grade;
        float collg_fee;
        struct date
        {
                int day, month, year;
        } dob;
}student1;
```

2. In the second approach, the structures are defined separately and a variable of a Structure type is declared inside the definition of another Structure.

Example:
```
struct date
{
        int day, month, year;
} dob;
struct student
{
        int roll_no,age;
        char name[20],grade;
        float collg_fee;
        struct date dob;
} student1;
```

- ➢ We can access the variables of a structure type that are nested inside another structure in the same way as we access other members of that structure.

Example: Write a C program to print the student details using nested structures.

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

struct student
{
        int roll_no;
        char name[20], grade;
        float collg_fee;
        struct date
        {
                int day,month,year;
        }dob;
};
```

OUTPUT:

```
Enter the student Details:
Enter the student Roll_no: 10
Enter the student Name:
RamaEnter the student
Grade: A Enter the College
fee: 35500
Enter the Date of Birth in dd/mm/yyyy
format: 13 05 1989
Student details:
        Roll_no: 10
        Name:
        Rama
        Grade: A
```

```
void main ( )
{
        struct student student1;
        printf("Enter the student Details:\n");
        printf("Enter the student Roll_no:");
        scanf("%d",&student1.roll_no);
        printf("Enter the student Name:");
        scanf("%s",student1.name);
        printf("Enter the student Grade:");
        scanf("%c",student1.grade);
        printf("Enter the College fee:");
        scanf("%f",&student1.collg_fee);
        printf("Enter the Date of Birth in dd/mm/yyyy format:");
        scanf("%d%d%d",&student1.dob.day, &student1.dob.month, &student1.dob.year);
        printf("Student Details:\n");
        printf("Roll_no:%d\n",student1.roll_no);
        printf("Name:%s\n",student1.name);
        printf("Grade:%c\n",student1.grade);
        printf("College fee:%f\n",student1.collg_fee);
        printf("Date_of_Birth:%d/%d/%d\n",student1.dob.day,student1.dob.month,student1.do
          b.year);
        getch();
}
```

Array of Structures:

- Similar to creating arrays of a predefined data type such as int, float and char, we can also create an array of structure type.
- Arrays of Structure type are required in situations when we need to apply the same Structure to **a set of Objects**.

Example: Defining a Structure and then creating an array of Structure type:

```
struct student
{
        int roll_no,age;
        char name[10],grade;
        float collg_fee;
}students[20];
```

- Here we have created a structure called 'student' with 5 fields: roll_no, age, name, grade and collg_fee and array of structure type of size 20(We can store the details of 20 students).

Example:

1. Write a C program to print the details of students using array of structures.

```
#include<stdio.h>
#include<conio.h>
struct student
{
        int roll_no,age;
        char name[10],grade;
        float collg_fee;
}students[20];

void main()
{
        int i;
        printf("Enter the details of 3 Students:\n");
        for(i=0;i<3;i++)
        {
                printf("Enter the Roll_no:");
                scanf("%d",&students[i].roll_no);
                printf("Enter the age:");
                scanf("%d",&students[i].age);
                printf("Enter the Name:");
                scanf("%s",students[i].name);
                printf("Enter the Grade:");
                scanf("%c",students[i].grade);
                printf("Enter the College fee:");
                scanf("%f",&students[i].collg_fee);
                }
                printf("Student details are:\n");
                for (i=0;i<3;i++)
                {
                    printf("RollNumber:%d\n",students[i].roll_no);
                    printf("Age:%d\n",students[i].age);
                    printf("Name:%s\n",students[i].name);
                    printf("Grade:%c\n",students[i].grade);
                    printf("Collegefee:%f\n",students[i].collg_fee);
                }
                getch();
          }
```

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<conio.h>
struct student
{
        int  rollno,marks;
        char name[20],grade;
};
```

```
void main()
{
        int i,n,found=0;
        struct student s[10];
        char sname[20];
        printf("Enter the number of student details 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("%c",&s[i].grade);
        }
        printf("\n Student details are:\n");
        printf("\nRollno\t\tName\t\tMarks\t\tGrade\n");for(i=0;i<n;i++)
        {
                printf("%d\t\t%s\t\t%d\t\t%c\n",s[i].rollno,s[i].name,s[i].marks,s[i].grade);
        }
        printf("Enter the student name to print the marks:");
        scanf("%s",sname);
        for(i=0;i<n;i++)
        {
                if(strcmp(s[i].name,sname)==0)
                {
                        printf("Marks of the student is:%d",s[i].marks);//s[1].marks
                        found=1;
                }
        }
        if(found==0)
                printf("Given student name not found\n");
        getch();
}
```

OUTPUT:

```
Enter the number of student details n=3
Enter the 1 student details:
Enter the Roll number: 123
Enter the student name: Abhi
Enter the marks: 90
Enter the grade: A

Enter the 2 student details:
Enter the Roll number: 124
Enter the student name: Bharath
Enter the marks: 65
```

```
Enter the grade: B
Enter the 3 student details:
Enter the Roll number: 125
Enter the student name: Chethan
Enter the marks: 50
Enter the grade: C

Student details are:
```

	Rollno	Name	Marks	Grade
S[0]	123	Abhi	90	A
S[1]	124	Bharath	65	B
S[2]	125	Chethan	50	C

```
Enter the student name to print the marks: Bharath
Marks of the student is: 65
```

The typedef Statement: (keyword)

- The 'typedef' statement allows us to create a new data type from an existing data type.
- The new data type has a different name but same characteristics as that of the existing data type.
- We create a new data type from both a predefined data types (int, float, char) and a user defined data type, such as structure.
- We can use the new data type to declare variables and arrays of the original data type.
- The Scope of the new data type is limited to the function in which it is to be created.

Syntax:

```
typedef old_data_type  new_data_type;
```

'typedef' is a keyword.
old_data_type: is the name of the original data type.
new_data_type: is the name of the new data type.

Example:

```
typedef int biet;
biet a, b, c;
biet a[10];
```

- Here first it creates a new data type called 'integer' from predefined data type int and declares 3 variables (a, b, c) and an array of type integer.

Example: Write a C program to print the Employee Details using type def statement.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
        int emp_id;
        char name[20];
        float salary;
};

void main()
{
```

```
typedef struct employee employee1;
employee1 emp;
clrscr();
printf("Enter the Employee id:\n");
scanf("%d",&emp.emp_id);
printf("Enter name of the Employee:\n");
scanf("%s",emp.name);
printf("Enter salary of the Employee:\n");
scanf("%s",emp.salary);
printf("Employee Details:\n");
printf("Employee ID:%d\n",emp.emp_id);
printf("Name:%s\n",emp.name);
printf("Salary:%f\n",emp.salary);
getch();
}
```

OUTPUT:

```
Enter the Employee id: 100
Enter name of the Employee: manu
Enter salary of the Employee: 10000
Employee Details:
Employee ID: 100
Name: manu
Salary: 10000.00
```

FILE MANAGEMENT

File

- **"A File is a collection of data or information stored on computer memory or the secondary device such as disk"**.
- An **Input file** (data file) contains the same item what we have typed in from the keyboard during interactive data entry.
- An **Output file** contains the same information that might have been sent to the screen as the output from our program.

Basic Operations on Files

- Declare a file pointer variable.
- Opening a file. If it does not exists, then creating a new file.
- Writing data to the file.
- Reading the data from the file.
- Determining the end of the file.
- Closing the file.

Declaring a file pointer variable

- **A 'File Pointer' is used to address the file from within a C program.**
- **A file pointer has data type FILE *, which means "pointer to a file".**
- The data type FILE is defined in 'stdio.h' as a structure. The structure details are hidden from the programmer and this structure is used to store information about a file.

Syntax :

```
FILE *file_pointer;
```

Example: FILE *fp;

File Management

- **"It refers to the process of writing data to a file, reading data from a file, and performing other operations on files".**

Streams in C

- A stream is a logical interface to the device that is connected to the computer.
- There are three standard streams in C language:
  i. **Standard Input (stdin):** Standard input is the stream from which the program receives its data. The requests transfers the data using the 'read' operation.
     Example: Keyboard, Mouse
  ii. **Standard Output(stdout):** It is the stream where a program writes its output data. The program requests data transfer using the 'write' operation.
     Example: Monitor, printer.
  iii. **Standard Error (stderr):** It is an output stream used by programs to report the error messages or diagnostics.
- ✓ A stream is linked to a file using an 'open' operation and disassociated with 'close' operation.

Opening and Closing of Files:

- Before using a file, a C program must open it, opening a file tells the operating system to look for a file by the specified name or if it does not exist, then create a new file and prepare it for read or write operation.
- After using a file, a C program must close it. Closing the file saves the file and any changes made to it.
- Between opening and closing a file, the program must specify in its I/O statements which file is to be used.

Opening a file:

- Opening a file is done by a call to the function "**fopen()**", which tells the operating system the name of a file and whether the file is to be opened for reading (input) or for writing (output).
- The function 'fopen()' associates a file with a stream and return a pointer to that stream. The pointer is the name by which the stream is known in the program.

Syntax:

```
file_pointer = fopen (xyz.txt, w);
```

- **fopen()**takes two parameters, both of which are strings.

where,
file name: It is a name of the disk file to be opened. If the file is not in the default directory, a full path name must be provided.
mode: It defines the way in which the file is to be opened. Mode is a string not a character and it must be enclosed in double quotation marks.

Return values

- The function may return the following:

i. File pointer of type FILE if successful.
ii. NULL if unsuccessful.

File Modes

i. read mode (r)- open a text file for reading

- If the mode is **"r",** the program will **read** from the file. The file should already exist.

ii. write mode(w)- open a text file for writing or if it does not exist then create a text file for writing

- If the mode is **"w"** the program will **write** to the file. If the file does not exist, it will be created. If the file does exist, it will be over written.

iii. append mode(a)- append to a text file

- If the mode is **"a"** the Program will **append** new output to the end of the file. If the file does not exist, it will be created.

Example Program: Write a C program to demonstrate the usage of fopen() function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    fp = fopen ("student.txt", "r");
    if(fp==NULL)
    {
            printf("Error in opening a file\n");
            exit(0);
    }
    /* Using fp access file contents*/
    ……..
    ……..
}
```

Closing a File:

- After a file has been used, it must be closed. This is done by a call to the function **"fclose()".**
- The **'fclose'** function breaks the connection between the stream and the file.

Syntax:

```
fclose (file_pointer);
```

- The function 'fclose' takes one parameter, which is a pointer to a file.

Return values

- The function may return the following:

i. 0 if successful.
ii. EOF (End of File) if unsuccessful. i.e. when if there is an error, such as trying to close a file that is not open.

Example Program: Write a C program to demonstrate the usage of fclose() function.

```
#include<stdio.h>
void main()
{
    FILE * fp;
    fp = fopen("student.txt", "r");
    if ( fp==NULL)
    {
            printf ("Error in Opening file\n");
            exit(0);
    }/* Using fp access file contents*/
    ……..
    if(fclose(fp)==EOF)             //eof=end of file
    {
            printf("Error in closing the file\n");
            return;
    }
}
```

File Input and Output Operations

Input Using fscanf():

- While the scanf() function reads input only from stdin that is from keyboard, the 'fscanf()' allows a program to receive input from either the keyboard or a file.
- The 'fscanf ()' function specifiesas the first parameter the source of the input, that can be either stdin or file.

Syntax:

```
fscanf(sourcefile, "format string", list of variables);
```

Where,

sourcefile- 'sourcefile' is either stdin or a pointer to a file which has already been opened.
format string and list of variables- 'format string' (%d%c%s) and 'list of variables' specifies which variable are to be read. i.e., the variables specified in the list will take the values from the source file that may be either through keyboard or from the file specified by fp using specifications provided in the format string.

Return values

- fscanf() returns EOF if it attempts to read at end-of-file,
- Otherwise it returns the number of items read in and successfully converted.

Examples:

1. fscanf (stdin, "%d", &rollno); and scanf("%d", &num);
   Both the statements are exactly equivalent.

2. fscanf(fp, "%d%s%f",&id,name,&salary);
   Suppose fp points to the source file. After executing the above statement, the values for the variables id, name and salary are obtained from the file associated with the file pointer fp. This function returns the number of items that are successfully read from the file.

Example Program:

1. Write a C program to demonstrate the usage of fscanf() function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        FILE *fp;
        int n;
        fp = fopen("student.txt", "r");
        if(fp==NULL)
        {
                printf("Error in opening the file\n");
                exit(0);
        }
        fscanf(fp, "%d", &n);
        fclose(fp);
        getch();
}
```

Output using fprintf():

- While printf() sends output to stdout (screen), the function fprintf () allows a program to send output to many different places: screen, printer or file on a disk.
- fprintf () specify as the first parameter, the destination of the output. This destination can be an I/O stream stdout or stderr or it can be a file.

Syntax:

```
fprintf (destfile, "format string", list of variables);
```

where,

destfile- 'destfile' is the name of an output stream (stdout or stderr) or a pointer to a file which has already been opened for writing.

format string and list of variables-'format string' (%d%c%s) and 'list of variables' specifies which variable to be printed. i.e. values of the variables specified in the list will be written into file associated with file pointer using specifications provided in the format string.

Return value

- This function returns the number of items that are successfully written into the file.

Examples:

1. fprintf (stdout, "%d\n",num); and printf("%d\n",num);

Both the statements are exactly equivalent. After executing fprintf, the value of the variable num is displayed on the standard output screen.

2. fprintf (fp, "%d%s%f",id,name,salary);

After executing fprintf(), the values of the variables id,name and salary are written into the file associated with file pointer fp.

Example programs:

1. **Write a C program to demonstrate the usage of fprintf().**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    fp = fopen("student.txt","r");
    if ( fp==NULL)
    {
        fprintf (stderr,"Error in Opening file\n");
        exit (0);
    }
    fclose(fp);
    getch();
}
```

'stderr' sends its output to the screen to alert the programmer to the error.

2. **Write a C program to open a file for input, reads in a series of numbers until end-of-file, and displays each number on the monitor.**

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

```
{
    FILE * fp;
    int num;
    fp = fopen("student.txt", "r");
    if ( fp==NULL)
    {
            printf (stderr,"Error in Opening file\n");
            exit (1);
    }
    while (fscanf (fp,"%d",&num)>0)
    {
            fprintf(stdout,"%d\n",num);
    }
    fclose (fp);
    }
```

- If the file does not exist, the program section prints an error message to 'stderr' and terminates with a zero exit code, including error also.
- If the file exists, the program reads numbers from the file, each time it reads a number, fscanf() returns 1,storing the value in 'num'.
- The call to 'fprintf()' inside the while loop prints the numbers. If there are no numbers in file or if it has read the last number, fscanf() returns EOF, which causes the loop to terminate.
- At the end of the program the program closes the opened file

3. Write a C program to read n numbers from the keyboard and write into a file

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE * fp;
    int num;
    fp=fopen("input.txt", "w");
    if(fp==NULL)
    {
        printf("Error in opening the file\n");
        exit(0);
    }
    while(scanf("%d",&num)!=EOF)
    {
        fprintf(fp, "%d\n",num);
    }
    fclose(fp);
    getch();
}
```

File I/O Functions for Strings

String Input from a file: fgets()

- fgets() reads characters from a file up to the maximum number of characters, which is specified as a parameter.
- fgets() function requires specifying the source of its input as a special case, 'stdin' may be that source or a file specified by a file pointer.

Syntax:

```
strptr = fgets (inputarea, n, source);
```

where,

inputarea-The parameter 'input area' is the name of the character array which is to get the data.

n- 'n' is an integer representing the size of the input field which is one more than the maximum number of characters to be read in.

source- 'source' identifies the file from which to read. It can be stdin or pointer to a file. File should already be open.

- 'fgets()' continues to read from 'source' until it has read 'n-1' characters or a new line character, whichever comes first.

Return value

- If the operation is successful, it returns a pointer to the string read in. The returned value is copied into strptr.
- Otherwise it returns NULL when it is unsuccessful.

Example:

```
char inarea[15];
char *fp;
fp = fgets (inarea, 15, stdin);
```

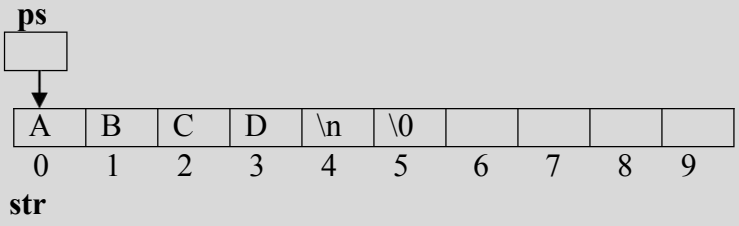
Example Program:

1. Write a C program to read a line from the keyboard using the function fgets().

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char str[15];
    char *ps;
    printf("Enter the string:\n");
    ps=fgets(str,10,stdin);
    if(ps!=NULL)
    {
            printf("The string is:");
            puts(str);
            return;
    }
    else
            printf("Reading is unsuccessful\n");
    }
```

Output:

Enter the string:
ABCD
The string is: ABCD



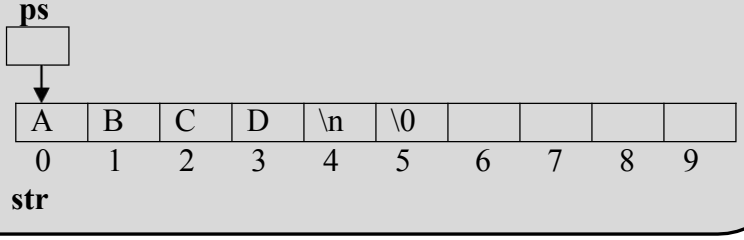
2. Write a C program to read a line from the file using the function fgets().

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char str[15];
    char *ps;
    fp=fopen("input.txt", "r");
    if(fp==NULL)
    {
            printf("Error in opening the file\n");
            exit(0);
    }
    ps=fgets(str,10,fp);
    if(ps!=NULL)
    {
            printf("The string is:");
            puts(str);
            return;
    }
    else
            printf("Reading is unsuccessful\n");
    getch( );
}
```

Output:
Contents of file "input.txt"

ABCD

The string is: ABCD



String output to a file: fputs()

- The **fputs( )** designed to print strings.

Syntax:

```
fputs (string, destfile);
```

- The 'fputs( )' function takes two parameters: a pointer to a string and destination of theoutput, which may be either an output stream (stdout or stderr) or pointer to a file.
- The 'fputs( )' function sends the string to the output destination. If the destination is apointer to a file, the file should already be open.
- The 'fputs( )' function writes a new line only if the output string contains a newline character.

Example: Write a C program to demonstrate the usage of fputs().

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char str[15];
    char *ps;
    printf("Enter few lines till EOF:\n");
    ps = fgets (str, 10, stdin);
    while (ps!=NULL)
    {
            fputs(str, stdout);
            ps = fgets (str, 10, stdin);
    }
    getch();
}
```

- Each string read in from stdin (keyboard) is sent to stdout (screen).

File I/O Functions for Characters:

fgetc( ):

Syntax:

```
ch = fgetc (source);
```

- fgetc( ) function reads a character from 'source', which is either stdin or a pointer to a file.
- If 'source' is a pointer to a file, the file should already be open.

Return vales

- The fgetc( ) function returns the character read in, converted to integer pointed to by fp ifsource is a file.
- On reaching end-of-file fgetc() returns EOF.

fputc( ):

Syntax:

```
fputc (ch, destfile);
```

- fputc() function takes two parameters: a character and the destination of the output, which may be either an output stream (stdout or stderr) or a pointer to a file.
- fputc() function sends the character to th output destination.
- If the destination is a pointer to a file, the file should already be open.

- This function writes a characterstored in ch to the stream(stdout) or t a file pointed by the file pointer fp. If there is any error or end-of-file is encountered the function returns EOF.

Example Programs:

1. **Write a C program to demonstrate the usage of fgetc() and fputc().**

```
#include<stdio.h>
void main()
{
        FILE * fp;
        int ch;
          fp = fopen("student.txt", "w");
        ch = fgetc (stdin);
         while (ch != EOF)
         {
              fputc (ch, fp);
              ch= fgetc (stdin);
          }
          fclose (fp);
}
```

2. **Write a C program to copy the contents from one file to another file.**

```
#include<stdio.h>
void main()
{
        FILE *fp1,*fp2;
        char file1[20],file2[20];
        char ch;
        clrscr();
        printf("Enter the source file name:\n");
        scanf("%s",file1);
        fp1=fopen("file1.txt", "r");
        if(fp1==NULL)
        {
                printf("Error in opening the file\n");
                exit(0);
        }
        printf("Enter the output file name:\n");
        scanf("%s",file2);
        fp2=fopen("file2.txt", "w");
        while(!feof(fp1))
        {
                ch=fgetc(fp1);
                fput(ch,fp2);
        }
        flcose(fp1);
        flcose(fp2);
}
```

3. **Write a C program to read and display a text from the file.**

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

```
        FILE * fp;
        char str[20];
        fp = fopen("file1.txt", "r");
        if(fp==NULL)
        {
                printf("Error in opening the file\n");
                exit(0);
        }
        while(fscanf(fp, "%s",str)!=EOF)
        {
                printf("%s",str);
        }
         fclose (fp);
         getch();
}
```

4. Given two text documentary files "Ramayana.in" and "Mahabharatha.in". Write a C program to create a new file "Karnataka.in" that appends the content of the file "Ramayana.in" to the file "Mahabharatha.in". Also calculate the number of words and new lines in the output file. Assume the following contents in the files:

"Ramayana.in"

Rama
Sita
Ravana
Bharatha

"Mahabharatha.in"

Arjuna
Krishna
Bhima
Karna

After executing the program the file **"Karnataka.in"** should contain the followingcontents:

"Karnataka.in"

Arjuna
Krisha
Bhima
Karna
Rama
Sita
Ravana
Bharat

```
#include<stdio.h>
#include<conio.h>
void main()
{
        FILE *fp1,*fp2,*fp3,*fp;
        int linecount=0,charcount=0,wordcount=0;
        char ch;
        char str1[20],str2[20],str[20];
        clrscr();
        fp1=fopen("Ramayana.in", "r");
        if(fp1==NULL)
        {
                printf("Error in opening the file\n");
```

```
            exit(0);
        }
        fp2=fopen("Mahabharatha.in", "r");
        if(fp2==NULL)
        {
            printf("Error in opening the file\n");
            exit(0);
        }
        fp3=fopen("Karnataka.in", "w");
        while(!feof(fp2))
        {
            fscanf(fp2, "%s",str2);
            fprintf(fp3, "%s",str2);
        }
        while(!feof(fp1))
        {
            fscanf(fp1, "%s",str1);
            fprintf(fp3, "%s",str1);
        }
        flcose(fp1);
        flcose(fp2);
        flcose(fp3);
        fp3=fopen("Karnataka.in", "r");
        while(!feof(fp3))
        {
            fscanf(fp3, "%s",str);
            printf("\n%s",str);
        }
        flcose(fp3);
        fp=fopen("Karnataka.in", "r");
        while((ch=getc(fp))!=EOF)
        {
            if(ch== ' '||ch== '\n'||ch== '\t')
                wordcount++;
            if(ch!= '\n'||ch!= '\t')
                charcount++;
            if(ch== '\n')
                linecount++;
        }
        printf("Number of words in the file is=%d\n",wordcount);
        printf("Number of characters excluding newline and tab space characters
        are=%d\n",charcount);
        printf("Number of lines are=%d",linecount+1);
        flcose(fp);
        getch();
}
```

Output:

Number of words in the file is = 8
Number of characters excluding newline and tab space characters are = 45
Number of lines are = 8