❖ File Handling Introduction

- Function scanf and printf are used for read and write data. These are console oriented I/O functions, which always use the terminal (keyboard and screen) as the target place.
- This work fine as long as the data is small.
- Many real life problems involve large volume of data and in such situations; the console oriented I/O operations pose two major problems.
    1. It becomes cumbersome and time consuming to handle large volume of data through terminals.
    2. The entire data is lost when either the program is terminated or the computer is turned off.
- It is therefore to have a more flexible approach where data can be stored on the disks and read whenever necessary, without destroying the data. The method employs the concept of files to store the data.
- **A file is a place on the disk where a group of related data is stored.** Like most other languages, c supports a number of functions that have the ability to perform basic file operations, which include
    - Naming a file,
    - Opening a file,
    - Reading a data from a file,
    - Writing a data to a file and
    - Closing a file.
- There are two distinct ways to perform the file operation in c.
- The first one is known as the low level I/O and uses UNIX system calls.
- The second method is referred to as the high-level I/O operation and uses function in C's standard I/O library.
- There are listed in table.

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.
getc()	Reads a character from a file.
putc()	Writes a character to a file.
fprintf()	Writes a set of data values to a file.
fscanf()	Reads a set of data values from a file.
getw()	Reads an integer from a file.
putw()	Writes an integer from a file.

There are many other functions. Not all of them are supported by all compilers.

❖ **Defining and Opening a File**

- If we want to store data in a file in the secondary memory, we must specify certain things about the file to the operating system. They include:
  1. File Name.
  2. Data Structure
  3. Purpose
- Filename is a string of characters that make up a valid filename for the operating system. It may contain two parts, a **primary name** and **an optional period** with the **extension**.

- **Examples:**

  Input.data
  Store
  Prog.c
  Student.c
  Text.out

- Data structure of a file is defined as **FILE** in the library of standard I/O function definitions.
- Therefore, all files should be declared as type FILE before they are used. **FILE** is a defined data type.
- When we open a file, we must specify what we want to do with the file. For Example, we may write data to the file or read the already existing data.
- Following is the general format for declaring and opening a file:

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

- **The first statement** declared the variable fp as a "pointer to the data type **FILE**" is a defined data type. File is structured that is defined in I/O library.
- **The second statement** opens the file named filename and assigns an identifier to the **FILE** type pointer fp. This pointer which contains all the information absolute the file is subsequently used as a communication link between the system and the program.
- The second statement also specifies the purpose of opening this file.
- The mode does this job. Mode can be one of the following.

r	Open the file for reading only.
w	Open the file for writing only.
a	Open the file for appending (or editing) data to it.

- Both the filename and mode are specified as strings.
- They should be enclosed in double quotation marks.
- When trying to open a file, one of the following things may happen:
  1. When the mode is 'writing', a file with the specified name is created if the file does not exist. The content is detected, if the file already exists.
  2. When the purpose is 'appending', the file is opened with the current contents safe. A file with the specified name is created if the file does not exist.

3. If the purpose is 'reading', and if it exists, then the file is opened with the current contents safe otherwise an error occurs.

- Consider the following statements:

```
FILE  *p1,  *p2;
p1  = fopen("data", "r");
p2  = fopen("results", "w");
```

- The file **data** is opened for reading and results is opened for writing.
- In case, the **result** file already exists, its content is detected and the file is opened as anew file.
- If **data** file does not exist, an error will occur.
- Many recent compilers include additional modes of operation. They include:

r+	The existing a file is opened to the beginning for reading and writing.
w+	same as w except both for reading and writing.
a+	same as a except both for reading and writing.

❖ **Closing a File**

- A file must be closed as soon as all operations on it have been completed.
- This ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken.
- It also prevents any accidental misuse of the file. In case, there is a limit to the number of files that can be kept open simultaneously, closing of unwanted files might help open the required files.
- Another instance where we have to close a file is when we want to reopen the same file in a different mode.
- The I/O library supports a function to do this for us. It takes the following form:

```
fclose (file_pointer);
```

- **T**his would close the file associated with the **FILE** pointer file_pointer. Look at the following segment of a program.

```
-----
-----
FILE    *p1, *p2;
p1 = fopen ("INPUT", "w");
p2 = fopen ("OUTPUT", "r");
-----
-----
fclose (p1);
fclose (p2);
```

- This program opens two files and closes them after all operation on them are completed.
- Once a file is closed, its file pointer can be reused for another file.
- As a matter of fact all files are closed automatically whenever a program terminates.

❖ **INPUT/OUTPUT Operations on Files**

- Once a file is opened, reading out of writing to it is accomplished using the statement I/O routines.

✓ **The getc and putc functions**

- The simplest file I/O functions are getcand **putc.**
- These are analogous to **getchar** and **putchar functions** and handle one character at a time.
- Assume that a file is opened with mode w and file pointer **fp1.** Then, the statement
  e.g putc(c, fp1);
- writes the character combined in the character variable **c** to the file associated with **FILE** pointer fp1.
- Similarly, getc is used to read a character from a file that has been opened in read mode. For example, the statement
  c = getc(fp2); //read character from the file whose file pointer is fp2.
- The file pointer moves by one character position for every operation of **getc** and **putc.**
- The **getc** will return an end-of-file maker EOF, when end of the file has been reached. Therefore, the reading should be terminated when EOF is encountered.

✓ **The getw and putw functions**

- 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 forms of **getw** and **putw** are:

```
putw(integer,fp);
getw(fp);
```

✓ **The fprintf and fscanf functions**

- Most compilers support two other functions, namely fprintf and fscanf, that can handle a group of mixed data simultaneously.
- The functions fprintf and fscanf perform I/O operations that are identical to the familiar printf and scanf functions, expect of course that they work on files.
- The first argument of these functions is a file pointer which specifies the file to be used. The general form of fprintf is

```
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.
- The list may include variables, constants and string.
- **Example:**

```
fprintf(f1,"%s %d %f",name,age,7.5);
```

- here, name is an array variable of type char and age is an int variable.
- The general format of fscanf is

```
fscanf(fp,"control string",list);
```

- This statement would cause the reading of the items in the list from the file specified by fp, according to the specifications contained in the control string.

- **Example:**

  fscanf(f2,"%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.

❖ **Differentiate :**

Printf	**Fprintf()**
printf is a C function to print a formatted string to the standard output stream which is the computer screen.	fprintf is a C function to print a formatted string to a file.
Formatted string and list of parameters are passed to printf function. **Syntax** printf("format", args);	File pointer, formatted string and list of parameters are passed to the fprintf function. **Syntax** fprintf(File *ptr, "format", args);

e.g
// simple print
#include<stdio.h>
void main()
{
printf("hello geeksquiz");
getch();
}
	e.g
	#include<stdio.h>
	void main()
	{
	int i, n=2;
	char str[50];
	//open file sample.txt in write mode
	FILE *fptr = fopen("sample.txt", "w");
	if (fptr == NULL)
	{
	printf("Could not open file");
	return 0;
	}
	for (i=0; i<n; i++)
	{
	puts("Enter a name");
	gets(str);
	fprintf(fptr,"%d.%s\n", i, str);
	}
	fclose(fptr);
	getch();
	}

Output Hello Word	Input: Reyansh Moksh **Output :** sample.txt file now having output as 0. Reyansh 1. Moksh
Getc()	**Getch()**

It reads a single character from a given input stream and returns the corresponding integer value (typically ASCII value of read character) on success. It returns EOF on failure.	getchar() reads from standard input
Syntax int getc(FILE *stream);	**Syntax** int getchar(void);

// Example for getc() in C
#include <stdio.h>
int main()
{
printf("%c", getc(stdin));
return(0);
}
Input: g (press enter key)
Output: g
	// Example for getchar() in C
	#include <stdio.h>
	int main()
	{
	printf("%c", getchar());
	return 0;
	}
	Input: g(press enter key)
	Output: g

Scanf()	**Fscanf()**
The function scanf() is used to read formatted input from stdin in C language. It returns the whole number of characters written in it otherwise, returns a negative value.	The function fscanf() is used to read the formatted input from the given stream in C language. It returns zero, if unsuccessful. Otherwise, it returns The input string, if successful.
syntax `int scanf(const char *characters_set)`	`Syntax` `int fscanf(FILE *stream_name, const char *set_of_characters)`

#include<stdio.h>
void main()
char s[20];
printf("Enter a string : ");
scanf("%s", s);
printf("\nEntered string : %s\n", s);
getch();
}
	#include<stdio.h>
	void main()
	{
	char str1[10];
	int year;
	FILE * fp;
	fp = fopen ("file.txt", "w");
	fputs("This is demo text!", fp);
	fscanf(fp, "%s", str1);
	printf("First word = \n%s\n", str1);
	fclose(fp);
	return(0);
	}

Enter a string : Peter! Entered string : Peter!	First word = This

Question Bank

	MCQ
1	f = fopen(filename, "r"); Referring to the code above, what is the proper definition for the variable f? (a) FILE F; (b) struct file f; (c) **FILE *f;** (d) int f;
2	Which one of the following is valid for opening a file for only reading? (a) fileOpen (filenm, "r"); (b) **fopen (filenm, "r");** (c) fileOpen (filenm, "ra"); (d) fileOpen (filenm, "read");
3	putc function is used to_____ (i) write characters to a file (ii) takes 2 parameters (iii) returns a character (iv) requires a file pointer (a) all are true (b) only I and ii are true (c) all are false (d) **only i,ii and iv are true**
4	By default, all the files are opened in _____mode . (a) binary (b) **text** (c) octal (d) decimal
5	_____is datatype of file pointer. (a) int (b) double (c) string (d) **FILE**
6	getc() returns EOF when_____ (a) End of files is reached (b) on error **(c) both a and b** (d) none of the above
7	When fopen() is not able to open a file, it returns_____ (a) EOF **(b) NULL** (c) one (d) zero
8	The mode _____ is used for opening a file for updating. (a) r (b) w **(c) a** (d) r+
	Short Questions.
1	List basic file operations performed on file.
2	Differentiate: printf and fprintf
3	Differentiate: scanf and fscanf
4	Differentiate: getc and getchar
6	List out different file modes.
7	Write down the syntax to open the text file both in read and write mode.
8	Explain any one file handling functions in brief. (Note: Ask any one file handling function from the list)
	Long Questions.
1	Explain fopen() in detail.
2	Explain flcose() with example.
3	Explain getc() and putc() with example.
4	Explain getw() and putw() with example.
5	Explain fprintf() and fscanf() with example.
6	Explain error handling during I/O operations.