LAB NO: 1 **Date:**

BASIC FILE HANDLING OPERATIONS

Objectives:

- Getting familiar with various file handling system calls.
- Ability to perform basic file operations**.**

Prerequisites:

- Knowledge of the C programing language.
- Knowledge of file pointers.

I. INTRODUCTION:

In any programming language it is vital to learn file handling techniques. Many applications will at some point involve accessing folders and files on the hard drive. In C, a stream is associated with a file.

A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a readymade structure. In C language, we use a structure pointer of file type to declare a file.

FILE *fp;

Table 1.1 shows some of the built-in functions for file handling.

Table 1.1: File Handling functions

Function	Description
fopen()	Create a new file or open an existing file
fclose()	Closes a file
getc()	Reads a character from a file
putc()	Writes a character to a file
fscanf()	Reads a set of data from a file
fprintf()	Writes a set of data to a file
getw()	Reads an integer from a file
putw()	Writes an integer to a file
fseek()	Set the position to desire point
ftell()	Gives current position in the file
rewind()	Set the position to the beginning point

1. **fopen()**: This function accepts two arguments as strings. The first argument denotes the name of the file to be opened and the second signifies the mode in which the file is to be opened. The second argument can be any of the following

Syntax: *fp = FILE *fopen(const char *filename, const char *mode);

Table 1.2: Various modes present in file handling

File Mode	Description
r	Opens a text file for reading.
w	Creates a text file for writing, if exists, it is overwritten.
a	Opens a text file and append text to the end of the file.
rb	Opens a binary file for reading.
wb	Creates a binary file for writing, if exists, it is overwritten.

ab	Opens a binary file and append text to the end of the file.

2. **fclose():**This function is used for closing opened files. The only argument it accepts is the file pointer. If a program terminates, it automatically closes all opened files. But it is a good programming habit to close any file once it is no longer needed. This helps in better utilization of system resources, and is very useful when you are working on numerous files simultaneously. Some operating systems place a limit on the number of files that can be open at any given point in time.

   Syntax: int fclose(FILE *fp);

3. **fscanf() and fprintf():** The functions fprintf() and fscanf() are similar to printf() and scanf() except that these functions operate on files and require one additional and first argument to be a file pointer.

   Syntax: fprintf(fileprinter,"format specifier",v1,v2,…);
   fscanf(filepointer,"format specifier",&v1,&v2,….);

4. **getc() and putc()**: The functions getc() and putc() are equivalent to getchar() and putchar() functions except that these functions require an argument which is the file pointer. Function getc() reads a single character from the file which has previously been opened using a function like fopen(). Function putc() does the opposite, it writes a character to the file identified by its second argument.

   Syntax: getc(in_file);

```
putc(c, out_file);
```

Note: The second argument in the putc() function must be a file opened in either write or append mode.

5. **fseek()**: This function positions the next I/O operation on an open stream to a new position relative to the current position.

Syntax: int fseek(FILE *fp, long int offset, int origin);

Here fp is the file pointer of the stream on which I/O operations are carried on, offset is the number of bytes to skip over. The offset can be either positive or negative, denting forward or backward movement in the file. Origin is the position in the stream to which the offset is applied, this can be one of the following constants:
SEEK_SET: offset is relative to beginning of the file
SEEK_CUR: offset is relative to the current position in the file
SEEK_END: offset is relative to end of the file

Binary stream input and output The functions fread() and fwrite() are a somewhat complex file handling functions used for reading or writing chunks of data containing NULL characters (‘\0’) terminating strings. The function prototype of fread() and fwrite() is as below :

```
size_t fread(void *ptr, size_t sz, size_t n, FILE *fp);
size_t fwrite(const void *ptr, size_t sz, size_t n, FILE *fp);
```

You may notice that the return type of fread() is size_t which is the number of items read. You will understand this once you understand how fread() works. It reads n items, each of size sz from a file pointed to by the pointer fp into a buffer pointed by a void pointer ptr which is nothing but a generic pointer. Function fread() reads it as a stream of bytes and advances the file pointer by the number of bytes read. If it encounters an error or endof-file, it returns a zero, you have to use feof() or ferror() to distinguish between these two. Function fwrite() works similarly, it writes n objects of sz bytes long from a location pointed to by ptr, to a file pointed to by fp, and returns the number of items written to fp.

II. SOLVED EXERCISE:

Write a C program to copy the contents of source file to destination file

Algorithm: CopyFileContents

Step 1: Enter the source filename.

Step 2: Check if the file exists. If NOT, display an error message and exit from the program.

Step 3: Enter the destination filename.

Step 4: Read each character from the source file and write into destination file using file pointers until EOF character is encountered in the source file.

Step 5: Stop

Program:

```
// Program to copy contents of source file to destination file
#include <stdio.h>
```

```
#include <stdlib.h> // For exit()
int main()
{
	FILE *fptr1, *fptr2;
	char filename[100], c;
	printf("Enter the filename to open for reading: \n");
	scanf("%s", filename);
	fptr1 = fopen(filename, "r");     // Open one file for reading
	if (fptr1 == NULL)
	{
		printf("Cannot open file %s \n", filename);
		exit(0);
	}
	printf("Enter the filename to open for writing: \n");
	scanf("%s", filename);
	fptr2 = fopen(filename, "w+");  // Open another file for writing
	c = fgetc(fptr1);                       // Read contents from file
	while (c != EOF)
	{
	fputc(c, fptr2);
	c = fgetc(fptr1);          }

	printf("\nContents copied to %s", filename);
	fclose(fptr1);
	fclose(fptr2);
```

```
    return 0;
}
```

Sample Input and Output

Enter the filename to open for reading: source.txt

Enter he filename to open for writing: destination.txt

Contents copied to destination.txt

III. LAB EXERCISES:

Write a 'C' program

1. To count the number of lines and characters in a file.
2. To print five lines of file at a time. The program prompts user to enter the suitable option. When the user presses 'Q' the program quits and continues when the user presses 'C'.
1. That gives user three chances to enter a valid file name.
2. To find the size of file using file handling function.
3. To compare the contents of two files.

IV. ADDITIONAL EXERCISES:

LAB No: 1

1. Write a C program that merges lines alternatively from 2 files and stores it in a resultant file.
2. Write a C program to collect statistics of a source file and display total number of blank lines, total number of lines ending with semicolon, total number of blank spaces.
3. Write a C program to reverse the file contents and store in another file.

LAB NO: 2 **Date:**

PRELIMINARY SCANNING APPLICATIONS

Objectives:

- To understand basics of scanning process.
- Ability to preprocess the input file so that it becomes suitable for compilation.

Prerequisites:

- Knowledge of the C programing language.
- Knowledge of file pointers and string handling functions.

I. INTRODUCTION

In this lab we mainly deal with preprocessing the input file so that it becomes suitable for scanning process. Preprocessing aims at removal of blank spaces, tabs, preprocessor directives, comments from the given input file.

II. SOLVED EXERCISE:

Write a program in 'C', that removes single and multiline comments from a given input 'C' file.

Algorithm: Removal of single and multiline comments

Step 1: Open the input C file in read mode.

Step 2: Check if the file exists. Display an error message if the file doesn't exists.

Step 3: Open the output file for writing.

Step 4: Read each character from the input file.

Step 5: If the character read is '/'

a. If next character is '/' then

i. Continue reading until newline character is encountered.

b. If the next character is '*' then
    i. Continue reading until next '*' is encountered.
    ii. Check if the next character is '/'

Step 6: Otherwise, write the characters into the output file.

Step 7: Repeat step 4, 5 and 6 until EOF is encountered.

Step 8: Stop

Program:

```
//Program to remove single and multiline comments from a given 'C' file.
#include <stdio.h>
int main()
{
        FILE *fa, *fb;
        int ca, cb;
        fa = fopen("q4in.c", "r");

                if (fa == NULL){
                printf("Cannot open file  \n");
                exit(0);}
        fb = fopen("q4out.c", "w");
        ca = getc(fa);
        while (ca != EOF)
        {
                if(ca==' ')
                {
                        putc(ca,fb);
                        while(ca==' ')
                                ca = getc(fa);
                }
```

```
        if (ca=='/')
        {
                cb = getc(fa);
                if (cb == '/')
                {
                        while(ca != '\n')
                                ca = getc(fa);
                }
                else if (cb == '*')
                {

                        do
                        {
                                while(ca != '*')
                                        ca = getc(fa);
                                ca = getc(fa);
                        } while (ca != '/');
                }
                else
                {
                        putc(ca,fb);
                        putc(cb,fb);
                }
        }
        else putc(ca,fb);
        ca = getc(fa);
}
fclose(fa);
fclose(fb);
```

```
 return 0;
 }
 }
```

Sample Input and Output

```
 / This is a single line comment
 /* *****This is a
 ******Multiline Comment
**** */
 #include <stdio.h>
  void   main()
 {
        FILE *fopen(),   *fp;
        int c ;
        fp = fopen( “prog.c”, “r” ); //Comment
        c = getc( fp ) ;
        while ( c     !=       EOF )
        {
              putchar( c );
              c = getc ( fp );
        }      /*multiline
        comment */
        fclose(      fp );
  }
```

Output file after the removal of comments:

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

```
void main(){
        FILE *fopen(), *fp;
        int c ;
        fp = fopen( "prog.c", "r" );
        c = getc( fp ) ;
        while ( c != EOF ){
                putchar( c );
                c = getc ( fp ); }
                    fclose( fp );
         }
```

III. LAB EXERCISES:

Write a 'C' program

1. That takes a file as input and replaces blank spaces and tabs by single space and writes the output to a file.
2. To discard preprocessor directives from the given input 'C' file.
3. That takes C program as input, recognizes all the keywords and prints them in upper case.

IV. ADDITIONAL EXERCISES:

1. Write a program to display the function names present in the given input 'C' file along with its return type and number of arguments.