C Programming Files

Why files are needed?

- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.

- If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.

- You can easily move your data from one computer to another without any changes.

Types of Files

1. Text files
2. Binary files

1. Text Files

Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.

2. Binary Files

Data is stored in the binary form (0's and 1's).

File Operations

In C, you can perform four major operations on the file, either text or binary:

- Creating a new file
- Opening an existing file
- Closing a file
- Reading from and writing information to a file

Working with files

- When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and program.

```
FILE *fptr;
```

Opening a file - for creation and edit

Header file: stdio.h

Syntax for opening a file in standard I/O is:

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

Examples:

```
fopen("E:\\cprogram\\newprogram.txt","w");
```

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.

r+	**Open for both reading and writing.**	**If the file does not exist, fopen() returns NULL.**
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be created.

Closing a File

- The file (both text and binary) should be closed after reading/writing.
- Closing a file is performed using library function fclose().

Syntax

- fclose(fptr); //fptr is the file pointer associated with file to be closed.

Reading and writing to a text file

```
FILE *fp1,*fp2,*fp;

putc(c,fp1);
// write the character in the variable c to  the file associated with fp1
c=getc(fp2);
//read a character from the file pointed by fp2.

putw(n,fp);
// write the integer in the variable n to  the file associated with fp
i=getw(fp);
//read integer from the file pointed by fp.

fprintf(fp, “format specifier”,list);
fscanf(fp, “format specifier”,list);
```

End of File

There are a number of ways to test for the end-of-file condition.

```
1. FILE *fptr1;
2. char s;
3. fptr=fopen("a.txt", "r");
4. while(fscanf(fptr, "%c", &s)!=EOF)
5. {
6.    printf ("%c", s) ;
7.  }
8.  printf("Reached end of the file");
```

```
1.  void main()
2.  {
3.      int num;
4.      FILE *fptr;
5.       if ((fptr = fopen("C:\\program.txt","r")) == NULL)
6.  {       printf("Error! opening file");
7.            exit(1); // Program exits if the file pointer returns NULL.
8.      }
9.       fscanf(fptr,"%d", &num);    //fgetc, fgetw, fgets
10.     printf("Value of n=%d", num);
11.    fclose(fptr);
12.     }
```

Writing to a text file

Example 1: Write to a text file using fprintf()

```
1. #include <stdio.h>
2. int main()
3. {
4.  int num;
5. FILE *fptr;
6. fptr = fopen("C:\\program.txt","w");
7. if(fptr == NULL)
8. {
9. printf("Error!");
10. exit(1);
11.}
12. printf("Enter num: ");
13. scanf("%d",&num);
14.fprintf(fptr,"%d",num);
15.fclose(fptr);
16. return 0;
17.}
```

fwrite

Declaration:

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE*stream);
```

ptr = Pointer to any object; the data written begins at ptr

size = Length of each item of data

n =Number of data items to be appended

stream = file pointer

Remarks:

fwrite appends a specified number of equal-sized data items to an output file.

Example:

```
1. #include <stdio.h>
2. int main()
3. {
4. char b[11]={'1','2','3','4','5','6','7','8','9','a','\0'};
5. FILE *fs;
6. fs=fopen("Project.txt","w");
7. fwrite(b,1,11,fs);
8. fclose(fs);
9. return 0;
10.}
```

fread

Declaration:

```
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
```

ptr = Points to a block into which data is read
size = Length of each item read, in bytes
n = Number of items read
stream = file pointer

Remarks:

fread reads a specified number of equal-sized data items from an input stream into a block.

Example:

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.     FILE *fptr;
5.     char b[11];
6.     if (f = fopen(“sample.txt", “r”))
7.     {
8.     fread(b, 1, 10, fptr);
9.     b[10] = ‘\0’;
10.    fclose(f);
11.    printf(“\n%s\n", b);
12.     }
13. return 0;
14.}
```

```
1. main()
2. {FILE *f1,*f2,*f3;
3. int num;
4. printf("Contents of DATA file\n");
5. f1=fopen("DATA", "w");
6. for(i=1;i<=30;i++)
7. {scanf("%d",&num);
8. if(num==-1) break;
9. putw(num,f1);
10.}
11.fclose(f1);
12.f1=fopen("DATA","r");
13.f2=fopen("ODD", "w");
14.F3=fopen("EVEN", "w");
15.while (num=getw(f1))!=feof(f1))
16.{if(num%2==00
17.putw(num,f3);
18. else
19. putw(num,f2);
20. }
21. fclose(f1); fclose(f2); fclose(f3);

22. f2=fopen("ODD", "r");
23. f3=fopen("EVEN", "r");

24. while (num=getw(f2))!=feof(f1))
25. printf("%d",num);

26. while (num=getw(f2))!=feof(f1))
27. printf("%d",num);

28. fclose(f2);
29. fclose(f3);
30. }
```

fseek

Declaration: `int fseek(FILE *stream, long int offset, int n) ;`

Parameters

- **stream** – This is the pointer to a FILE object that identifies the stream.
- **offset** – This is the number of bytes to offset from whence.
- **n** – This is the position from where offset is added. It is specified by one of the following constants –

SEEK_SET / 0	Beginning of file
SEEK_CUR / 1	Current position of the file pointer
SEEK_END / 2	End of file

This function returns zero if successful, or else it returns a non-zero value

```c
#include <stdio.h>
int main ()
{
FILE *fp;
fp = fopen("file.txt","w+");
fputs("This is an example", fp);
fseek( fp, 8, SEEK_SET );
fputs(" C Programming Language", fp);
fclose(fp);
return(0);
}
```

ftell

Declaration: long int ftell(FILE *fp)

Parameters

- **fp** − This is the pointer to a FILE object that identifies the stream.

This function returns the current value of the position indicator. If an error occurs, -1L is returned

```c
#include <stdio.h>
int main ()
{
FILE *fp; int len;
fp = fopen("file.txt", "r");
if( fp == NULL ) {
perror ("Error opening file");
return(-1);
}
fseek(fp, 0, SEEK_END);
len = ftell(fp);
fclose(fp);
printf("Total size of file.txt = %d bytes\n", len);
 return(0);
}
```