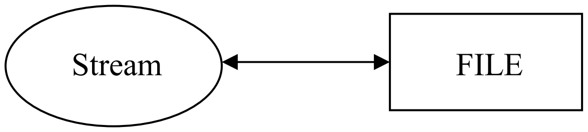
Lecture – 21

FILE:
Stream: In C the **stream** is a common logical interface to the various devices that comprises the computer. In it's most common form, a **stream** is a logical interface to a **file**. As C defines the term **file**, it can refer to a disk file, the screen, keyboard, a port, a file on tape and so on. Although files differ in form and capabilities but all streams are the same.

A **stream** is linked to a **file** using an open operation. A **stream** is disassociated from a **file** using a close operation.

Two types of stream:
- Text
- Binary



Text stream: A text stream is used with ASCII character, when a text stream is being used some character translation may take place (ex: new line is converted to CR/LF sequence). For this reason there may not be a one to one correspondence between what is sent to the stream and what is written to the file.

Binary stream: A binary stream may be used with any type of data. No character translation will occur and there is an one to one correspondence between what is sent to the stream and what is actually contained in the file.

- Text file stores lines of characters.
- Binary files contain arbitrary byte values.
- Binary files may be smaller.
- Text files tend to be large (take longer than binary files to read/write).
- Text files are quite portable.
- Binary files will not necessarily work if they are moved between machines having different work size and data formats.

To open a file and associate with a stream we will use the function **fopen(. . .)**

```
FILE *fopen( char *fname, char *mode)
```

Mode	**Meaning**
r	Open a text file for reading
w	Create a text file for writing
a	Append to a text file
rb	Open a binary file for reading
wb	Create a binary file for writing
ab	Append to a binary file
r+	Open a text file for read/write
w+	Create a text file for read/write
a+	Append a text file for read/write
r+b	Open a binary file for read/write
w+b	Create a binary file for read/write
a+b	Append a binary file for read/write

Example: Suppose we have to open a text file named “C:\student\test.txt” in read mode.

```
FILE *fp;
fp =  fopen( “C:\\student\\test.txt”, “r” );
```

Closing a file: A file must be closed as soon as all operations on it been completed. This ensures that all outstanding information associated with the file is flush 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);
```

This 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 operations 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. However, closing a file as soon as you have done with it is a good programming habit.

The **fclose(. . .)** function returns 0 if successful and return **EOF** if an error occurs. Once a file has been opened, depending upon its mode, you can read/write bytes by using these two functions.

```
int fgetc ( FILE *fp);
int fputc(int ch, FILE *fp);
```

The **fgetc(. . .)** function reads the next byte from the file described by *fp* as an unsigned char and returns it as an integer. It returns **EOF** if any error occurs or when the end of file reached. The **fputc(. . .)** function writes the byte contained in *ch* to the file associated with *fp* as an unsigned char if successful or **EOF** if an error occurs.

Example:

```
void main()
{
        char str[80] = "This is a test file";
        FILE *fp ;
        char *p, ch ;

        if((fp = fopen("myfile", "w")) = = NULL){
                printf("Cannot open file\n");
                exit(1);
        }

        p = str;

        while(*p){
                if(fputc(*p, fp) = = EOF){
                        printf("Error writing file\n");
                        exit(1);
                }
                p++;
        }
        fclose(fp);

        if((fp = fopen("myfile", "r")) = = NULL){
                printf("Cannot open file\n");
                exit(1);
        }

        while((ch = fgetc(fp)) ! = EOF )
                putchar(ch);
        fclose(fp);
}
```