FILE INPUT and OUTPUT

Files and streams

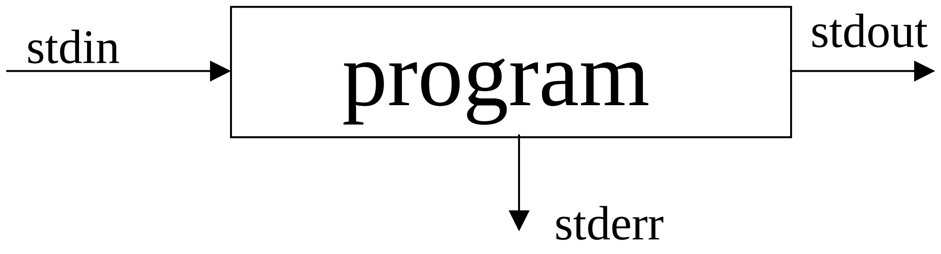
- C views each file simply as a sequential stream of bytes.
- Each file ends with an end-of-file (EOF) marker, which is a special character (for eg with ascii value –1)
- EOF is #defined in stdio.h
- For some programs it is possible to supply input from a file (instead of keyboard). Similarly, it is possible to write output to a file (instead of screen).
- For eg, `$ls > outfile`

  Output from `ls` command is written to the file named `outfile`.

Streams, files

- Similarly output of a program can be given as input to some other program.
- `$ls | wc -w`
- Output of ls is given as input to `wc -w`
- This is called pipe mechanism.

Streams



- Every program is automatically associated with three streams
  - stdin ➔ standard input. The program reads from this stream. This is by default mapped to keyboad.
  - stdout ➔ standard output. The program writes to this. This is screen by default.
  - stderr ➔ standard error. Error messages are written to this. This is also by default screen.

File access

- $cat x.c
- The program cat reads the file x.c and displays it on the screen.
- For doing this the program needs to open a input stream and associate it with the file x.c
- Then it has to read a character from the stream and write it to standard output, like this it has continue in a loop until EOF.

File access rules

- Before reading from a stream or writing to a stream, the stream has to be opened. For this a library function fopen(…) is used.
  - There is no need to open stdin, stdout, stderr. Because these are automatically opened and associated with each program.
- Prototype:
  - FILE *fopen(const char *name, const char *mode);
  - This is defined in stdio.h
  - This will associate a file name with a file pointer.
  - FILE can be seen as a separate data type. (just like int, but FILE is not a basic data type).

File opening and closing

- Similar to fopen there is fclose.
- int fclose(FILE *fp);
- This function closes the associated stream with fp. It returns EOF if any error occurred, and zero otherwise.
- When writing to a stream is completed, closing it guarantees that, the associated file is saved to the disk.
- When the program is completed, then automatically all open streams are closed by the operating system.
- But it is a good practice to explicitly close a stream.

FILE

- FILE *fp; /* says fp is a file pointer */

  fp = fopen(“x.c”, “r”);
- This results in opening the file x.c for reading, and fp is assigned with an address of variable whose type is FILE.
- If there is an error (like x.c may not exist at all!) then the function returns NULL.
- It is good to compare fp with NULL before reading from the stream. If fp is NULL, then reading or writing to that stream will cause a run-time error.

Reading from a file.

- FILE *fp;
  char c;
  fp = fopen(“x.c”, “r”);
  c = fgetc(fp);
- int fgetc(FILE *fp); returns the next character from the stream referred y fp; it return EOF for end of file or error.
- c = fgetc(stdin);

  This causes reading next character from standard input, that is keyboard (by default). So, *stdin* is actually a file pointer.

Modes

1)	“r”	→	open text file for reading
2)	“w”	→	create text file for writing; discard previous contents if any
3)	“a”	→	append; open or create text file for writing at end of file
4)	“r+”	→	open text file for update (that is reading and writing)
5)	“w+”	→	create text file for update; discard previous contents if any
6)	“a+”	→	append; open or create text file for update, writing at end

Common programming Errors

- Opening an existing file for writing (“w”) when, in fact, the user wants to preserve the file; the contents of the file are discarded without warning.
- Forgetting to open a file before reading/writing to it.
- Forgetting to close an opened file can cause some times losing the contents written to the file.
- Reading or writing using a file pointer whose value is junk or NULL.

An example --- displaying contents of a file

```
#include<stdio.h>
main( )
{
        FILE *fp;   char ch;
        fp = fopen(“x.c”, “r”);
        if(fp == NULL) {
                        puts(“error in opening file x.c”);
                        exit(1);
                       }
        while( (ch = fgetc(fp))  != EOF)
                putchar(ch);
        fclose(fp);
}
```

Some comments about the example

- It displays contents of file x.c
- If you want to see contents of file y.c then modify the source, recompile and execute it.
- It would be better if we can supply the file to be displayed as an input to the program.

The Example --- improved

```
#include<stdio.h>
main( )
{
        FILE *fp;   char ch, s[64];
        puts(“Enter the file to be displayed:”);
        gets(s);
        fp = fopen(s,  “r”);
        if(fp == NULL) {
                        puts(“error in opening file ”);
                        puts(s);
                        exit(1);
                       }
        while( (ch = fgetc(fp))  != EOF)
                putchar(ch);
        fclose(fp);
}
```

Can't we further improve ?

- The command `$cat x.c` will display contents of file `x.c`
- Can't we write a program so that `$a.out x.c` will display the contents of file `x.c`
- Yes, it is possible. For this we need to learn some thing called *command line arguments*.
- This will be discussed after a few classes.

Writing to a file

- int fputc(int c, FILE *fp);
- This writes the character c to the file for which fp is a pointer. It returns the character written or EOF for error.

Copying a file -- example

```
#include<stdio.h>
main( )
{
          FILE *fpr, *fpw;   char ch;
          fpr = fopen(“x.c”, “r”);
          fpw = fopen(“y.c”, “w”);
          if(fpr == NULL || fpw == NULL) {
                                   puts(“error in opening file x.c or y.c”);
                                   exit(1);
                                 }
          while( (ch = fgetc(fpr))  != EOF)
                    fputc(ch, fpw);
          fclose(fpr);  fclose(fpw);
}
```

Character input and output functions

- int fgetc(FILE *stream);
- char *fgets(char *s, int n, FILE *stream);
  - Reads at most the next (n –1) characters into the array s, stopping if a newline is encountered; the newline is included in the array, which is terminated by ‘\0’. fgets returns s, or NULL if end of file or error occurs.
- int fpuc(int c, FILE *stream);
- int fputs(const char *s, FILE *stream);
  - Writes the string s (which need not contain ‘\n’) on stream; it returns non-negative, or EOF for an error

Character input and output functions

- int getc(FILE *stream);
  - Is equivalent to fgetc except that it is a macro, it may evaluate the stream more than once.
- int putc(int c, FILE *stream);
- int getchar(void);
  - This is equivalent to getc(stdin).
- char *gets(char *s);
  - It reads the next input line into the array s; it replaces the terminating newline with ‘\0’. It returns s, or NULL if end of file or error occurs.
- int putchar(int c)   ⬄   putc(c, stdout)

Character input and output functions

- int puts(const char *s);
  - It writes the string s and a newline to stdout. It returns EOF if an error occurs, non-negative otherwise.
- int ungetc(int c, FILE *stream);
  - It pushes c back onto stream, where it will be returned on the next read. Only one character of pushback per stream is guaranteed. EOF may not be pushed back. ungetc returns the charater pushed back, or EOF for error.

Formatted Output and Input

- int fprintf(FILE *stream, const char *format, …);
  - Return value is the number of characters written, or negative if an error occurred.
- printf(…) ⇔ fprintf(stdout, …)
- int fscanf(FILE *stream, const char *format, …);
- scanf(…) ⇔ fscanf(stdin, …)

Some other operations

- FILE *freopen(const char *filename,
  const char *mode, FILE *stream);
  - Opens the file with the specified mode and associates the stream with it. It returns stream, or NULL if an error occurs. freopen is normally used to change the files associated with stdin, stdout, or stderr.
- int fflush(FILE *stream);
  - On an output stream, fflush causes any buffered but unwritten data to be written; on an input stream the effect is undefined. It returns EOF for a write error, and zero otherwise.

Some other operations

- int remove(const char *filename);
  - Removes the named file, so that the subsequent attempt to open it will fail. It returns non-zero if the attempt fails.
- int rename(const char *oldname,
                      const char *newname);
  - Changes the name of a file; it returns non-zero if the attempt fails.
- int feof(FILE *stream);
  - Returns non-zero if end of file occurred, zero otherwise.
- void rewind(FILE *stream);
  - The next read/write occurs from the beginning of the file.

Other …

- What we saw is text file operations. In contrast to this one there is some thing called binary files and their operations.
- For first level course, we do not go into binary files, random access operations, etc.