File Management in C

File Access

Program to read a file and print on screen

```
cat x.c y.c
```

Prints the contents of the files *x.c* and *y.c* on the standard output.

Concatenates a set of named files and prints it into the standard output.

We will see on Thursday how to write the `cat` program.

How to read named files from a program?

A file first has to be opened before reading from it or writing to it.

`fopen` opens a file.

`fopen` is a library function (the `stdio.h`).

`fopen` command

`fopen` takes an external name like *x.c* and returns a pointer to be used in subsequent reads or writes of the file.

```
FILE *fp                        // file pointer

fp = fopen(name, mode)  // call to fopen
```

File pointer

```
FILE *fp                    // file pointer

fp = fopen(name, mode)   // call to fopen
```

File pointer points to the structure (`FILE`) that contains information about the file.

`FILE` is a type name, like `int`. It is defined with `typedef`.

FILE declaration

For example a FILE declaration can look like this in stdio.h

```
typedef struct _iobuf{

        int cnt; /* characters left */

        char *ptr; /*next character position */

        char *base; /*location of buffer*/

        int flag; /*mode of file access*/

        int fd;   /* file descriptor*/ } FILE;
```

File pointer information

```
FILE *fp                        // file pointer
```

The file pointer has information about:

location of a buffer, the current character position in the buffer,

whether the file is being read or written, and

whether errors or end of file has occurred

fopen command arguments

```
FILE *fp                        // file pointer

fp = fopen(“x.c”, “r”)  // call to fopen
```

First argument of `fopen` is a character string containing the name of the file.

fopen command input arguments

```
FILE *fp                        // file pointer

fp = fopen(“x.c”, “r”)   // call to fopen
```

Second argument of `fopen` is also a character string; the string indicates how we are planning to use the file.

Some allowable modes: read (“r”), write (“w”), and append (“a”).

fopen command output return

```
FILE *fp                    // file pointer

fp = fopen(“x.c”, “r”)      // call to fopen
```

When executing above command, if *“x.c”* exist, it is opened for reading; fopen returns a *stream*.

For example a text *stream* is a sequence of lines; each lines has zero or more characters and is terminated by `‘\n’`.

fopen command output return

```
FILE *fp                    // file pointer

fp = fopen(“x.c”, “r”)  // call to fopen
```

When executing above command, if *“x.c”* does not exist, it is an error; fopen will return `NULL`.

fopen command output return

```
FILE *fp                    // file pointer

fp = fopen(“x.c”, “w”)   // call to fopen
```

When executing above command, if “*x.c*” exist, it causes the old content to be discarded; file is opened for writing.

fopen command output return

```
FILE *fp                          // file pointer

fp = fopen(“x.c”, “a”)   // call to fopen
```

When executing above command, if “*x.c*” exist, it preserves the old content and opens for appending.

fopen command output return

```
FILE *fp                     // file pointer

fp = fopen(“x.c”, “r”)   // call to fopen
```

When executing above command, if *“x.c”* does not exist, it is an error; `fopen` will return `NULL`.

There can be other causes for error as well. We will see this in error handling when accessing files.

Read from the file

```
/* filecopy: copy file ifp to file ofp
*/

void filecopy(FILE *ifp, FILE *ofp){

int c;

while ((c = getc(ifp)) != EOF)

   putc(c, ofp);

}
```

`getc` returns the next character from a file

The file pointer argument in the `getc` tells which file

Write to the file

```
/* filecopy: copy file ifp to file ofp
*/
void filecopy(FILE *ifp, FILE *ofp){
int c;
while ((c = getc(ifp)) != EOF)
    putc(c, ofp);
}
```

`putc` writes the character `c` to the file

Returns character written or EOF if an error occurs

Closing the file

```
FILE *fp                  // file pointer
fp = fopen(name, mode)    // call to fopen
fclose(fp)               // closes the file
```

Frees the file pointer; after the execution of the command no connection between the file pointer and the file name.

Closing the file

```
FILE *fp                        // file pointer
fp = fopen(name, mode)   // call to fopen
fclose(fp)                     // closes the file
```

Need for freeing the pointer: OS may have some limit on the number of files that are open simultaneously

Closing the file

```
FILE *fp                    // file pointer
fp = fopen(name, mode)      // call to fopen
fclose(fp)                  // closes the file
```

When a program terminates normally, fclose is called for each open file (that is, the files that the program opened during execution).