Programming & Data Structure

CS 11002

Partha Bhowmick

http://cse.iitkgp.ac.in/~pb

CSE Department
IIT Kharagpur

Spring 2012-2013

FILE HANDLING

File

A **file** is a named collection of data, stored in secondary storage (typically).

File Operations

- Open
- Read
- Write
- Close

File

How is a file stored?

- Stored as sequence of bytes, logically contiguous (may not be physically contiguous on disk).
- The last byte of a file contains the *end-of-file character* (`EOF`), with ASCII code `1A` (hex).
- While reading a text file, the `EOF` character can be checked to know the end.

File

Types of files

- `Text file`—contains ASCII codes only
- `Binary file`—may contain non-ASCII characters.
  Ex: Image, audio, video, executable, etc.
  To check the end of file here, the file size (also stored on disk) needs to be checked.

fopen

In C, we use `FILE *` to represent a *pointer to a file*.
The function `fopen` is used to open a file. It returns `NULL` when it is unable to open the file.

```
FILE *fptr;
char filename[] = "file2.dat";
fptr = fopen (filename,"w");
if (fptr == NULL) {
  printf ("ERROR IN FILE CREATION");
  /* DO SOMETHING */
}
```

fopen


```
fptr = fopen (filename,"w");
```

The second argument of `fopen` is the mode in which we open the file. There are three modes:

- `"r"` opens a file for reading.
- `"w"` creates a file for writing, and writes over all previous contents (so be careful!).
- `"a"` opens a file for appending, i.e., writing on the end of the file.

fopen

We can add a "b" character to indicate that the file is a *binary file*: "rb", "wb", or "ab".

Example

```
fptr = fopen("myImage.bmp", "rb");
```

exit

On error checking, we may need an *emergency exit* from a program. In `main()`, we can use `return` to stop. In functions, we can use `exit()` to do this. The function `exit` is defined in `stdlib.h` library.

`exit(-1);` in a function is exactly the same as `return -1;` in `main()`.

exit (2)

```
FILE *fptr;
char filename[]= "file2.dat";
fptr = fopen (filename,"w");
if (fptr == NULL) {
  printf("ERROR IN FILE CREATION");
  exit(-1);
}
...
```

fprintf()

fprintf() works just like printf() except that its first argument is a file pointer.

```
int a=10, b=100;
FILE *fptr;
fptr = fopen ("file.dat","w");
if (fptr == NULL) {
  printf("ERROR IN FILE CREATION");
  exit(-1);}
fprintf (fptr, "Hello World!\n");
fprintf (fptr, "%d %d", a, b);
```

fscanf()

We also read data from a file using `fscanf()`.

```
int a, b;
FILE *fptr;
fptr = fopen ("input.dat", "r");
if (fptr == NULL) {
  printf("ERROR IN FILE CREATION");
  exit(-1);
}
fscanf (fptr, "%d %d", &x, &y);
...
```

fgets()

Reading lines from a file using fgets()

```
FILE *fptr;
char line [1000];
fptr = fopen ("input.dat", "r");
if (fptr == NULL) {
  printf("ERROR IN FILE CREATION");
  exit(-1);
}
while (fgets(line,1000,fptr) != NULL)
  printf ("Read line %s\n",line);
```

fgets()

`fgets()` takes 3 arguments: a string, maximum number of characters to read, and a file pointer. It returns `NULL` if there is an error (e.g., `EOF`).

fclose()

We can close a file using `fclose()` and the file pointer.

```
FILE *fptr;
char filename[]= "myfile.dat";
fptr = fopen (filename,"w");
if (fptr == NULL) {
    printf ("Cannot open file to write!\n");
    exit(-1);}
fprintf (fptr,"New file created!\n");
fclose (fptr);
```

Special file streams (1)

Three special file streams are defined in `<stdio.h>`:

- `stdin` reads input from the keyboard
- `stdout` send output to the screen
- `stderr` prints errors to an error device (usually also the screen)

Special file streams (2)

```
#include <stdio.h>
main(){
 int i;
 fprintf(stdout,"Give value of i \n");
 fscanf(stdin,"%d",&i);
 fprintf(stdout,"Value of i=%d \n",i);
 fprintf(stderr,"No error:\nBut an example to
                 show error message.\n");
}
```

Special file streams (3)

```
Give value of i
15
Value of i=15
No error:
But an example to show error message.
```

Special file streams (4)

Input File & Output File Redirection

One may redirect the standard input and standard output to other files (other than `stdin` and `stdout`).

Usage: Suppose the executable file is `a.out`.

```
$./a.out <in.dat >out.dat
```

`scanf()` will read input data from the file `in.dat`, and `printf()` will print the result on the (newly created) file `out.dat`.

Special file streams


```
$./a.out <in.dat >>out.dat
```

`scanf()` will read input data from the file `in.dat`, and `printf()` will append the result at the end of the file `out.dat`.

Character read & write (1)

A character reading or writing is equivalent to reading or writing a *byte*.

With stdin and stdout:

```
int getchar();
int putchar(int c);
```

With files:

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

Character read & write (2)

```
#include <stdio.h>
main(){
  int c;
  printf("Type text and press return to
    see it again \n");
  printf("For exiting press <CTRL D> \n");
  while((c = getchar()) != EOF)
    putchar(c);
}
```

Command Line Arguments (1)

A program can be executed by directly typing a command at the $ prompt.

```
$./a.out in1.dat in2.dat out.dat
```

The individual items specified are separated from one another by spaces.
First item is the program name.
Variables `argc` and `argv` keep track of the items specified in the command line.

Command Line Arguments (2)

Command line arguments are passed by specifying them under `main()`:

```
int main (int argc, char *argv[]);
```

`argc` denotes *argument count*.

`*argv[]` denotes the array of strings as command line arguments, including the command itself.

Example

```
$./a.out s.dat d.dat
```

Here, `argc` = 3,

`argv[0]` = `"./a.out"`, `argv[1]` = `"s.dat"`,

`argv[2]` = `"d.dat"`.

Command Line Arguments (3)

```
#include <stdio.h>
#include <string.h>
int main(int argc,char *argv[]){
  FILE *ifp, *ofp;
  int i, c;
  char src_file[100], dst_file[100];

  if(argc!=3){
    printf("Usage: ./a.out
            <src_file> <dst_file> \n");
    exit(0);
  }
```

Command Line Arguments (4)

```
else{
  strcpy(src_file, argv[1]);
  strcpy(dst_file, argv[2]);
}

if ((ifp = fopen(src_file,"r")) == NULL){
  printf ("File does not exist.\n");
  exit(0);
}
```

Command Line Arguments (5)

```
  if ((ofp = fopen(dst_file,"w")) == NULL){
    printf ("File not created.\n");
    exit(0);
  }

  while ((c = fgetc(ifp)) != EOF)
    fputc (c,ofp);

  fclose(ifp);
  fclose(ofp);
}
```