Lecture 9
Basic File Operations

Dr. Hacer Yalım Keleş

Before it can be read or written, a file has to be opened by the library function **fopen.** fopen takes an external filename like 'input.txt' and does some housekeeping and negotiation with the operating system, and returns a pointer to be used in subsequent reads or writes of file.

This pointer is called a **file pointer**. It points to a structure that contains information about the file, such as the location of the buffer, the current character position in the buffer, whether the file is being read or written, and errors or end of file have occurred.

Definitions of the above information are included in **<stdio.h>** with a structure declaration called **FILE**.

```
FILE* fp;
FILE* fopen(char* name,char* mode);
```

This says that fp is a pointer to a FILE, and fopen returns a pointer to a FILE.

Note that, FILE is a typename (like int), not a structure tag; it is defined with a typedef.

```
FILE* fp;
fp = fopen(name, mode);
```

name: character string containing the name of the file.
mode: a character string which indicates how one intends to use the file. Allowable modes:

read: "r"
write: "w"
append: "a".
binary: append "b" to the mode string (in some Op. systems)

If a file that does not exist is opened for writing or appending, it is created if possible.

Opening an existing file for writing causes the old contents to be discarded.

Opening an existing file for appending preserves the original content of the file. All the writing is appended at the end of the file.

Trying to read a file that does not exist (or when you don't have permission to read) is an error.

If there is any error, fopen will return NULL.

Suppose we successfully opened a file, how do we read the content?

There are several possibilities to read the content of a file:

Simplest: getc and putc

getc: returns the next character from a file; it needs the file pointer to get data from. returns EOF for end of file or error.

```
int getc(FILE* fp);
```

How to write into a file?

Simplest: putc

int putc(int c, FILE* fp);

putc: writes a character c into the file fp and returns the character written, or EOF if an error occurrs.

When a C program is started, the OS environment is responsible for opening three files and providing file pointers for them:

stdin: standard input
stdout: standard output
stderr: standard error

The file pointers stdin, stdout and stderr are defined in <stdio.h>. Normally, stdin is connected to the keyboard and stdout and stderr are connected to the screen, but stdin and stdout may be redirected to files or pipes.

For example, a file may be substituted for the keyboard by using the < convention for input redirection.

```
prog < infile
```

causes prog to read characters from infile instead. The switching of the input is done in such a way that prog itself is unaware of the change; in particular the string "<infile" is not included in the command-line arguments in argv.

Similarly, output can be redirected to a file using > operator:

```
prog > outfile
```

will write the standard output to outfile instead of stdout.

getchar and putchar can be defined in terms of getc, putc, stdin, and stdout as follows:

```
#define getchar()  getc(stdin)
#define putchar(c)      putc((c),stdout)
```

For formatted input and output of files, the functions fscanf and fprintf may be used.

```
int fscanf(FILE* fp, char* format, ...)
int fprintf(FILE* fp, char* format, ...)
```

similar to printf() and scanf() functions, in which FILE pointers are defined as stdout and stdin respectively.

The file pointers stdin and stdout are objects of type FILE *. They are constants, however, not variables, so it is not possible to assingn to them.

int fclose(FILE* fp) --> closes the file.

For formatted input and output of files, the functions fscanf and fprintf may be used.

```
int fscanf(FILE* fp, char* format, ...)
int fprintf(FILE* fp, char* format, ...)
```

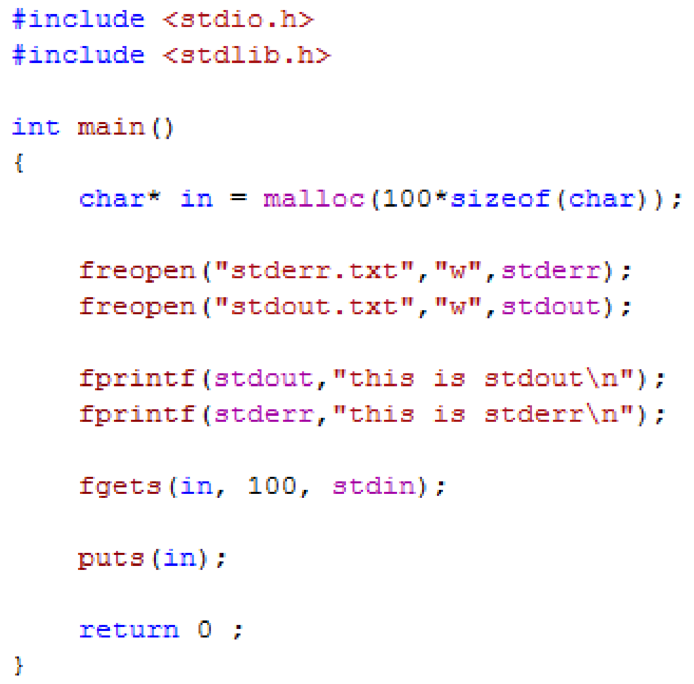
similar to printf() and scanf() functions, in which FILE pointers are defined as stdout and stdin respectively.

The file pointers stdin and stdout are objects of type FILE *. They are constants, however, not variables, so it is not possible to assingn to them.

int fclose(FILE* fp) --> closes the file.

You can close stdin and stdout if they are not needed.

They can also be reassigned by the library function **freopen**.



```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char* in = malloc(100*sizeof(char));

    freopen("stderr.txt","w",stderr);
    freopen("stdout.txt","w",stdout);

    fprintf(stdout,"this is stdout\n");
    fprintf(stderr,"this is stderr\n");

    fgets(in, 100, stdin);

    puts(in);

    return 0 ;
}
```

```c
#include <stdio.h>

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

int main(int argc, char** argv)
{
    FILE* fp;
    if( argc == 1) // no args. copy stdin
        filecopy(stdin,stdout);
    else
    {
        while (--argc > 0)
            if((fp=fopen(*++argv,"r")) == NULL)
            {
                printf("can't open %s\n",*argv);
                return 1;
            }
            else{
                filecopy(fp,stdout);
                fclose(fp);
            }
    }

    return 0;
}
```