EEE101: C Programming & Software Engineering I

Lecture 9: Files

Dr. Rui Lin/Dr. Mark Leach
Office: EE512/EE510
Email: rui.lin/mark.leach@xjtlu.edu.cn
Dept. of EEE XJTLU

Outline of Today's Lecture (9)

- Files – what are they?
- File operations:
  - Opening or creating a file
  - Writing to a file
  - Reading from a file
  - Accessing specific data (moving in the file)
  - Save/Read a structure to/from a file
  - Closing a file

What is a Computer File?

- A file is a **named** unit of storage
- It is a block of arbitrary information or resource for storing information which is available to a computer program.
- Files are stored on disk, DVD, USB etc.
- Computer files are the modern equivalent of paper documents

File Structure

- A file occupies a sequential block of memory on the media where it is stored
- Files end with an EOF (End-of-File) marker or at a byte number specified by the operating system
- There are two standard views of a file:
  - Binary
  - Text
- A file can be opened in either mode.

Binary vs. Text Mode/View

Binary Mode

- Each byte stored in the file can be accessed

Text Mode

- Bytes are interpreted as characters
- Reading in the text mode stops at the first occurrence of ^z (ctrl+z) even if there is more data
- Escape sequences (e.g. \n newline, \r carriage return) are seen as \n

Declaring a File

- The standard input output library <stdio.h> contains several file manipulation functions
- Declaring a file in C:

```
FILE *my_file
```

- Declares that my_file is a pointer to a FILE
- Each file must have a unique FILE pointer
- my_file is an internal file name used by the C program referring to the external file name which is the actual physical file stored on disk

Standard Files

Actually, the operation system uses files to handle ALL inputs and outputs.

Three files are automatically opened when a program starts to run (pointers are generated).

The files are:

File	**Pointer name**
standard input	stdin
standard output	stdout
standard error	stderr

These are all defined in stdio.h

Redirection of stdin/stdout

With most C systems, you can use redirection, either for all programs through the operating system or else just for C programs, courtesy of the C compiler.

In the following, let “prog” be the name of the executable program and let “file1” and “file2” be names of files. Executing on the command line:

- Redirecting Output filepath:>prog>file1
- Redirecting Input filepath:>prog<file2

Opening a file; fopen() (1/2)

Before file data can be processed, the file must be either **opened** or **created**

Syntax for fopen():

```
Internal_filename = fopen(external_filename, openmode);
```

The function fopen() takes 2 arguments:

- filename
- filemode

Opening a file; fopen() (2/2)

Example:

```
thefile = fopen(“my_file.txt”, “r”);
```

This statement will attempt to open a file called “my_file.txt”

If “my_file.txt” is located in a different directory than the current program directory. The full path needs to be included. For example “c:\documents\my_file.txt”

The function fopen() returns a pointer to the opened file, otherwise NULL is returned.

Open Modes for fopen()

- "r" - open for reading only
- "w" - create for writing; if a file by that name already exists, it will be overwritten
- "a" - append; open for writing at end-of-file or create for writing if the file does not exist
- "r+" - open an existing file for update from beginning (reading and writing)
- "w+" - create a new file for reading and writing; if a file by that name already exists, it will be overwritten
- "a+" - open for append; open (or create if the file does not exist) for update at the end of the file
- add a "b" to the end if you want to use binary files instead of text files, like this: "rb", "wb", "ab", "r+b", "w+b", "a+b"

fopen() Failure

If fopen() returns a NULL pointer, the operation failed.

This may be due to:

- Opening a non-existing file
- Opening a file for reading without access rights
- Opening a file for writing when no disk space is available

Hence programs should be written to handle this failure

Testing for failure

Check the value of the pointer returned by fopen()

```
FILE *fptr;             /*file pointer*/
fptr = fopen(“hello.txt”,”r”);
if(fptr == NULL){
        printf(“Failed opening Hello.txt”);
        exit(1);        /*exits program*/
        }
```

fgetc()

Once a file is open data can be read and wrote

fgetc() is a function that would read a character from a file.

Syntax

```
ch = fgetc(internal_filename);
```

ch – is a character variable **char**

Internal_filename – is the pointer to the open file

fputc()

fputc() is a function to write a character to a file

Syntax:

```
fputc(ch, internal_filename);
```

```
fputc(‘a’, internal_filename);
```

ch – is a character variable **char**

Internal_filename – is the pointer to the open file

Example program fputc()

```
FILE *pfile;                    /*file pointer pfile*/
char ch = ‘A’;
pfile = fopen(“hello.txt”,”w+”);           /*new file for r&w*/
fputc(ch,pfile);
rewind(pfile);                  /*rewind moves the pointer to*/
ch=fgetc(pfile);                /* the start of the file*/
printf(“%c”, ch);
```

What will be printed on the screen?

fscanf()

fscanf() is a function for reading a value or values of specified data types from a file.

Syntax:

```
fscanf(internal_filename, format_specifier(s),variable list);
```

Example:

```
FILE *fptr;
char name[50];
int age;
fptr = fopen(“filename.txt”,”r”);
fscanf(fptr, “%s %d”, name, &age);
```

Why does name not have &?

fprintf()

fprintf() is a function to print values to a file

Syntax:

```
fprintf(internal_filename, format_specifier(s), variable_expression);
```

Examples

```
fprintf(fptr, “This is an example”);
```

```
char name[] = “Peter”;
fprintf(fptr, “%s”, name);
```

```
char name[] = “Mary”;
fprintf(fptr, “%s %s”, “My name is”, name);
```

Example program fprintf()

```
FILE *pfile;                    /*file pointer pfile*/
char text[20];
pfile = fopen(“hello.txt”,”w+”);            /*new file for r&w*/
fprintf(pfile, “HELLO”);
rewind(pfile);                  /*rewind moves the pointer to*/
fscanf(pfile, “%s”, text);              /* the start of the file*/
printf(“%s”, text);
```

What will be printed on the screen?

fseek()

fseek() is a function that allows you to be able to access any particular position inside a file by specifying the amount in bytes to move from the starting_point which can be:

SEEK_SET – start of the file

SEEK_CUR – current position in the file

SEEK_END – end of the file

```
fseek(pfile, offset, starting_point);
```

Example program fseek()

```
FILE *pfile;                /*file pointer pfile*/
char ch = ‘A’;
pfile = fopen(“hello.txt”,”w+”);
fputc(ch, pfile);
fseek(pfile,-1, SEEK_END);               /*offset is type long*/
ch = getc(pfile);                    /*moves back 1 character*/
printf(“%c”,ch);
```

What will be printed on the screen?

ftell()

ftell () is a function that returns the current file position (number of bytes) from the beginning of the file in the type of a **long int**:

```
FILE *pfile;
char ch = ‘A’;
long position;
pfile = fopen(“hello.txt”,”w”);
fputc(ch, pfile);
position = ftell(pfile);
printf(“Current position %ld”, position);
```

What will be printed on the screen?

Read/Save Structure Contents in a File Using fread()/fwrite() (1/2)

- **struct**'s can hold many different variables
- Important tools for constructing databases. Needs to be saved and retrieved from, a file.
- Use fprintf() and fscanf() to read and write **struct** members.
- Use fread() and fwrite() to read and write structure sized units of data.

Read/Save Structure Contents in a File Using fread()/fwrite() (2/2)

Example data write:

pbooks is a file pointer

primer is a variable of type **struct** book

```
fwrite(&primer, sizeof(struct book), 1, pbooks);
```

The fread() function has the same arguments.
Copying a data block of size **struct** book into primer

Note: fread()/fwrite() are for binary files. Hence, fopen should use binary form. Use rewind() to ensure the file position pointer is at the start of the file.

EOF (End of File)

The EOF symbol is part of stdio.h, it marks the end of the file.

The function fgetc can be used to check when the EOF is reached.

fgetc() returns EOF if it tries to read a character but finds the EOF marker

```
if(fgetc(pfile) == EOF)
        printf("You've reached the end of the file");
```

Example program EOF test

```
FILE *fptr;
char ch = ‘A’;

fptr = fopen(“hello.txt”,”r”);

while((ch = fgetc(fprt))!=EOF)
        fscanf(fptr,  “%c”, &ch);
```

Can we use the EOF check with a binary file????

Closing the File fclose()

The function fclose() can be used to close the file:

- If the file is successfully closed **return** 0
- On failure, **return** EOF

Syntax

```
fclose(fptr);
```

Example program fclose()

```
#include<stdio.h>
main(){
FILE *pfile;
int age=23;
pfile = fopen(“hello.txt”,”w+”);
if(pfile == NULL){
          printf(”Cannot open file\n”);
          exit(1);}
fprintf(pfile, ”My age is %d\n”, age);
if(fclose(pfile) != 0)
          printf(“Error closing file\n”);
return 0;
}
```

Quiz

1.State whether the following statements are **true** or **false**

- A file must be opened before it can be used.
- All file must be explicitly closed.
- Files are always referred to by name in C programs.
- Function fseek may be used to seek from the beginning of the file only.
- Using fseek to position a file beyond the end of the file is an error.

Text and Binary

```
fscanf(fptrp,"%s%lf%d%lf%lf",
        planet.name,
        &planet.diameter,
        &planet.moons,
        &planet.orbit,
        &planet.rotation);

fprintf (fptro, "%s%e%d%e%e",
        planet.name,
        planet.diameter,
        planet.moons,
        planet.orbit,
        planet.rotation);
```

```
fread(&planet,
        sizeof(planet), 1,
        fptro);

fwrite(&planet,
        sizeof(planet), 1,
        fptrp);
```

Questions?

Week 10 is over…almost finished ☺