Power Engineering School

M.CS201 “Programming language”

Lecture 16

Lecturer:

Prof. Dr. T.Uranchimeg

Agenda

- ***Opening a File***
- ***Errors with open files***
- ***Writing and Reading File Data***
- ***Formatted File Input***
- ***Direct File Input and Output***

Opening a File

The process of creating a stream linked to a disk file is called *opening* the file. When you open a file, it becomes available for reading (meaning that data is input from the file to the program), writing (meaning that data from the program is saved in the file), or both. When you're done using the file, you must close it.

Prototype

To open a file, you use the fopen() library function. The prototype of fopen() is located in STDIO.H and reads as follows:

FILE *fopen(const char **filename*, const char **mode*);

This prototype tells you that fopen() returns a pointer to type FILE, which is a structure declared in STDIO.H.

Fopen()

When you call fopen(), that function creates an instance of the FILE structure and returns a pointer to that structure. You use this pointer in all subsequent operations on the file. If fopen() fails, it returns NULL.

Values of mode for the fopen() function

mode	Meaning
r	Opens the file for reading. If the file doesn't exist, fopen() returns NULL.
w	Opens the file for writing. If a file of the specified name doesn't exist, it is created. If a file of the specified name does exist, it is deleted without warning, and a new, empty file is created.
a	Opens the file for appending. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is appended to the end of the file.

Values of mode for the fopen() function

r+	Opens the file for reading and writing. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is added to the beginning of the file, overwriting existing data.
w+	Opens the file for reading and writing. If a file of the specified name doesn't exist, it is created. If the file does exist, it is overwritten.
a+	Opens a file for reading and appending. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is appended to the end of the file.

Mode argument

The default file mode is text. To open a file in binary mode, you append a b to the *mode* argument. Thus, a *mode* argument of a would open a text-mode file for appending, whereas ab would open a binary-mode file for appending.

Errors with open files

- Using an invalid filename.
- Trying to open a file on a disk that isn't ready (the drive door isn't closed or the disk isn't formatted, for example).
- Trying to open a file in a nonexistent directory or on a nonexistent disk drive.
- Trying to open a nonexistent file in mode r.

Using fopen() to open disk files in various modes

```
#include <stdlib.h>
#include <stdio.h>
main()
{
   FILE *fp;
   char filename[40], mode[4];
```

Cont. 2

```
while (1)
{
    printf("\nEnter a filename: ");
    gets(filename);
    printf("\nEnter a mode (max 3 characters): ");
```

Cont. 3

```
    gets(mode);
    if ( (fp = fopen( filename, mode )) != NULL )
    {
            printf("\nSuccessful opening %s in mode %s.\n", filename, mode);
```

Cont. 4

```
        fclose(fp);
        puts("Enter x to exit, any other to continue.");
        if ( (getc(stdin)) == `x')
        break;
        else
```

Cont. 5

```
            continue;
        }
        else
        {
            fprintf(stderr, "\nError opening file
    %s in mode %s.\n", filename, mode);
```

Cont. 6

```
            puts("Enter x to exit, any other to try again.");
            if ( (getc(stdin)) == `x')
            break;
            else
            continue;
            }
        }
}
```

Writing and Reading File Data

A program that uses a disk file can write data to a file, read data from a file, or a combination of the two. You can write data to a disk file in three ways:

1. Formatted output

You can use formatted output to save formatted data to a file. You should use formatted output only with text-mode files. The primary use of formatted output is to create files containing text and numeric data to be read by other programs such as spreadsheets or databases. You rarely, if ever, use formatted output to create a file to be read again by a C program.

2. Character output

You can use character output to save single characters or lines of characters to a file. Although technically it's possible to use character output with binary-mode files, it can be tricky. You should restrict character-mode output to text files. The main use of character output is to save text (but not numeric) data in a form that can be read by C, as well as other programs such as word processors.

3. Direct output

You can use direct output to save the contents of a section of memory directly to a disk file. This method is for binary files only. Direct output is the best way to save data for later use by a C program.

Read data from file

When you want to read data from a file, you have the same three options: formatted input, character input, or direct input. The type of input you use in a particular case depends almost entirely on the nature of the file being read. Generally, you will read data in the same mode that it was saved in, but this is not a requirement.

Formatted File Input and Output

Formatted file input/output deals with text and numeric data that is formatted in a specific way. It is directly analogous to formatted keyboard input and screen output done with the printf() and scanf() functions

Formatted File Output

Formatted file output is done with the library function fprintf(). The prototype of fprintf() is in the header file STDIO.H, and it reads as follows:

int fprintf(FILE *fp, char *fmt, ...);

The first argument is a pointer to type FILE. To write data to a particular disk file, you pass the pointer that was returned when you opened the file with fopen().

Demonstrates the fprintf() function

```
#include <stdlib.h>
#include <stdio.h>
void clear_kb(void);
main()
{
    FILE *fp;
```

Cont. 2

```
float data[5];
 int count;
 char filename[20];
 puts("Enter 5 floating-point numerical values.");
 for (count = 0; count < 5; count++)
```

Cont. 3

```
scanf("%f", &data[count]);
clear_kb();
puts("Enter a name for the file.");
gets(filename);
if ( (fp = fopen(filename, "w")) == NULL)
```

Cont. 4

```
{
    fprintf(stderr, "Error opening file %s.", filename);
    exit(1);
}
 for (count = 0; count < 5; count++)
```

Cont. 5

```
{
    fprintf(fp, "\ndata[%d] = %f", count, data[count]);
    fprintf(stdout, "\ndata[%d] = %f", count, data[count]);
}
```

Cont. 6

```
    fclose(fp);
    printf("\n");
    return(0);
}
void clear_kb(void)
{
```

Cont. 7

```
    char junk[80];
    gets(junk);
}
```

Formatted File Input

```
#include <stdlib.h>
#include <stdio.h>
void clear_kb(void);
main()
{
   float f1, f2, f3, f4, f5;
```

Cont. 2

```
FILE *fp;
if ( (fp = fopen("INPUT.TXT", "r")) == NULL)
{
    fprintf(stderr, "Error opening file.\n");
```

Cont. 3

```
    exit(1);
}
fscanf(fp, "%f %f %f %f %f", &f1, &f2,
&f3, &f4, &f5);
printf("The values are %f, %f, %f, %f,
and %f\n.", f1, f2, f3, f4, f5);
```

Cont. 4

```
    fclose(fp);
    return(0);
}
```

Character Input and Output

There are three character input functions: getc() and fgetc() for single characters, and fgets() for lines.

two character output functions: putc() and fputs().

Direct File Input and Output

The fwrite() library function writes a block of data from memory to a binary-mode file. Its prototype in STDIO.H is

int fwrite(void **buf*, int *size*, int *count*, FILE *fp);

The argument *buf* is a pointer to the region of memory holding the data to be written to the file. The pointer type is void; it can be a pointer to anything.

The fread() Function

The fread() library function reads a block of data from a binary-mode file into memory. Its prototype in STDIO.H is

int fread(void **buf*, int *size*, int *count*, FILE *fp);

The argument *buf* is a pointer to the region of memory that receives the data read from the file. As with fwrite(), the pointer type is void.

Using fwrite() and fread() for direct file access

```
#include <stdlib.h>
#include <stdio.h>
#define SIZE 20
main()
{
   int count, array1[SIZE], array2[SIZE];
```

Example

```
FILE *fp;
for (count = 0; count < SIZE; count++)
array1[count] = 2 * count;
if ( (fp = fopen("direct.txt", "wb")) ==
NULL)
{
```

Example

```
        fprintf(stderr, "Error opening file.");
        exit(1);
    }
    if  (fwrite(array1,  sizeof(int),  SIZE,  fp)
    != SIZE)
    {
```

Example

```
        fprintf(stderr, "Error writing to file.");
        exit(1);
    }
    fclose(fp);
    if ( (fp = fopen("direct.txt", "rb")) ==
    NULL)
```

Example

```
{
    fprintf(stderr, "Error opening file.");
    exit(1);
}
if (fread(array2, sizeof(int), SIZE, fp) !=
SIZE)
```

Example

```
{
    fprintf(stderr, "Error reading file.");
    exit(1);
}
fclose(fp);
for (count = 0; count < SIZE; count++)
```

Example

```
    printf("%d\t%d\n",          array1[count],
    array2[count]);
    return(0);
}
```

Detecting the End of a File

When reading from a text-mode file character-by-character, you can look for the end-of-file character. The symbolic constant EOF is defined in STDIO.H as -1, a value never used by a "real" character. When a character input function reads EOF from a text-mode stream, you can be sure that you've reached the end of the file.

Example

```
#include <stdlib.h>
#include <stdio.h>
#define BUFSIZE 100
main()
{
   char buf[BUFSIZE];
```

Example

```
char filename[60];
FILE *fp;
puts("Enter name of text file to display: ");
gets(filename);
if ( (fp = fopen(filename, "r")) == NULL)
{
```

Example

```
        fprintf(stderr, "Error opening file.");
        exit(1);
    }
    while ( !feof(fp) )
    {
        fgets(buf, BUFSIZE, fp);
```

Example

```
        printf("%s",buf);
    }
    fclose(fp);
    return(0);
}
```

Deleting a File

To delete a file, you use the library function remove(). Its prototype is in STDIO.H, as follows:

int remove(const char *filename);

The variable **filename* is a pointer to the name of the file to be deleted.

Renaming a File

The rename() function changes the name of an existing disk file. The function prototype, in STDIO.H, is as follows:

int rename(const char **oldname*, const char **newname*);

The filenames pointed to by *oldname* and *newname* follow the rules

Summary

- ***Opening a File***
- ***Errors with open files***
- ***Writing and Reading File Data***
- ***Formatted File Input***
- ***Direct File Input and Output***

Any questions?

Thank you for attention