CPNM Lecture 17 - File Handling

Mridul Sankar Barik

Jadavpur University

2022-23

Introduction

- Files are storage abstraction provided by the Operating Systems
- Applications generate / require information that need to be written to / read from secondary storage in form of data files

The FILE Structure

- A pointer to a structure of type FILE
- Contains information about the file
  - Name of the file
  - Status
  - Current position of the file
- Creates a buffer area where information is temporarily stored while being transferred between computer's memory and a data file

```
FILE *fp;
```

Text vs. Binary Mode

- Text mode
  - Host system may perform transformations on data written to or read from files; Ex- A new line may be converted to a line-feed/carriage-return pair
  - There may not be a one to one relationship between the characters that are written (or read) and those stored on the external device
  - Number of characters may not be the same as the number of characters that is stored on the external device
- Binary mode
  - No character translation occurs
  - An implementation defined number of null bytes may be appended to a binary stream

Opening a File

- `fopen()` function opens a file and returns the file pointer associated with that file

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

- `filename` is a string that make up a valid filename and may include a path specification
- `mode` is a string that determines how the file will be opened
- Returns `NULL` if error

File Modes I

- "`r`": Open an existing file for reading only
  - Fails if the file does not exist or the host system does not permit you to read
- "`w`": Open a new file for writing only
  - Always creates a file, if the file exists its old contents are discarded
- "`a`": Open an existing file for appending only
  - Creates the file if it does not exist, otherwise writes new data at the end of the existing file content

File Modes II

- "`r+`": Open an existing file for both reading and writing
- "`w+`": Open a new file for reading and writing only
  - If the file exists, it will be destroyed and a new file will be created
- "`a+`": Open an existing file for reading and appending.
  - If the file does not exist, a new file will be created
  - The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file
- The mode string can also include the letter 'b'

File Modes III

```
FILE * fp;
if((fp=fopen("test", "w"))==NULL){
    printf("Cannot open file\n");
    exit(0);
}
```

Closing a File

- `fclose()` function
  - Closes a file that was opened by a call to `fopen()`
  - Writes any data still remaining in the disk buffer to the file
    `int fclose(FILE *fp);`
  - Return value of zero signifies a successful close operation
  - Returns `EOF` if error

Character I/O

- Writing a character
  - `int fputc(int ch, FILE *fp);`
  - Where `fp` is the file pointer returned by `fopen()` and `ch` is the character to be output
  - Returns the character if successful, `EOF` otherwise
- Reading a character
  - `int fgetc(FILE *fp);`
  - Where `fp` is the file pointer returned by `fopen()`
  - Returns an integer containing the character in the low order byte, and high order byte set to zero
  - Returns `EOF` when the end of file has been reached

Using feof()

- `int feof(FILE *fp);`
- Returns true if the end of file has been reached; otherwise it returns zero
- `while(!feof(fp)) ch=fgetc(fp);`
- Can be used for both text and binary files

Example - Keyboard to Disk

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

int main(int argc, char *argv[]){
 FILE *fp;
 char ch;

 if(argc!=2){
  printf("You forgot to enter the filename\n");
  exit(0);
 }
 if((fp=fopen(argv[1], "w"))==NULL){
  printf("Cannot open file \n");
  exit(0);
 }
 ch=getchar();
 while(ch!='$'){
  fputc(ch, fp);
  ch=getchar();
 }

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

Example - Disk to Screen

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

int main(int argc, char *argv[]){
 FILE *fp;
 char ch;

 if(argc!=2){
  printf("You forgot to enter the filename\n");
  exit(0);
 }
 if((fp=fopen(argv[1], "r"))==NULL){
  printf("Cannot open file \n");
  exit(0);
 }
 ch=fgetc(fp);
 while(ch!=EOF){
  putchar(ch);
  ch=fgetc(fp);
 }

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

Example - File Copy I

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

int main(int argc, char *argv[]){
 FILE *in, *out;
 char ch;

 if(argc!=3){
  printf("You forgot to enter the filename\n");
  exit(0);
 }
 if((in=fopen(argv[1], "r"))==NULL){
  printf("Cannot open source file \n");
  exit(0);
 }
 if((out=fopen(argv[2], "w"))==NULL){
  printf("Cannot open destination file \n");
  exit(0);
 }

 do{
  ch=fgetc(in);
  if(feof(in))
   break;
  fputc(ch, out);
 }while(1);
 fclose(in);
 fclose(out);
 return(0);
}
```

String I/O

- `int fputs(const char *str, FILE *fp);`
  - If successful returns zero, EOF otherwise
- `char *fgets(char *str, int length, FILE *fp);`
  - Reads a string from the specified file until either a newline character is read or `length-1` characters have been read
  - If newline is read it will be part of the string
  - The resulting string will be `NULL` terminated

The rewind() Function

- Resets the file position pointer indicator to the beginning of the file specified as its arguments
- `void rewind(FILE *fp);`

Example - File Copy I

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

int main(void){
 FILE *fp;
 char str[80];

 if((fp=fopen("TEST", "w"))==NULL){
  printf("Cannot open file \n");
  exit(0);
 }
do{
  printf("Enter a string (CR to quit):\n");
  gets(str);
  strcat(str, "\n");
  fputs(str, fp);
 }while(*str!='\n');

 rewind(fp);
 while(!feof(fp)){
  fgets(str, 79, fp);
  printf(str);
 }
 return(0);
}
```

Erasing a File

- `remove()` function
- `int remove(const char *filename);`
- Returns zero if successful, otherwise a non-zero value

Flushing

- `int fflush(FILE *fp);`
- Writes the content of any buffered data to the associated file
- If `fp` is `NULL`, all opened files are flushed

fread() and fwrite()

- To read and write data types that are longer than 1 byte

```
size_t fread(void *buffer, size_t num_bytes,
                    size_t count, FILE *fp);
```

  - `buffer` is pointer to a region of memory that will receive the data from the file
  - `count` is number items read with each item being `num_byte` bytes in length
  - Returns the number of items read

```
size_t fwrite(const void *buffer, size_t num_bytes,
                    size_t count, FILE *fp);
```

  - buffer is pointer to the information that will be written to the file
  - Returns the number of items written

- Typically used for binary files
- Useful for reading and writing user defined data types, i.e. structures

Example I

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

struct temp{
 char name[30];
 int  age;
 char sub[10];
};

int main(void){
 FILE *fp;
 double d=12.23, e;
 int i=101, j;
 long l=123023, m;
 char str[10];
 char s[10];
 struct temp t1={"ABC XYZ", 18, "BCSE"}, t2;

 if((fp=fopen("test", "wb+"))==NULL){
  printf("Cannot open file \n");
  exit(0);
 }
```

Example II

```
 strcpy(str, "Ravi");

 fwrite(&d, sizeof(double), 1, fp);
 fwrite(&i, sizeof(int), 1, fp);
 fwrite(&l, sizeof(long), 1, fp);
 fwrite(str, sizeof(char), strlen(str), fp);
 fwrite(&t1, sizeof(struct temp), 1, fp);
 rewind(fp);

 fread(&e, sizeof(double), 1, fp);
 fread(&j, sizeof(int), 1, fp);
 fread(&m, sizeof(long), 1, fp);
 fread(s, sizeof(char), sizeof(s), fp);
 fread(&t2, sizeof(struct temp), 1, fp);

 printf("%f %d %ld %s\n", e, j, m, s);
 printf("%s %d %s\n", t2.name, t2.age, t2.sub);

 return(0);
}
```

fseek() and Random Access I/O

- `int fseek(FILE *fp, long int numbytes, int origin);`
- Sets the file position indicator numbytes distance away from `origin`
- `origin` can be one of the following
  - `SEEK_SET` Beginning of the file
  - `SEEK_CUR` Current position
  - `SEEK_END` End of file
- Returns zero if successful, a nonzero value otherwise

ftell()

- Determine the current location of the position indicator within a file
- `long int ftell(FILE *fp);`
- Returns -1 if failure

fprintf() and fscanf()

- ASCII formatted file I/O

```
int fprintf(FILE *fp, const char *control_string, ...);
int fscanf(FILE *fp, const char *control_string, ...);
```

- Not always efficient, extra overhead
- File content is human readable

Example I

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

struct temp{
 char name[30];
 int  age;
 char sub[10];
};

int main(void){
 FILE *fp;
 float d=12.23, e;
 int i=101, j;
 long l=123023, m;
 char str[10];
 char s[10];
 char c;
 struct temp t1={"ABC XYZ$", 18, "BCSE"}, t2;

 if((fp=fopen("test", "w"))==NULL){
  printf("Cannot open file \n");
  exit(0);
```

Example II

```
 }

strcpy(str, "Ravi");

 fprintf(fp, "%f%d%ld %s ", d, i, l, str);
 fprintf(fp, "%s %d %s", t1.name, t1.age, t1.sub);

 fclose(fp);

 if((fp=fopen("test", "r"))==NULL){
  printf("Cannot open file \n");
  exit(0);
 }

 fscanf(fp, "%f%d%ld%s", &e, &j, &m, s);
 fscanf(fp, "%[^$]s%c%d%s", t2.name, &c, &t2.age, t2.sub);
 printf("%f %d %ld %s", e, j, m, s);
 printf("%s %d %s\n", t2.name, t2.age, t2.sub);

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