File Handling
in C Programming Language

Sukalyan Som (SKS) [1]

[1]Department of Computer Science,
Barrackpore R S N College

27 septembre 2015

Outline

Outline

Introduction

Need for File in C

- It is difficult to deal with large volume of data through terminals. Hope we have seen some student database program as an example to this.

Introduction

Need for File in C

- It is difficult to deal with large volume of data through terminals. Hope we have seen some student database program as an example to this.
- Entire data is available only during the execution of the program.

Introduction

Need for File in C

- It is difficult to deal with large volume of data through terminals. Hope we have seen some student database program as an example to this.
- Entire data is available only during the execution of the program.

As a remedy we need some flexible approach where data can be stored on the disks and read whenever necessary.

Introduction

Need for File in C

- It is difficult to deal with large volume of data through terminals. Hope we have seen some student database program as an example to this.
- Entire data is available only during the execution of the program.

As a remedy we need some flexible approach where data can be stored on the disks and read whenever necessary.
This necessitates the use of **file** in C.

Introduction

Need for File in C

- It is difficult to deal with large volume of data through terminals. Hope we have seen some student database program as an example to this.
- Entire data is available only during the execution of the program.

As a remedy we need some flexible approach where data can be stored on the disks and read whenever necessary.
This necessitates the use of **file** in C. There are two approaches of performing file handling in C -

1. Low level file handling (using **system calls**)
2. High level file handling (using **built-in functions**)

General Skeleton a C Program with File Handling

General structure

```
precprocessor directives
int main()
{
FILE *fp;
fp = fopen("filename", "mode");
// File operation
fclose(fp);
return 0;
}
```

Discussion on "General Structure

- **FILE** FILE is a defined data type describing the data structure of a file.

Discussion on "General Structure

- **FILE** FILE is a defined data type describing the data structure of a file.
- ***fp*** A pointer of cast type FILE.

Discussion on "General Structure

- **FILE** FILE is a defined data type describing the data structure of a file.
- ***fp*** A pointer of cast type FILE.
- **fopen()** A function to open the file with **filename** specified in specified **mode**, if possible otherwise returns **NULL**.

Discussion on "General Structure

- **FILE** FILE is a defined data type describing the data structure of a file.
- ***fp*** A pointer of cast type FILE.
- **fopen()** A function to open the file with **filename** specified in specified **mode**, if possible otherwise returns **NULL**.
- **File activity** can be some File I/O operation and thereby somehow accessing the contents of the file.

Discussion on "General Structure

- **FILE** FILE is a defined data type describing the data structure of a file.
- ***fp*** A pointer of cast type FILE.
- **fopen()** A function to open the file with **filename** specified in specified **mode**, if possible otherwise returns **NULL**.
- **File activity** can be some File I/O operation and thereby somehow accessing the contents of the file.
- **fclose()** A function to close the file held by the FILE pointer emfp

File opening by fopen()

fopen() FILE *fopen(const char *filename, const char *mode) ; where,
filename – This is the C string containing the name of the file to be opened.
mode – This is the C string containing a file access mode. It includes -

1. **r** Opens a file for reading. The file must exist.

File opening by fopen()

fopen() FILE *fopen(const char *filename, const char *mode) ; where,
filename – This is the C string containing the name of the file to be opened.
mode – This is the C string containing a file access mode. It includes -

1. **r** Opens a file for reading. The file must exist.
2. **w** Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.

File opening by fopen()

fopen() FILE *fopen(const char *filename, const char *mode) ; where,
filename – This is the C string containing the name of the file to be opened.
mode – This is the C string containing a file access mode. It includes -

1. **r** Opens a file for reading. The file must exist.
2. **w** Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
3. **a** Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist.

File opening by fopen()

fopen() FILE *fopen(const char *filename, const char *mode) ; where,
filename – This is the C string containing the name of the file to be opened.
mode – This is the C string containing a file access mode. It includes -

1. **r** Opens a file for reading. The file must exist.
2. **w** Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
3. **a** Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist.
4. **r+** Opens a file to update both reading and writing. The file must exist.

File opening by fopen()

fopen() FILE *fopen(const char *filename, const char *mode) ; where,
filename – This is the C string containing the name of the file to be opened.
mode – This is the C string containing a file access mode. It includes -

1. **r** Opens a file for reading. The file must exist.
2. **w** Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
3. **a** Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist.
4. **r+** Opens a file to update both reading and writing. The file must exist.
5. **w+** Creates an empty file for both reading and writing.

File opening by fopen()

fopen() FILE *fopen(const char *filename, const char *mode) ; where,
filename – This is the C string containing the name of the file to be opened.
mode – This is the C string containing a file access mode. It includes -

1. **r** Opens a file for reading. The file must exist.
2. **w** Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
3. **a** Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist.
4. **r+** Opens a file to update both reading and writing. The file must exist.
5. **w+** Creates an empty file for both reading and writing.
6. **a+** Opens a file for reading and appending.

File closing by fclose()

- A file must be closed as soon as all file operations are performed.

File closing by fclose()

- A file must be closed as soon as all file operations are performed.
- It ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken.

File closing by fclose()

- A file must be closed as soon as all file operations are performed.
- It ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken.
- It prevents accidental misuse of the file.

File closing by fclose()

- A file must be closed as soon as all file operations are performed.
- It ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken.
- It prevents accidental misuse of the file.
- The general syntax for **fclose()** is
  int fclose(FILE *stream) ; where,
  stream – This is the pointer to a FILE object that specifies the stream to be closed. **return value** – This method returns zero if the stream is successfully closed. On failure, EOF is returned.

Outline

Input & Output Operations in a File

- Input and Output operations to/ from a file can be performed by using built-in functions.

Input & Output Operations in a File

- Input and Output operations to/ from a file can be performed by using built-in functions.
- The Output functions are
  - putc()
  - fputc()
  - fputs()
  - putw()
  - fwrite()
  - fprintf()

Input & Output Operations in a File

- Input and Output operations to/ from a file can be performed by using built-in functions.
- The Output functions are
  - putc()
  - fputc()
  - fputs()
  - putw()
  - fwrite()
  - fprintf()
- The Input functions are
  - getc()
  - fgetc()
  - fgets()
  - getw()
  - fread()
  - fscanf()

using putc()

- **Description** The C library function int putc(int char, FILE *fp) writes a character (an unsigned char) specified by the argument char to the specified file pointer fp and advances/ increments the file pointer.

using putc()

- **Description** The C library function int putc(int char, FILE *fp) writes a character (an unsigned char) specified by the argument char to the specified file pointer fp and advances/ increments the file pointer.
- This function returns the character written as an unsigned char cast to an int or EOF on error.

using getc()

- **Description** The C library function int getc(FILE *fp) gets a character (an unsigned char) from the specified stream and advances/ increments the position indicator i.e. the file pointer for the stream.

using getc()

- **Description** The C library function int getc(FILE *fp) gets a character (an unsigned char) from the specified stream and advances/ increments the position indicator i.e. the file pointer for the stream.
- **fp** is a pointer to FILE that must be opened in read mode.

using getc()

- **Description** The C library function int getc(FILE *fp) gets a character (an unsigned char) from the specified stream and advances/ increments the position indicator i.e. the file pointer for the stream.
- **fp** is a pointer to FILE that must be opened in read mode.
- This function returns the character read as an unsigned char cast to an int or EOF on end of file or error.

fgetc()

- fgetc gets a character from a stream

fgetc()

- fgetc gets a character from a stream
- **Declaration :**
  int fgetc(FILE *stream) ;

fgetc()

- fgetc gets a character from a stream
- **Declaration :**
  int fgetc(FILE *stream) ;
- **Remarks :**
  fgetc returns the next character on the named input stream. It is a function version of the getc macro.

fgetc()

- fgetc gets a character from a stream
- **Declaration :**
  int fgetc(FILE *stream) ;
- **Remarks :**
  fgetc returns the next character on the named input stream. It is a function version of the getc macro.
- **Return value :**
  - On success,
    fgetc returns the character read, after converting it to an int without sign extension.
  - On end-of-file or error,
    fgetc returns EOF.

fputc()

- fputc outputs a character to a stream

fputc()

- fputc outputs a character to a stream
- **Declaration :**
  int fputc(int c, FILE *stream) ;

fputc()

- fputc outputs a character to a stream
- **Declaration :**
  int fputc(int c, FILE *stream) ;
- **Remarks :**
  fputc outputs character c to the named stream.

fputc()

- fputc outputs a character to a stream
- **Declaration :**
  int fputc(int c, FILE *stream) ;
- **Remarks :**
  fputc outputs character c to the named stream.
- **Return Value :**
  - On success,
    fputc returns the character c.
  - On error, fputc returns EOF.

using fputs()

- fputs writes a string to a file

using fputs()

- fputs writes a string to a file
- **Declaration :** int fputs(const char *s, FILE *stream) ;

using fputs()

- fputs writes a string to a file
- **Declaration :** int fputs(const char *s, FILE *stream) ;
- **Remarks :**
  fputs copies the null-terminated string s to the given output stream. It does not append a newline character, and the terminating null character is not copied.

using fputs()

- fputs writes a string to a file
- **Declaration :** int fputs(const char *s, FILE *stream) ;
- **Remarks :**
  fputs copies the null-terminated string s to the given output stream. It does not append a newline character, and the terminating null character is not copied.
- **Return Value :**
  - On success :
    fputs returns the last character written.
  - On error :
    fputs returns EOF.

using fgets()

- fgets gets a string from a stream

using fgets()

- fgets gets a string from a stream
- **Declaration :** char *fgets(char *s, int n, FILE *stream) ;

using fgets()

- fgets gets a string from a stream
- **Declaration :** char *fgets(char *s, int n, FILE *stream) ;
- **Remarks :**
  fgets reads characters from stream into the string s. It stops when it reads either n - 1 characters or a newline character, whichever comes first.

using fgets()

- fgets gets a string from a stream
- **Declaration :** char *fgets(char *s, int n, FILE *stream) ;
- **Remarks :**
  fgets reads characters from stream into the string s. It stops when it reads either n - 1 characters or a newline character, whichever comes first. fgets retains the newline character at the end of s and appends a null byte to s to mark the end of the string.

using fgets()

- fgets gets a string from a stream
- **Declaration :** char *fgets(char *s, int n, FILE *stream) ;
- **Remarks :**
  fgets reads characters from stream into the string s. It stops when it reads either n - 1 characters or a newline character, whichever comes first. fgets retains the newline character at the end of s and appends a null byte to s to mark the end of the string.
- **Return Value :**
  - On success :
    fgets returns the string pointed to by s.
  - On error :
    On end-of-file or error, fgets returns null.

using putw()

- putw outputs an integer on a stream

using putw()

- putw outputs an integer on a stream
- **Declaration :**
  int putw(int w, FILE *stream) ;

using putw()

- putw outputs an integer on a stream
- **Declaration :**
  int putw(int w, FILE *stream) ;
- **Remarks :** putw outputs the integer w to the given stream. It does not expect (and does not cause) special alignment in the file.

using putw()

- putw outputs an integer on a stream
- **Declaration :**
  int putw(int w, FILE *stream) ;
- **Remarks :** putw outputs the integer w to the given stream. It does not expect (and does not cause) special alignment in the file.
- **Return value :**
  - On success - putw returns the integer w.
  - On error - putw returns EOF

using getw()

- getw gets an integer from stream

using getw()

- getw gets an integer from stream
- **Declaration :**
  int getw(FILE *stream) ;

using getw()

- getw gets an integer from stream
- **Declaration :**
  int getw(FILE *stream) ;
- **Remarks :**
  getw returns the next integer in the named input stream. It assumes no special alignment in the file. getw should not be used when the stream is opened in text mode.

using getw()

- getw gets an integer from stream
- **Declaration :**
  int getw(FILE *stream) ;
- **Remarks :**
  getw returns the next integer in the named input stream. It assumes no special alignment in the file. getw should not be used when the stream is opened in text mode.
- **Return Value :**
  - On success,
    getw returns the next integer on the input stream.
  - On error,
    getw returns EOF On end-of-file, getw returns EOF

using fprintf()

- fprintf sends formatted output to a stream.
- **Declaration :**
  int fprintf (FILE *stream, const char *format [, argument, ...]) ;
- **Remarks :**
  fprintf function do the following :
  1. Accept a series of arguments
  2. Apply to each argument a format specifier contained in the format string *format
  3. Output the formatted data (to the screen, a stream, stdin, or a string)

  These functions apply the first format specifier to the first argument, the second specifier to the second argument, the third to the third, etc., to the end of the format.

using fscanf()

Pretty much similar to scanf()

Outline

Command-line Arguments

The general structure of a Program with Command-line argument is

```
int main(int argc, char * argv[])
{
// body of main

return 0;
}
```

where, **argc** - is an integer that holds the no of command-line arguments.
argv - is an array of pointer to string that holds the command-line arguments.

Outline

Error Handling During File I/O

use of feof()

- feof() is a macro that tests if end-of-file has been reached on a stream.

use of feof()

- feof() is a macro that tests if end-of-file has been reached on a stream.
- **Declaration :**
  int feof(FILE *stream) ;

use of feof()

- feof() is a macro that tests if end-of-file has been reached on a stream.
- **Declaration :**
  int feof(FILE *stream) ;
- **Remarks :**
  1. feof is a macro that tests the given stream for an end-of-file indicator.
  2. Once the indicator is set, read operations on the file return the indicator until rewind is called, or the file is closed.
  3. The end-of-file indicator is reset with each input operation.

use of feof()

- feof() is a macro that tests if end-of-file has been reached on a stream.
- **Declaration :**
  int feof(FILE *stream) ;
- **Remarks :**
  1. feof is a macro that tests the given stream for an end-of-file indicator.
  2. Once the indicator is set, read operations on the file return the indicator until rewind is called, or the file is closed.
  3. The end-of-file indicator is reset with each input operation.
- **Return Value :**
  1. Returns non-zero if an end-of-file indicator was detected on the last input operation on the named stream.
  2. Returns 0 if end-of-file has not been reached.

use of ferror()

- Macro that tests if an error has occurred on a stream

use of ferror()

- Macro that tests if an error has occurred on a stream
- **Declaration :**
  int ferror(FILE *stream) ;

use of ferror()

- Macro that tests if an error has occurred on a stream
- **Declaration :**
  int ferror(FILE *stream) ;
- **Remarks :**
  ferror is a macro that tests the given stream for a read or write error. If the stream's error indicator has been set, it remains set until clearerr or rewind is called, or until the stream is closed.

use of ferror()

- Macro that tests if an error has occurred on a stream
- **Declaration :**
  int ferror(FILE *stream) ;
- **Remarks :**
  ferror is a macro that tests the given stream for a read or write error. If the stream's error indicator has been set, it remains set until clearerr or rewind is called, or until the stream is closed.
- **Return Value :**
  ferror returns non-zero if an error was detected on the named stream.

use of fp == NULL

Outline

use of ftell()

- Returns the current file pointer

use of ftell()

- Returns the current file pointer
- **Declaration :**
  long ftell(FILE *stream) ;

use of ftell()

- Returns the current file pointer
- **Declaration :**
  long ftell(FILE *stream) ;
- **Remarks :**
  1. ftell returns the current file pointer for stream.
  2. If the file is binary, the offset is measured in bytes from the beginning of the file.
  3. The value returned by ftell can be used in a subsequent call to fseek.

use of ftell()

- Returns the current file pointer
- **Declaration :**
  long ftell(FILE *stream) ;
- **Remarks :**
  1. ftell returns the current file pointer for stream.
  2. If the file is binary, the offset is measured in bytes from the beginning of the file.
  3. The value returned by ftell can be used in a subsequent call to fseek.
- **Return Value :**
  On success, returns the current file pointer position.
  On error, returns -1L and sets errno to a positive value.

use of rewind()

- Repositions file pointer to beginning

use of rewind()

- Repositions file pointer to beginning
- **Declaration :**
  void rewind(FILE stream);

use of rewind()

- Repositions file pointer to beginning
- **Declaration :**
  void rewind(FILE stream);
- **Remarks :**
  - rewind(stream) is equivalent to fseek except that rewind clears the EOF and error indicators, while fseek only clears the end-of-file indicator.
- **Return Value :**
  None

use of fseek()

Outline

Renaming with rename()

- Renames a file

Renaming with rename()

- Renames a file
- **Declaration :**
  int rename(const char *oldname, const char *newname) ;

Renaming with rename()

- Renames a file
- **Declaration :**
  int rename(const char *oldname, const char *newname) ;
- **Remarks :**
  1. rename changes the name of a file from oldname to newname.
  2. If a drive specifier is given in newname, the specifier must be the same as that given in oldname.
  3. Directories in oldname and newname do not need to be the same, so rename can be used to move a file from one directory to another.
  4. Wildcards are not allowed.
- **Return Value :**
  On success
  returns 0
  On error, returns -1

Deletion with unlink

- Deletes a file

Deletion with unlink

- Deletes a file
- **Declaration :**
  int unlink(const char *filename) ;

Deletion with unlink

- Deletes a file
- **Declaration :**
  int unlink(const char *filename) ;
- **Remarks :**
  1. unlink deletes a file specified by filename. Any DOS drive, path, and file name can be used as filename.
  2. Wildcards are not allowed.
  3. Read-only files can not be deleted by this call. To remove read-only files, first use chmod to change the read-only attribute.
  4. If your file is open, be sure to close it before unlinking it.
- **Return Value :** On success
  unlink returns 0
  On error, it returns -1

Deletion with remove

- Macro that removes a file

Deletion with remove

- Macro that removes a file
- **Declaration :**
  int remove(const char *filename) ;

Deletion with remove

- Macro that removes a file
- **Declaration :**
  int remove(const char *filename) ;
- **Remarks :**
  1. remove deletes the file specified by filename.
  2. It is a macro that simply translates its call to a call to unlink.
  3. If your file is open, be sure to close it before removing it.
  4. The string *filename can include a full DOS path.
- **Return Value :**
  On success, remove returns 0
  On error, it returns -1

Outline

Thank You

For

Your Attention