Problem Solving With C - UE24CS151B

File Handling in C

Prof. Sindhu R Pai
PSWC Theory Anchor, Feb-May, 2025
Department of Computer Science and Engineering

PROBLEM SOLVING WITH C

File Handling

1. Points to Discuss
2. Introduction
3. Need of Files
4. File Classification
5. Operations on Files
6. Read/Write Operations on Files

Points to Discuss!!!

- Can we use the data entered by the user in one program execution, in another program execution without asking the user to enter again?
- How to generate same input several times?
- How to store the output produced for future references?
- Program stores the result what if one wants to store other data too?
- How to categorize and manage forms of data produced?

Introduction

- Variables used in program will die at the end of execution
- To persist data even after the program execution is complete, use Files
- File represents a sequence of bytes and it is a source of storing information in sequence of bytes on a disk
- The data in a file can be structured or unstructured
- A program in C can itself be the data file for the program
- The keyboard and the output screen are also considered as files

Need of Files

- When a program is terminated, the entire data is lost. Storing in a data file will preserve the data even if the program terminates.
- If the data is too large, a lot of time spent in entering them to the program every time the code is run.
- If stored in a data file, easier to access the contents of the data file using few functions in C

File Classification

- **Text File**
  - Contains textual information in the form of alphabets, digits and special characters or symbols
  - Created using a text editor
- **Binary File**
  - Contain bytes or a compiled version of a text file i.e. data in the form of 0’s and 1’s
  - Can store larger amount of data that are not readable but secured.

Operations on Files

- Reading the contents of the file
- Writing the contents to the file

Note: To perform any operation on Files, **The physical filename, the logical filename and the mode** must be connected using **fopen()**

- **Physical Name:** A file is maintained by the OS. The OS decides the naming convention of a file
- **Logical Name**: In a C Program, identifier is used to refer to a file. Also called as **File Handle**
- **Mode:** Can be read only, write only, append or a combination of these.

C Functions to perform Operations

- Creation of new file or open an existing file using **fopen()**
- Read operation on a file using **fgetc(), getc(), fscanf()** and **fgets()**
- Write operation on a file using **fputc(),putc(), fprintf()** and **fputs()**
- Moving to a specific location in a file **using fseek(), ftell()** and **rewind()**
- Closing a file using **fclose()**

fopen()

Syntax: fopen(“path of the file with filename”, “mode”);
// mode can be r, w, a, r+, w+, a+

- Opens a file in the specified mode and returns a FILE pointer(address of the structure which contains information about the attributes of the file) if it succeeds. Else returns NULL.

- The reasons for failure in file opening:
  - File might not be available
  - File might not have permission to open it in the mode specified.

- Initializing a FILE pointer does not mean that the whole file is made available in memory. Other functions are required to access the contents

fclose()

- Closes the stream. All buffers are flushed
- Returns zero if the stream is successfully closed. On failure, EOF is returned
- All links to the file are broken
- Misuse of files is prevented

Syntax: `fclose(file_pointer);`

- File pointer can be reused
- Coding Examples

Read /Write operations on File

- Categories
  - **Character read/write**
    - fputc() , fgetc(), getc() and putc().
  - **String read/write**
    - fgets() and fputs().
  - **Formatted read/write**
    - fscanf() and fprinft().
  - **Block read/write**
    - fread() and fwrite()

Character I/O operations on File

- Reads a character from the file and increments the file pointer position.
  - **Syntax**: **int fgetc(FILE *fp);**
  - **Return Value:**Next byte from the input stream on success, EOF on error.
- Write operation at current file position and increments the file pointer position.
  - **Syntax**: **int fputc(int c,FILE *fp);**
  - **Return Value:**Character that is written on success, EOF on error.
- Coding Examples

String I/O Operations on File(fgets() and fputs())

- Reads a line of characters from file and stores it into the string pointed to by char_array variable. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first
  - **Syntax:** char* fgets(char *char_array, int n, FILE *stream)
  - **Return Value:** Pointer to the string buffer on success, NULL on EOF or Error.
- Write a line of characters to a file.
  - **Syntax:** int fputs(const char *s, FILE *stream)
  - **Return Value:** A non-negative number on success, EOF on error.
- Coding examples

Formatted Read/Write Operations on File(fscanf(),fprintf())

- Reads the formatted data from the file instead of standard input.
  - **Syntax:** int fscanf(FILE *fp,const char *format[,address,…..]);
  - **Return Value:** The number of **values read** on success ,**EOF** on failure.
- Writes the formatted data to a file instead of standard output.
  - **Syntax:** int fprintf(FILE *fp, const char *format[,argument,….]);
  - **Return value:** The number of **characters written** on success , **EOF** on failure.
- Coding Examples

Block Read/Write Operations on a File

- Reads an entire block from a given file.
  - **Syntax:** **size_t fread(void *p, size_t size, size_t n, FILE *fp);**
  - **Return Value:** The number of values successfully read , error or zero for size <=0.
- Writes an entire block to the file.
  - **Syntax:** **size_t fwrite(const void *p, size_t size, size_t n, FILE *fp);**
  - **Return Value:** The number of values successfully written, error   or zero for size <=0.

Random access to a file

- Seeking the pointer position in the file at the specified byte.
  - **Syntax:** fseek(FILE* pointer, long offset, int whence)
  - **Return Value:** zero if successful, or else it returns a non-zero value.
- The function returns the current pointer position ,value from the beginning of file.
  - **Syntax:** ftell(FILE* pointer)
  - **Return Value:** 0 or a positive integer on success and -1 on error.
- The function is used to move the file pointer to the beginning
  - **Syntax:** rewind(FILE* pointer) // Function does not return anything.

THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CCBD & CDSAML, PESU
Prof. Sindhu R Pai - sindhurpai@pes.edu