Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

C Programming

File I/O in C

Education and Research

Unit Plan

- Understand the need for File I/O
- Classification of File I/O Functions
- Understanding Streams
- Various Functions for High Level I/O
  - Functions for character I/O, string I/O and block I/O
- Command Line Arguments

Need for File I/O

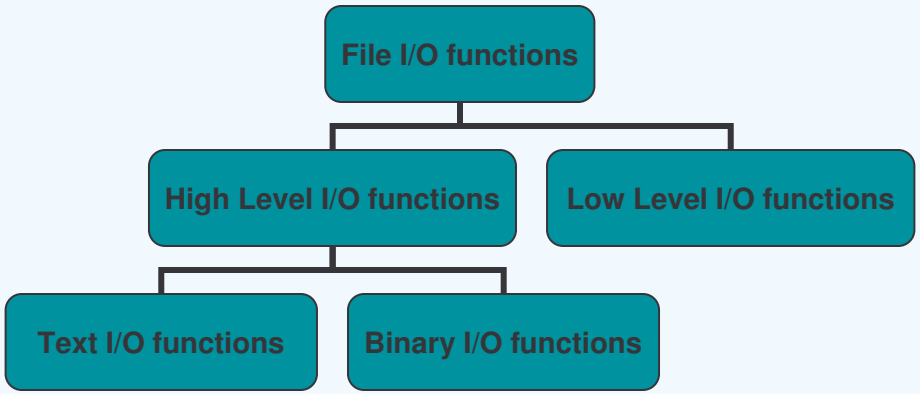
- Data that outlives the program needs to be stored in a permanent storage so that it can be referred later.

- For example, a word processing application might save the text in a linked list or some other data structure when the application is running, but when the application is closed the contents of the linked list need to stored in a file.

Need for File I/O

- The C Language has no provision for file I/O. The developers of a C compilers write several functions for I/O and make it available as a I/O library.
- The functions in the I/O library take assistance from the corresponding I/O function's provided by the underlying operating system.
- These functions in the I/O library are not a part of C's formal definition but they have become a standard feature of C Language.

Classification of File I/O functions

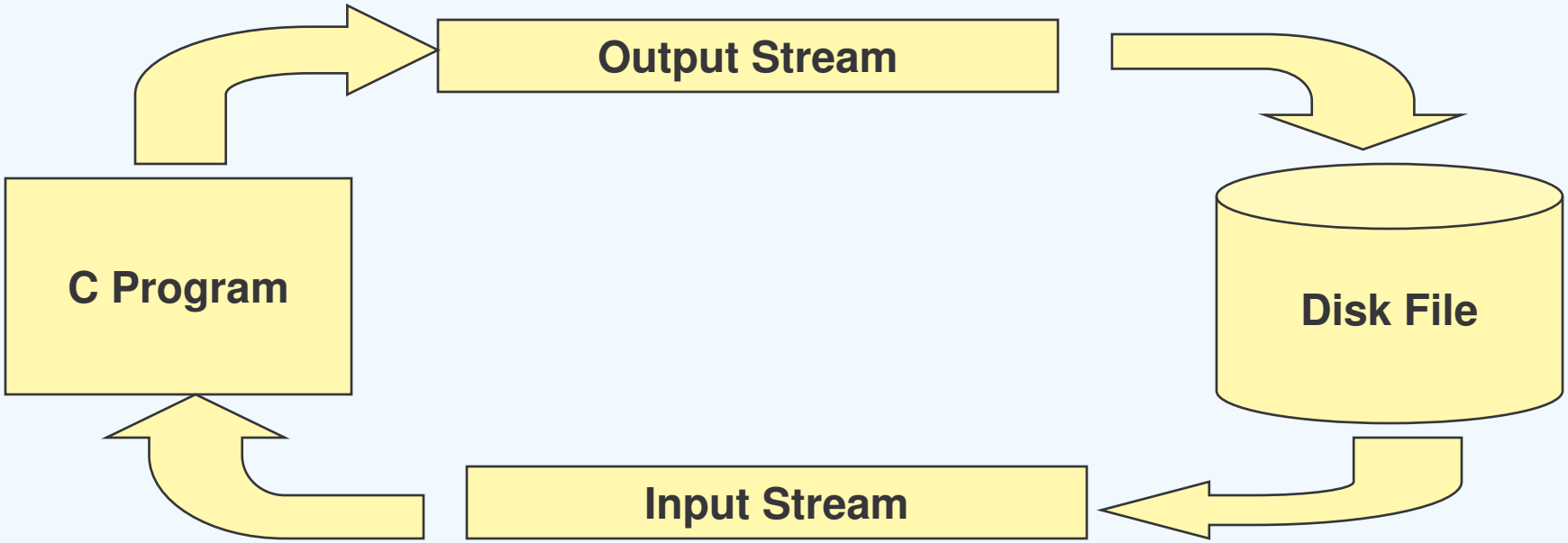


Understanding Streams

- The C Language I/O system provides a consistent interface to the programmer independent of the actual I/O device used.
- C provides a level of abstraction between the programmer and the I/O device. This abstraction is called as a stream.
- Stream is a logical connection to a file, screen , keyboard, and various other I/O devises.

Understanding Streams (Contd…)

Illustrating a stream



Understanding Streams (Contd…)

- There are 2 types of streams
  - Text stream (stream of ASCII characters)
  - Binary stream (stream of bytes)
- C has a inbuilt structure FILE which represents a the logical stream.
- This structure has a buffer
- Whenever a C Program interacts with a disk file it creates a variable of structure FILE.

File Opening Modes

Mode	Description
“r”	Searches file. If the file exists, loads it into memory and sets up a pointer which points to the first character in it. If the file doesn’t exists it returns a null. Operations possible - reading from the files
“w”	Searches file. If the file exists, contents are overwritten. If the file doesn’t exists a new file is created. Returns a null if unable to open the file. Operations possible - writing to the file.
“a”	Searches file. If the file exists, loads it into memory and sets up a pointer which points to the first character in it. If the file doesn’t exists a new file is created. Returns a null if unable to open the file. Operations possible - appending new contents at the end of the file.

File Opening Modes (Contd...)

Mode	Description
“r”	Same as “r” Operations possible – reading existing contents, writing new contents, modifying existing contents of the file.
“w”	Same as “w” Operations possible – writing new contents, reading them back and modifying existing contents of the file.
“a”	Same as “a” Operations possible – reading existing contents, appending new contents to the end of file, cannot modify existing contents of the file.

High Level I/O functions

- Functions to open a File.
  - fopen() is the function to open a file and its prototype is **FILE *fopen (char *name, char *mode);**
  - fopen() establishes a link between the program and the operating system.
  - The first argument specifies the file name and the second argument specifies the mode in which the file has to be opened.

High Level I/O functions (Contd…)

- Functions to open a File.
  - If the file is found at the specified location then fopen() creates a variable of type FILE structure and returns a pointer to it. This pointer is termed as a file pointer.
  - The FILE structure is nothing but a stream of characters/bytes which acts as a logical representative of the actual file.
  - If the file could not be opened then fopen() returns a NULL.

High Level I/O functions (Contd…)

- Functions to close a File.
  - fclose() is the counterpart of fopen() and its prototype is **int fclose(FILE *_stream_);**.
  - The argument to fclose() is the FILE pointer returned by fopen().
  - fclose() returns 0 if the file was closed successfully otherwise it returns -1.

High Level I/O functions (Contd...)

- Functions for reading characters.
  - fgetc()is a function to read characters from a file and its prototype is **int fgetc(FILE **stream*);**
  - The argument to fgetc() is the FILE pointer returned by fopen().
  - fgetc() returns the character read as an **int** or returns **EOF** to indicate an error or end of file.
  - EOF is a macro which is # defined to -1.

High Level I/O functions (Contd…)

- Functions for writing characters.
  - fputc() is a function to write characters to a file and its prototype is **int fputc(int** *c*, **FILE** **stream* **);**
  - The first argument to fputc() is the character to be written to the file.
  - The second argument to fputc() is the FILE pointer returned by fopen().
  - fputc() returns the character written to the file. It returns a **EOF** to indicate an error.

High Level I/O functions (Contd…)

- Functions for reading lines
  - fgets()is a function to read lines from a file and its prototype is **char *fgets(char **string*, int *n*, FILE **stream*);**
  - The first argument to fgets() is a character pointer which points to a buffer which will receive the line read from the file.
  - The second argument to fgets() is the size of the buffer.
  - The Third argument to fgets() is the FILE pointer returned by fopen().
  - fgets() returns NULL to indicate an error or end of file.

High Level I/O functions (Contd…)

- Functions for writing lines
  - fputs() is a function to write lines to a file and its prototype is **int fputs(const char **string*, FILE **stream*);**
  - The first argument to fputc() is a character pointer which points to a buffer which holds the string to be written to the file.
  - The second argument to fputc() is the FILE pointer returned by fopen().
  - fputs() returns a non negative number if successful else It returns a **EOF** to indicate an error.

High Level I/O functions (Contd…)

- Functions for reading blocks of data
  - fread()is a function to read blocks of data from a file and its prototype is **size_t fread(void **buffer*, size_t *size*, size_t *count*, FILE **stream*);**
  - A block will have records of the same type. Example of record could be employee record which stores employee details like employee #, Name date of joining etc.
  - **fread** returns the number of full items actually read, which may be less than *count* if an error occurs or if the end of the file is encountered before reaching *count.*

High Level I/O functions (Contd…)

- Functions for reading blocks of data
  - The first argument to fread() is a void pointer, which points to a buffer that receives the blocks read from the file.
  - The second argument to fread() is the size of each record.
  - The third argument to fread() is the number of records to be read.
  - The fourth argument to fread() is the FILE pointer returned by fopen().

High Level I/O functions (Contd…)

- Functions for writing blocks of data
  - fwrite()is a function to write blocks of data to a file and its prototype is **size_t fwrite(const void *_buffer_, size_t _size_, size_t _count_, FILE *_stream_);**
  - A block will have records of the same type. Example of record could be employee record which stores employee details like employee #, Name date of joining etc.
  - fwrite() returns the number of full items actually written, which may be less than *count* if an error occurs.

High Level I/O functions (Contd…)

- Functions for writing blocks of data
  - The first argument to fwrite() is a void pointer to a constant, which points to a buffer that has the blocks of data, to be written to the file.
  - The second argument to fwrite() is the size of each record.
  - The third argument to fwrite() is the number of records to be written.
  - The fourth argument to fwrite() is the FILE pointer returned by fopen().

High Level I/O functions (Contd…)

- High Level functions can be categorized as text mode and binary mode functions and so far we have seen the text mode functions.

- There are 3 main areas where text and binary mode files are different
  - Handling newlines.
  - Representation of End of File.
  - Storage of numbers.

High Level I/O functions (Contd...)

- **Behavior of newlines if a file is opened in Binary mode.**
  - In a text mode operation every newline character is converted to a carriage return – linefeed combination when it is written to the disk. Likewise the carriage return – linefeed combination is converted to a newline character when it is read from the disk.
  - In a binary mode operation the above mentioned conversions don't take place.

High Level I/O functions (Contd…)

- End of File detection for a file opened in Binary mode
  - In a text mode operation, a special character, whose ASCII value 26, is inserted after the last character in the file to mark the end of file.
  - In a binary mode operation there is no special character to indicate end of file. The binary mode files keep track of the end of file from the number of characters present in the directory entry of the file.
  - A file opened for writing in binary mode should be opened for reading in binary mode only.

High Level I/O functions (Contd...)

- Storage of numbers for a file opened in binary mode
  - In text mode operation, integers & floats are stored as strings of characters i.e. 12345 will occupy 5 bytes because individual digits are stored as characters occupying one byte each.
  - In a binary mode operation, integers and floats are stored exactly in the same way as they are stored in memory i.e. integers will occupy 2 bytes and floats will occupy 4 bytes.

High Level I/O functions (Contd…)

- Functions for random access of file contents
  - fseek() is a function to move the file pointer to a specified location and its prototype is **int fseek(FILE** ***stream*****, long** *offset*, **int** *origin* **);**
  - If successful, fseek() returns zero. Otherwise, it returns a nonzero value.
  - The value of *offset* can be a negative value also.

High Level I/O functions (Contd...)

- Functions for random access of file contents
  - The first argument to fseek() is the FILE pointer returned by fopen().
  - The second argument specifies the number of offset bytes from the initial position.
  - The third argument specifies the initial position which could be one of the following
    - **SEEK_CUR** Current position of file pointer
    - **SEEK_END** End of file
    - **SEEK_SET** Beginning of file

High Level I/O functions (Contd…)

- Function for returning the current position of the file pointer.
  - ftell() is a function to get the current position of a file pointer and its prototype is **long ftell(FILE *_stream_);**
  - The argument to ftell() is the FILE pointer returned by fopen().
  - On error, ftell() returns –1L.

High Level I/O functions (Contd…)

- Function for detecting errors in read/write operations.
  - ferror() is a function which reports any errors that might have occurred during read/write operation on a file.
  - It returns a zero if the read/write is successful and a nonzero value in case of a failure.

High Level I/O functions (Contd…)

- Function for repositioning the file pointer to the beginning of a file.
  - rewind() repositions the file pointer to the beginning of a file and its prototype is void rewind(FILE *_stream_);
  - The argument to rewind() is the FILE pointer returned by fopen().

Command Line Arguments

- Just like any other function, we can pass arguments to main( ) .The function main( ) can have two arguments traditionally named as argc & argv.
- Syntax:
  - main (int argc, char *argv[]){ }
- argv is an array of pointers to strings. argc is an integer whose value is equal to the no of strings to which argv points. argv[argc] is always NULL.

Summary

- Topics covered in this unit
  - Concept of streams.
  - High level File I/O functions.
  - Command line arguments.

Thank You!