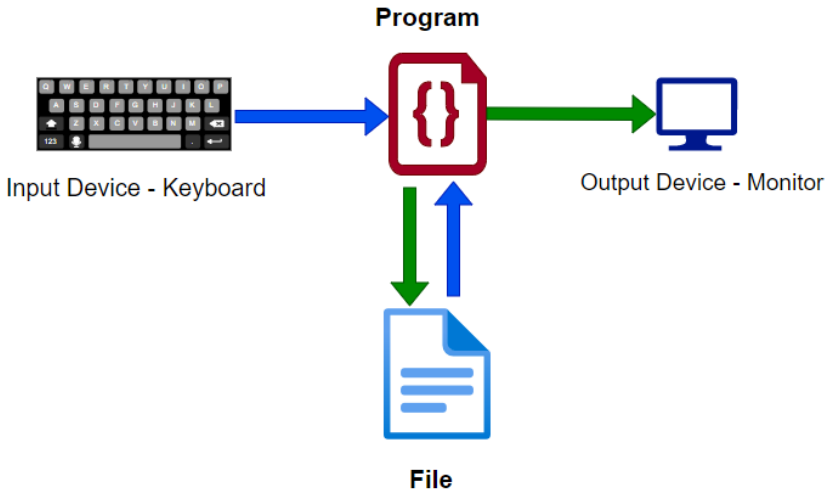
File Handling

<Training Topic /Lesson Name>

Lesson Objectives

- *The terms "file" and "stream"*
- *Text streams and binary streams*
- *FILE pointer*



Section 1

Terminology and definitions

Terminology and definitions

- *File, Stream and data type: FILE*
- *Text Streams and Binary Streams*

File (Computer file)

- *File system, a method of storing and organizing computer files and their data*
- *Computer file, a resource for storing information.*

Streams

- *The interfacing with different sources of sequential data, like files, sockets, keyboards, USB ports, printers or whatever.*
- *Interface uses properties that are common to all of them.*
- *For historical reasons, the type of the C data structure that represents a stream is called FILE rather than “stream”.*

Data Type: FILE

- *This is the data type used to represent stream objects.*
- *A FILE object holds all of the internal state information about the connection to the associated file*
- *FILE objects are allocated and managed internally by the input/output library functions.*

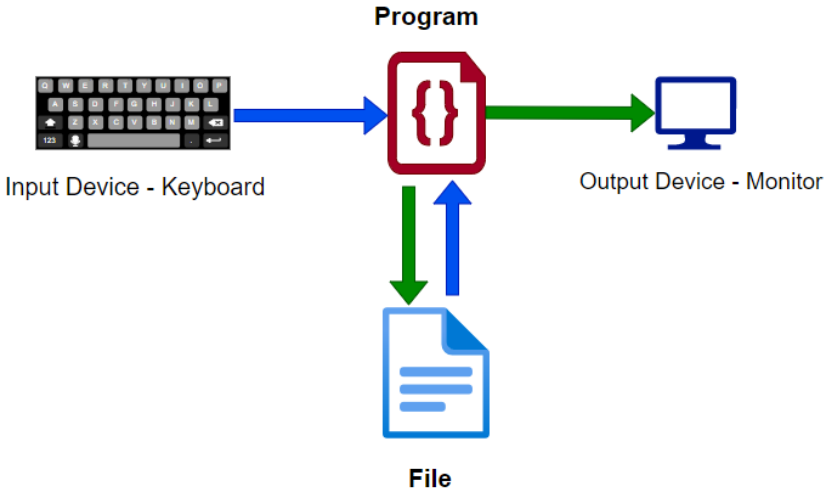
Text stream

- *The data read from a text stream is divided into lines which are terminated by newline ('\n') characters*
- *On some systems, text files can contain only printing characters, horizontal tab characters, and newlines, and so text streams may not support other characters.*
- *In a text stream, certain character translations may occur as required by the environment*

Binary streams

- *A binary stream is simply a long series of characters.*
- *Binary streams can handle any character value.*
- *Binary stream is always more capable and more predictable than a text stream*

Terminology and definitions Summary



Section 2

Working with Streams

Working with Streams

- *Opening/Closing Streams*
- *Stream Buffering*
- *Stream Input/Output*
- *File Positioning*
- *End-Of-File and Errors*
- *Streams and Threads*

Standard Streams

- *When the main function of your program is invoked, it already has five predefined streams open and available for use.*
  - *The standard input (stdin)*
  - *The standard output (stdout)*
  - *The standard error (stderr)*
  - *The standard printer (stdprn)*
  - *The standard auxiliary (stdaux)*

Opening Streams

- *Opening a file with the fopen function creates a new stream and establishes a connection between the stream and a file. This may involve creating a new file.*
- *The fopen function opens a stream for I / O to the file filename, and returns a pointer to the stream.*
- *If the open fails, fopen returns a null pointer.*
- *You can have multiple streams (or file descriptors) pointing to the same file open at the same time.*

*FILE * **fopen** (const char *filename, const char *opentype)*

Opening Streams

OpenType	
'r'	Open an existing file for reading only.
'w'	Open the file for writing only. If the file already exists, it is truncated to zero length. Otherwise a new file is created.
'a'	Open a file for append access; that is, writing at the end of file only. If the file already exists, its initial contents are unchanged and output to the stream is appended to the end of the file. Otherwise, a new, empty file is created.
'r+'	Open an existing file for both reading and writing. The initial contents of the file are unchanged and the initial file position is at the beginning of the file.
'w+'	Open a file for both reading and writing. If the file already exists, it is truncated to zero length. Otherwise, a new file is created.
'a+'	Open or create file for both reading and appending. If the file exists, its initial contents are unchanged. Otherwise, a new file is 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.

Closing Streams

- *When a stream is closed with fclose, the connection between the stream and the file is canceled. After you have closed a stream, you cannot perform any additional operations on it.*
- *The fclose function returns a value of 0 if the file was closed successfully, and EOF if an error was detected.*

int **fclose** *(FILE *stream)*

int **fcloseall** *(void)*

Closing Streams

- *If the main function to your program returns, or if you call the exit function, all open streams are automatically closed properly.*
- *If your program terminates in any other manner, such as by calling the abort function or from a fatal signal, open streams might not be closed properly. Buffered output might not be flushed and files may be incomplete.*

Stream Buffering

- *Characters that are written to a stream are normally accumulated and transmitted asynchronously to the file in a block, instead of appearing as soon as they are output by the application program. This is called buffering.*
- There are three different kinds of buffering: *unbuffered, line buffered, fully buffered*

Flushing Buffers

- *Flushing output on a buffered stream means transmitting all accumulated characters to the file.*
- *There are many circumstances when buffered output on a stream is flushed automatically: the output buffer is full, the stream is closed, the program terminates by calling exit, …*
- *If you want to flush the buffered output at another time, call* ***fflush***

  *int fflush (FILE *stream)*

Formatted Input/Output

- *The fprintf() and fscanf() though the easiest, are not always the most efficient*
- *Extra overhead is incurred with each call, since the data is written in formatted ASCII data instead of binary format*
- *So, if speed or file size is a concern, fread() and fwrite() are a better choice.*

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

Character Input/Output

Prototype	Function
int **fgetc** *(FILE *stream)*	Reads the next character.
int **getc** *(FILE *stream)*	Just like **fgetc**, except that it is implemented as a macro
int **getchar** *(void)*	Equivalent to **getc** with stdin as the value of the *stream* argument.
int **fputc** *(int c, FILE *stream)*	Converts the character *c* to type unsigned char, and writes it to the *stream*.
int **putc** *(int c, FILE *stream)*	Just like **fputc**, except that it can be implement as a macro, making it faster.
int **putchar** *(int c)*	Equivalent to **putc** with *stdout* as the value of the *stream* argument.
int **fputs** *(const char *s, FILE *stream)*	Writes the string *s* to the *stream*

Block Input/Output

- *You can use these functions to read and write binary data, as well as to read and write text in fixed-size blocks instead of by characters or lines.*

  *size_t fread (void *data, size_t size, size_t count, FILE *stream)*

  *size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)*

File Positioning

- *The file position of a stream describes where in the file the stream is currently reading or writing. I/O on the stream advances the file position through the file.*
- *The file position is represented as an integer, which counts the number of bytes from the beginning of the file.*

  *long int ftell (FILE *stream)*

  *int fseek (FILE *stream, long int offset, int whence)*

  *void rewind (FILE *stream)*

End-Of-File and Errors

- *Many of the functions return the value of the macro EOF to indicate unsuccessful completion of the operation. Since EOF is used to report both end of file and random errors, it's often better to use the feof function to check explicitly for end of file and ferror to check for errors.*

  EOF (GNU C Library, EOF is -1)

  *int feof (FILE *stream)*

  *int ferror (FILE *stream)*

Streams and Threads

- *Streams can be used in multi-threaded applications in the same way they are used in single-threaded applications. But the programmer must be aware of the possible complications.*

  *void flockfile (FILE *stream)*

  int ftrylockfile (FILE * stream)

  void funlockfile (FILE * stream)

Lesson Summary

Thank you