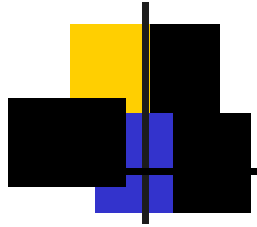




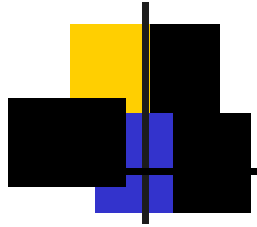
File Handling

Session 12



Objectives

- Explain streams and files
- Discuss text streams and binary streams
- Explain the various file functions
- Explain file pointer
- Discuss current active pointer
- Explain command-line arguments



File Input/Output

- All I/O operations in C are carried out using functions from the standard library
- This approach makes the C file system very powerful and flexible
- I/O in C is unique because data may be transferred in its internal binary representation or in a human-readable text format



Streams

- The C file system works with a wide variety of devices including printers, disk drives, tape drives and terminals
- Though all these devices are very different from each other, the buffered file system transforms each device into a logical device called a stream
- Since all streams act similarly, it is easy to handle the different devices
- There are two types of streams - the text and binary streams



Text Streams

- A text stream is a sequence of characters that can be organized into lines terminated by a new line character
- In a text stream, certain character translations may occur as required by the environment
- Therefore, there may not be a one-to-one relationship between the characters that are written (or read) and those in the external device
- Also, because of possible translations, the number of characters written (or read) may not be the same as those in the external device



Binary Streams

- A binary stream is a sequence of bytes with a one-to-one correspondence to those in the external device, that is, there are no character translations
- The number of bytes written (or read) is the same as the number on the external device
- Binary streams are a flat sequence of bytes, which do not have any flags to indicate the end of file or end of record
- The end of file is determined by the size of the file



Files

- A file can refer to anything from a disk file to a terminal or a printer
- A file is associated with a stream by performing an open operation and disassociated by a close operation
- When a program terminates normally, all files are automatically closed
- When a program crashes, the files remain open



Basic File Functions

Name	Function
fopen()	Opens a file
fclose()	Closes a file
fputc()	Writes a character to a file
fgetc()	Reads a character from a file
fread()	Reads from a file to a buffer
fwrite()	Writes from a buffer to a file
fseek()	Seeks a specific location in the file
fprintf()	Operates like printf(), but on a file
fscanf()	Operates like scanf(), but on a file
feof()	Returns true if end-of-file is reached
ferror()	Returns true if an error has occurred
rewind()	Resets the file position locator to the beginning of the file
remove()	Erases a file
fflush()	Writes data from internal buffers to a specified file



File Pointer

- A file pointer is essential for reading or writing files
- It is a pointer to a structure that contains the file name, current position of the file, whether the file is being read or written, and whether any errors or the end of the file have occurred
- The definitions obtained from `stdio.h` include a structure declaration called `FILE`
- The only declaration needed for a file pointer is:

`FILE *fp`

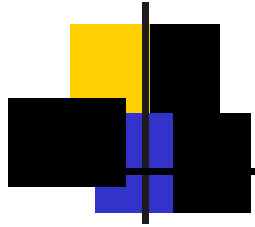


Opening a Text File

- The `fopen()` function opens a stream for use and links a file with that stream
- The `fopen()` function returns a file pointer associated with the file
- The prototype for the `fopen()` function is:

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

Mode	Meaning
r	Open a text file for reading
w	Create a text file for writing
a	Append to a text file
r+	Open a text file for read/write
w+	Create a text file for read/write
a+f	Append or create a text file for read/write



Closing a Text File

- It is important to close a file once it has been used
- This frees system resources and reduces the risk of overshooting the limit of files that can be open
- Closing a stream flushes out any associated buffer, an important operation that prevents loss of data when writing to a disk
- The `fclose()` function closes a stream that was opened by a call to `fopen()`
- The prototype for `fclose()` is:

`int fclose(FILE *fp);`

- The `fcloseall()` function closes all open streams



Writing a Character – Text File

- Streams can be written to either character by character or as strings
- The `fputc()` function is used for writing characters to a file previously opened by `fopen()`
- The prototype is:

`int fputc(int ch, FILE *fp);`



Reading a Character – Text File

- The `fgetc()` function is used for reading characters from a file opened in read mode, using `fopen()`
- The prototype is:
`int fgetc(int ch, FILE *fp);`
- The `fgetc()` function returns the next character from the current position in the input stream, and increments the file position indicator



String I/O

- The functions `fputs()` and `fgets()` write and read character strings to and from a disk file
- The `fputs()` function writes the entire string to the specified stream
- The `fgets()` function reads a string from the specified stream until either a new line character is read or length-1 characters have been read
- The prototypes are:

`int fputs(const char *str, FILE *fp);`

`char *fgets(char *str, int length, FILE *fp);`

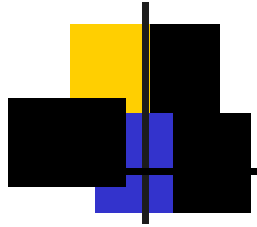


Opening a File-Binary

- The fopen() function opens a stream for use and links a file with that stream
- The fopen() function returns a file pointer associated with the file
- The prototype for the fopen() function is:

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

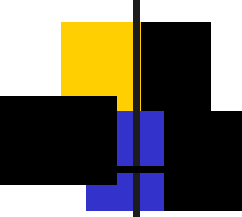
Mode	Meaning
rb	Open a binary file for reading
wb	Create a binary file for writing
ab	Append to a binary file
r+b	Open a binary file for read/write
w+b	Create a binary file for read/write
a+b	Append a binary file for read/write



Closing a File Binary

- The `fclose()` function closes a stream that was opened by a call to `fopen()`
- The prototype for `fclose()` is:

```
int fclose(FILE *fp);
```

The fread() and fwrite() functions

- The functions fread() and fwrite() are referred to as unformatted read or write functions
- They are used to read and write an entire block of data to and from a file
- The most useful application involves reading and writing user-defined data types, especially structures
- The prototypes for the functions are:

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

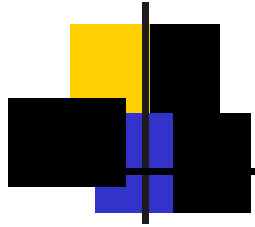
```
size_t fwrite(const void *buffer, size_t num_bytes, size_t  
count, FILE *fp);
```



Using feof()

- The function feof() returns true if the end of the file has been reached, otherwise it returns false (0)
- This function is used while reading binary data
- The prototype is:

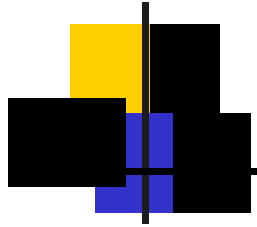
int feof (FILE *fp);



The rewind() function

- The rewind() function resets the file position indicator to the beginning of the file
- It takes the file pointer as its argument
- Syntax:

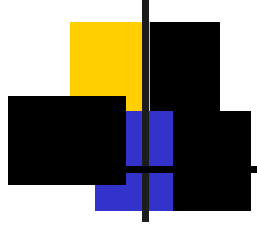
rewind(fp);



The `ferror()` function

- The `ferror()` function determines whether a file operation has produced an error
- As each operation sets the error condition, `ferror()` should be called immediately after each operation; otherwise, an error may be lost
- Its prototype is:

`int ferror(FILE *fp);`



Erasing Files

- The `remove()` function erases a specified file
- Its prototype is:

`int remove(char *filename);`

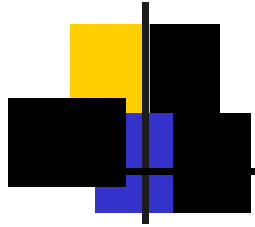


Flushing streams

- The `fflush()` function flushes out the buffer depending upon the file type
- A file opened for read will have its input buffer cleared, while a file opened for write will have its output buffer written to the files
- Its prototype is:

`int fflush(FILE *fp);`

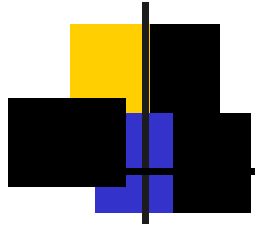
- The `fflush()` function, with a null, flushes all files opened for output



The Standard Streams

Whenever a C program starts execution under DOS, five special streams are opened automatically by the operating system

- **The standard input (stdin)**
- **The standard output (stdout)**
- **The standard error (stderr)**
- **The standard printer (stdprn)**
- **The standard auxiliary (stdaux)**



Current Active Pointer

- A pointer is maintained in the FILE structure to keep track of the position where I/O operations take place
- Whenever a character is read from or written to the stream, the current active pointer (known as curp) is advanced
- The current location of the current active pointer can be found with the help of the ftell() function
- The prototype is:

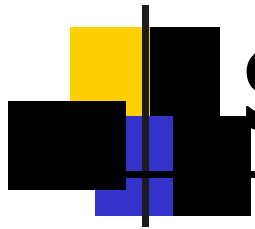
long int ftell(FILE *fp);



Setting Current Position-1

- The `fseek()` function repositions the curp by the specified number bytes from the start, the current position or the end of the stream depending upon the position specified in the `fseek()` function
- The prototype is:

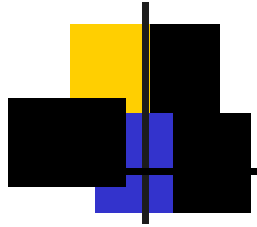
`int fseek (FILE *fp, long int offset, int origin);`



Setting Current Position-2

- The origin indicates the starting position of the search and has values as follows:

Origin	File Location
SEEK_SET or 0	Beginning of file
SEEK_CUR or 1	Current file pointer position
SEEK_END or 2	End of file

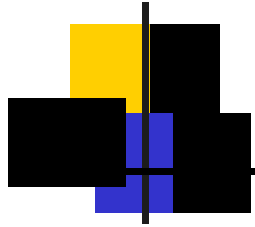


fprintf() and fscanf()-1

- The buffered I/O system includes fprintf() and fscanf() functions that are similar to printf() and scanf() except that they operate with files
- The prototypes of are:

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

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



fprintf() and fscanf()-2

- 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