Types of disk files

text streams are associated with **text-mode files**

line in a text mode file not same as string- no terminating null character

when use text-mode stream translation between C's newline and whatever OS uses mark end of lines

binary streams associated binary files. any and all data written and read unchanged, no separation or end of line characters

some file I/O functions restricted one file mode, others use either mode

Using filenames

must use file name when dealing with disk files

Opening a file

opening- stream linked to a disk file

when open becomes available for reading, writing, or both

when done using a file **YOU MUST CLOSE IT**

prototype for **fopen()**

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

returns a pointer to type file (structure in stdio.h) for that file

for each file want to open must declared a pointer to type file

creates instance of file structure and returns a pointer

if fopen() fails, returns NULL

argument **mode** specifies in which mode to open the file. pg 442 for modes

default mode is text

open in binary mode, append a b to mode argument

Writing and Reading File Data

3 ways write to disk file:

```
1. use **formatted output** to save formatted data to a file. only with text mode files. pri
2. use **character output** to save single characters or lines of characters to files. shoud
3. use **direct output** to save contents of section of memory directly to disk file. binary
```

generally read data same mode saved in

Formatted File Input and Output

text and numeric data foramtted specific way

Formatted File output

fprintf()

```
int fprintf(FILE *fp, char *fmt, ...);
```

1. pointer to type file
2. format string. same rules as printf()
3. names of variables to be outputted to specific stream (like printf)

formatted file input

fscanf()

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

like scanf

Character input and output

refers to single characters as well as lines of characters

character input

getc() and fgetc() input single character from specified stream

```
int getc(FILE *fp)
```

fgets() read line of characters from file

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

str pointer to buffer which input stored. n maximum nubmer of characters to be input. fp is pointer to type FILE from fopen()

character output

fputs() write line of characters to stream

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

Direct file input and output

most often used save data to be read later by the same or different c program

only used binary files

blocks of data written from disk to memory

fwrite() writes block of data from memory to binary-mode file

```
int fwrite(void *buf, int size, int count, FILE *fp);
```

buf- pointer to region of memory holding data to be written to file. size- size in bytes of individual items. count- number of items to be written. fp- pointer to file returned by fopen()

returns number of items written on success. if value less than count there was an error

good way to usually write:

```
if ( (fwrite(buf, size, count, fp)) != count)
fprintf(stderr, "Error writing to file.");
```

fread() reads a block of data from a binary mode file into memory

```
int fread(void *buf, int size, int count, FILE *fp);
```

“”

always do something to ensure fopens() fwrite() and fread() function correctly:

```
if ( (fp = fopen("direct.txt", "wb")) == NULL) {
fprintf(stderr, "Error opening file.");
exit(1) }

if (fwrite(array1, sizeof(int), SIZE, fp) != SIZE) {
fprintf(stderr, "Error writing to file.");
exit(1); }

fclose(fp);

if (fread(array2, sizeof(int), SIZE, FP) != SIZE) {
fprintf(stderr, "Error reading file.");
exit(1); }
```

File Buffering: Closing and Flushing Files

must always close with fclose()

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

returns 0 on success, -1 on error.

when close file's buffer is flushed (written to the file)

can close all open streams except standard ones with fcloseall()

```
int fcloseall(void);
```

when program terminates either by reaching end of main or exiting prematurely, streams closed. can be a good idea to close explicitly

when create stream to disk file buffer automatically created and associated with the stream

as program writes data to the stream, data is saved in the buffer until the buffer is full and then the entire contents of the buffer are written as a block to the disk

can flush streams buffers without closing by using fflush() or flushall()

```
int fflush(FILE *fp);
int flushall(void);
```

Understanding Sequential vs Random Memory Access

every open file has position indicator associated with it. specifies where read and write operations take place in file

when new file opened, position indicator at beginning of file. position 0

when exisitng file opened, indicator at end of file in append mode or at beginning of file if opened in any other mode

reading and writing operations occur at location of position indicator and update position indicator as well

don't need to be concerned about position indicator for reading

by controlling position indicator can perform **random file access**- read or write data to any position in a file without reading or writing all of the proceeding data

rewind()- set position indicator at beginning of file

```
void rewind(FILE *fp);
```

ftell()- determine value of file's position indicator

```
long ftell9FILE *fp);
```

fseek()- set position indicator anywhere in the file

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

offset distance want pointer to go origin specifies moves relative starting point. SEEK_SET- move offset bytes from beginning of file. SEEK_CUR- move offset bytes from current position. SEEK_END- move offset bytes from end of file

returns 0 if indicator successfully moved or nonzero if error occurred.

make sure you are not reading beyond the EOF

Detecting end of file

symbolic constant in stdio.h is -1

for text-mode file:

```
while ( (c = fgetc( fp )) !=EOF )
```

can't use -1 with binary either:

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

feof() returns 0 if end of file has not been reached or nonzero if has been reached

if EOF reached, no further read operations permitted until a rewind(), fseek() or file closed and reopened

```
while (!feof(fp)) {
fgets(buf, BUFSIZE, fp);
printf("%s", buf); }
```

File Management Functions

deleting, renaming, copying

remove()- deleting file forever

```
int remove(const char *filename);
```

0 on success, -1 on failure

rename()

```
int rename( const char *oldname, const char *newname );
```

both names must refer to the same disk drive

no explicit copy function:

```
1. open source file for reading in binary mode
2. open destination file in binary mode
3. read a character from source file
4. if feof indicates at end of source file can close both files
5. if have not reached eof write character to destination file and loop back to step 3
```

Using temp files

tmpnam()- creates temp filename does not exist anywhere in file system

```
char *tmpnam(char *s);
```

s is pointer to buffer large enuf hold file name. can also pass null in which case stored in buffer internal to tmpnam and function returns poitner to buffer