File Input and Output

November 17, 2011

Outline

Opening Files

- A prerequisite to reading/writing to a file is Opening a File.
- fopen()

fopen()

- fopen() takes a filename, does some housekeeping and negotiation with the OS and returns a pointer to be used in subsequent reads/writes. This pointer is called File Pointer.
- The file pointer points to a structure that contains information about the file, location of buffer, the current position in the buffer, read/write mode, errors/EOF that may have occured.
- FILE *fp
- fp=fopen(char *name,char *mode)

fopen()

- `Return Values`
  - `NULL on Error`
  - `File Pointer Otherwise`
- `Allowed Modes`
  - 'r': Read. Gives an Error if File Absent/No Read Permission.
  - 'w': Write. Creates missing file. Overwrites an exisiting file. Error if can't write/create file.
  - 'a': Append.Creates missing file. Preserves contents of existing file. Error if can't write/create file.

Reading and Writing Characters

- `int getc(FILE *fp)`
- `int putc(int c, FILE *fp)`
- `getc()` and `putc()` return the character read from/written to the file pointed to by `fp` or `EOF` on error.

Formatted File IO

- `int fscanf(FILE *fp,char *format,...)`
- `int fprintf(FILE *fp,char *format,...)`
- Return Values are similar to those of `scanf()` and `printf()`

Reading and Writing Lines/ Line IO

- `char* fgets(char *line,int maxline,FILE* fp)`
  - `reads the next input line(including newline) from fp into array line`
  - `reads at most maxline -1 characters`
  - `line is terminated by \0`
  - `Returns line on success, NULL on Error`
- `int fputs(char *line,FILE* fp)`
  - `writes a string line to fp`
  - `line may not contain a newline`
  - `Returns non-zero on success, EOF on Error`

Closing Files

- `int fclose(FILE *fp)`
  - Dissociates fp from the filename. Frees pointer for another file
  - OSs impose limit on number of simultaneously open files.
  - also flushes the output buffer
  - After normal program termination, fclose() is called automatically for each open file.

Error Handling

- `int ferror(FILE *fp)`
  - Returns non-zero if error occured on the stream fp.
- `int feof(FILE *fp)`
  - Returns non-zero if EOF has occured on stream fp.
- `void clearerr(FILE *fp)`
  - Clears EOF/Error indicator.

fseek()

- `int fseek(FILE *fp,long offset,int origin)`
  - Sets the file position for fp. Next read/write will access data beginning at new position.
- Allowed values for origin
  - SEEK_SET (beginning)
  - SEEK_CUR (current position)
  - SEEK_END (end of file)
- Returns non-zero on Error

fseek()

- For a text stream, offset must be 0 or a value returned by ftell() (in which case, origin must be SEEK_SET.)
- long ftell(FILE* fp)
  - Returns current file position for fp. Returns -1 on error.

rewind()

- `void rewind(FILE *fp)`
  - `equivalent to: fseek(fp,0,SEEK_SET); clearerr(fp);`

fgetpos() and fsetpos()

- `int fgetpos(FILE *fp,fpos_t *ptr)`
  - Gets current file position in ptr.
- `int fsetpos(FILE *fp,fpos_t *ptr)`
  - Sets file position equal to the one in ptr.
- Both these functions return non-zero on Error

fread() and fwrite()

- Read and Write variables from/into files.
- size_t fread(void *ptr, size_t size_of_elements, size_t number_of_elements, FILE *a_file);
- size_t fwrite(const void *ptr, size_t size_of_elements, size_t number_of_elements, FILE *a_file);
- Return the number of items successfully read/written. On Error, the return value is short item count.
- fread() does not distinguish between end-of-file and error, and callers must use feof() and ferror().