File function summary

- Open/Close files
  - fopen() – open a stream for a file
  - fclose() – closes a stream
- One character at a time:
  - fgetc() – similar to getchar()
  - fputc() – similar to putchar()
- One line at a time:
  - fprintf()/fputs() – similar to printf()
  - fscanf()/fgets() – similar to scanf()
- File errors
  - perror() – reports an error in a system call

Text Streams

- Files accessed through the FILE mechanism provided by <stdio.h>
  - http://www.acm.uiuc.edu/webmonkeys/book/c_guide/2.12.html
- Text streams are composed of lines.
- Each line has zero or more characters and are terminated by a new-line character which is the last character in a line.
- Conversions may occur on text streams during input and output.
- Text streams consist of only printable characters, the tab character, and the new-line character.
- Spaces cannot appear before a newline character, although it is implementation-defined whether or not reading a text stream removes these spaces.

File constants <stdio.h>

- FILE – a variable type suitable for string information for a file stream
- fpos_t – a variable type suitable for staring any position in a file
- NULL – value of a null pointer constant
- EOF – negative integer which indicates end-of-file has been reached
- FOPEN_MAX – integer which represents the maximum number of files that the system can guarantee that can be opened simultaneously
- FILENAME_MAX – integer which represents the longest length of a char array
- stderr/stdin/stdout – pointers to FILE types which correspond to the standard streams

File usage

- When a program begins, there are already three available streams which are predefined and need not be opened explicitly and are of type “pointer to FILE”
  - standard input
  - standard output
  - standard error
- Files are associated with streams and must be opened to be used.
- The point of I/O within a file is determined by the file position.
- When a file is opened, the file position points to the beginning of the file (unless the file is opened for an append operation in which case the position points to the end of the file).
- The file position follows read and write operations to indicate where the next operation will occur.
- When a file is closed, no more actions can be taken on it until it is opened again.
- Exiting from the main function causes all open files to be closed.

Open/Read a File – one char at a time

```
#include<stdio.h>
#include<stdlib.h>
int main() {
    char ch;
    FILE *fp;
    fp = fopen("lab2p2in","r"); // read mode
    if( fp == NULL ) {
        perror("Error while opening the file.\n");
        exit(EXIT_FAILURE); }
    printf("The contents of the file is :- \n\n");
    while( ( ch = fgetc(fp) ) != EOF )
        printf("%c",ch);
     fclose(fp);
return 0; }
```

C programming code to open a file and print its contents to the screen, one character at a time.

//fileio1.c

(1) fgetc returns the value of an int that is converted from the character

(2) What happens if delete lab2p2in file? i.e. it can't be found to open?

File open and close

- **FILE *fopen(const char *_filename_, const char *_mode_);**
- **Mode… (lots more!)**
  - r – read text mode
  - w – write text mode (truncates file to zero length or creates a new file)
  - If the file does not exist and it is opened with read mode (**r**), then the open fails → need to check for this
- Declaration: **int fclose(FILE *_stream_);**
  - Closes the stream.
  - If successful, it returns zero.
  - On error it returns **EOF**.
- perror
  - **void perror(const char *_str_);**
  - Prints a descriptive error message to stderr. First the string *str* is printed followed by a colon then a space (your error message). Then an error message based on the current setting of the variable **errno** is printed (system error message).

fgetc and fputc

- Declaration: **int fgetc(FILE *_stream_);**
  - Gets the next character (an **unsigned char**) from the specified stream and advances the position indicator for the stream.
  - On success the character is returned.
  - If the end-of-file is encountered, then **EOF** is returned and the end-of-file indicator is set.
  - If an error occurs then the error indicator for the stream is set and **EOF** is returned.
- Declaration: **int fputc(int** *char*, **FILE ***_stream_**);**
  - Writes a character (an **unsigned char**) specified by the argument *char* to the specified stream and advances the position indicator for the stream.
  - On success the character is returned.
  - If an error occurs, the error indicator for the stream is set and **EOF** is returned.

Open/Read/Write/Close… one char at a time

```
#include<stdio.h>
#include<stdlib.h>
int main() {
    char ch, chout;
    FILE *fpin, *fpout;
    fpin = fopen("lab2p2in","r");   // read mode
    fpout = fopen("lab2p2inout","w"); // write mode
    if( fpin == NULL ) {
        perror("Error while opening the input file.\n");
        exit(EXIT_FAILURE); }
    if (fpout == NULL ) {
        perror("Error while opening the output file.\n");
        exit(EXIT_FAILURE); }
    while( ( ch = fgetc(fpin) ) != EOF && chout != EOF )
        chout = fputc(ch,fpout);   // ret char if success ow EOF
     fclose(fpin);
     fclose(fpout);
return 0; }
```

C programming code to open a file and print its contents to the another file, one character at a time.

//fileio2.c

Lab2p2 excerpt example

```
FILE *infp, *outfp;
char * mode = "r";
char outfile[] = "lab2p2out";

char input[101], save_first_letter;
char  *inptr;
int first_letter = TRUE, n=101;

 infp = fopen("lab2p2in","r");
 if (infp == NULL){
       fprintf(stderr, "can't open input file lab2p2in!\n");
       exit(EXIT_FAILURE); }

 outfp = fopen(outfile,"w");
  if (outfp == NULL) {
       fprintf(stderr, "Can't open output file %s!\n", outfile);
       exit(EXIT_FAILURE); }

 fgets(input,n,infp);
 while (!feof(infp))
    {  // etc
          fgets(input,n,infp);
    }
//close files
```

- **fgets(*buffer*,*size*,stdin);**
- *buffer* is the location of your string storage space or buffer.
- *size* is the number of characters to input. This sets a limit on input
- Note that *fgets()* also reads in the carriage return (enter key; newline character) at the end of the line. That character becomes part of the string you input.
- fscanf(infp,"%s",input);
- **while (!feof(infp))**

fgets vs fscanf

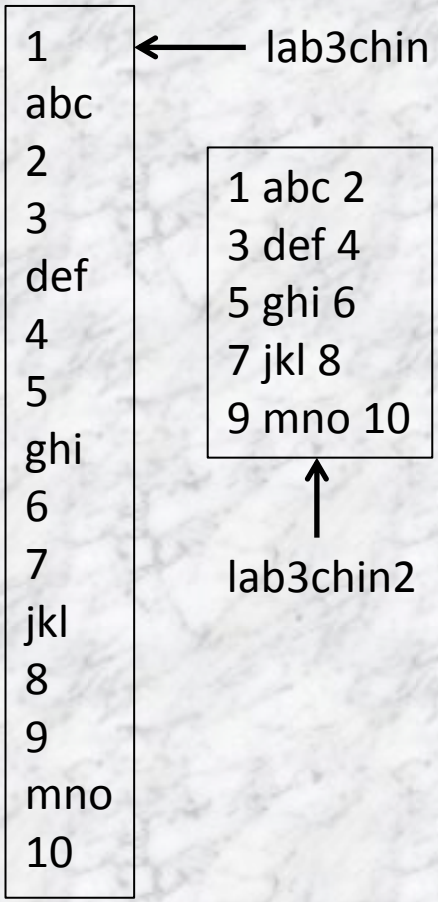
- Declaration: **char *fgets(char *_str_, int _n_, FILE *_stream_);**
  - Reads a line from the specified stream and stores it into the string pointed to by *str*.
  - It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.
  - It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.
  - **The newline character is copied to the string.**
  - A null character is appended to the end of the string.
  - On error a null pointer is returned. If the end-of-file occurs before any characters have been read, the string remains unchanged.
- Declaration: **int fscanf(FILE *_stream_, const char *_format_, ...);**
  - Reading an input field (designated with a conversion specifier) ends when an incompatible character is met, or the width field is satisfied.
  - On success the number of input fields converted and stored are returned. If an input failure occurred, then EOF is returned.
  - Returns EOF in case of errors or if it reaches eof

fprintf and feof

- Declaration:
  - int fprintf(FILE *_stream_, const char *_format_, ...);
  - sends formatted output to a stream
  - Just like printf, but puts file pointer as first argument
  - In lab2p2:
    - **fprintf(outfp, "Your sipher coded message is %s\n",input);**

- Declaration: int feof(FILE *_stream_);
  - Tests the end-of-file indicator for the given stream
  - If the stream is at the end-of-file, then it returns a nonzero value. If it is not at the end of the file, then it returns zero.

Lab3 fileio example



See handout