Text Files in C

A file is for storing permanent data. C provides file operations in `stdio.h`. A file is viewed as a stream of characters. Files must be opened before being accessed, and characters can be read one at a time, in order, from the file.

There is a current position in the file's character stream. The current position starts out at the first character in the file, and moves one character over as a result of a character read (or write) to the file; to read the 10th character you need to first read the first 9 characters (or you need to explicitly move the current position in the file to the 10th character).

There are special hidden chars (just like there are in the `stdin` input stream), `'\n'`, `'\t'`, etc. In a file there is another special hidden char, `EOF`, marking the end of the file.

Using text files in C

1. DECLARE a FILE * variable

```
FILE *infile;
FILE *outfile;
```

2. OPEN the file: associate the variable with an actual file using fopen you can open a file in read, "r", write, "w", or append, "a" mode

```
infile = fopen("input.txt", "r");       // using relative path name of file
if (infile == NULL) {
  // assume that Error is some error function that handles the error, maybe
        // printing out the passed error string and calling exit(1);
        Error("Unable to open file.");
}
outfile = fopen("/home/newhall/output.txt", "w");  // using absolute path name of file
if (outfile == NULL) {
        Error("Unable to open file.");
}
```

3. USE I/O operations to read, write, or move the current position in the file

```
int ch;   // EOF is not considered a char, but an int, and since all
          // char values can be stored in int, we define ch to be an int

ch = getc(infile);  // read in the character at the current position in the inputfile
putc(ch, outfile);  // write the value of ch to the current position in the outfile
```

4. CLOSE the file: use fclose to close the file after you are done with it

```
fclose(infile);
fclose(outfile);
```

You can also move the current file position in a file:

```
// to reset current position to beginning of file
void rewind(FILE *f);

rewind(infile);

// to move to a specific location in the file:
fseek(FILE *f, long offset, int whence);

fseek(f, 0, SEEK_SET);    // seek to the beginning of the file
fseek(f, 3, SEEK_CUR);    // seek to 3 chars from the current position
fseek(f, -3, SEEK_END);   // seek to 3 chars from the end of the file
```

File I/O operations in stdio.h

```
---------------
Character Based
---------------

int getc(FILE *f): returns the next character in the file stream f
                   as an integer or EOF
```

```
int fgetc(FILE *f): same as getc

int putc(int c, FILE *f): writes the character c to the file stream f and
                          returns the character written
int fputc(int c, FILE *f): same as putc

int ungetc(int c, FILE *f): pushes the character c back onto the file stream f
                            returns the chars pushed

        you can only push back one character and EOF cannot be pushed back

        used when you need to read in a char value and test it, and based on
        test results need to put it back in the file.

int getchar();    read from stdin

putchar(int c);   write to stdout (we usually use printf to do this instead)

-----------
Line  Based:  WARNING THESE CAN BE A BIT MORE TRICKY TO USE  than getc & putc
------------

char *fgets(char *s, int n, FILE *f);

    reads at most n-1 characters into the array s stopping if a newline is
    encountered, newline is included in the array which is '\0' terminated

int fputs(char *s, FILE *f);

    writes the string s which need not contain a newline onto the file stream f

---------
Formatted:  WARNING THESE CAN BE REALLY TRICKY TO USE
---------   (NEVER EVER USE scanf for reading in input from a user)

int fscanf(FILE *f, char *format, arg1, arg2, ...);

    designed to be the counterpart to printf, uses a similar control string
    returns EOF if end of file or an error occurs
    otherwise returns the number of input items converted and assigned

    int scanf(char *format, arg1, arg2, ...);   for stdin

%d integer
%f float
%lf double
%c character
%s string, up to first white space

%[...] string, up to first character not in brackets
%[0123456789] would read in digits
%[^...] string, up to first character in brackets
%[^\n] would read everything up to a newline


int fprintf(FILE *f, char *format, addrofarg1, addrofarg2, ...);

    just like standard printf, which assumes a file stream of stdout
    allows you to specify other file streams

Examples:
----------
int x;
double d;
char c, array[MAX];

scanf("%d %c", &x, &c);   // read int & char from stdin, ignore ALL whitespace

fscanf(infile, "%d,%c", &x, &c); // read an int & char from file where int and
                                 // char are separated by a comma

fprintf(outfile, "%d:%c\n", x, c); // write int & char char values to file
                                   // separated by colon, followd by new line char

fscanf(infile,"%s", array);  // read a string from file into array
```

```
                              // stops at white space

fscanf(infile, "%lf %24s", &d, array);  // read a double and a string upto
                                        // 24 chars from infile

fscanf(infile, "%20[012345]",array);  // read a string of at most 20 chars
                                      // consisting of only chars in set

fscanf(infile, "%[^.,:!;]", array); // read in a string, stop when hit
                                    // punctuation mark

fscanf(infile, "%ld %d%c", &x, &b, &c);  // read in two integer values store
                                         // first in long, second in int
                                         // read in end of line char into c

fscanf returns the number of items read (above example would return 3) or
it returns EOF if it reaches EOF while reading
```