C File I/O Implementation

You MUST compile all code with the -Wall compilation flag!

Include Files

You will need the following include files. If your system does not have it, then don't include it (your milage may vary)

stdio.h, stdlib.h, unistd.h, errno.h, fcntl.h, io.h, sys\stat.h, sys\types.h

Flags

You will need to define O_BINARY flag if it is not already defined. Some compilers default to opening files as binary, others as text. Those that default to binary do not have the O_BINARY flag while those that default to text do. It is not defined in ATCS.H because I want you to know what you compiler is doing. We are going to use it all the time regardless.

```
#ifndef O_BINARY
#define O_BINARY 0
#endif
```

I/O Functions

There are five (5) non-buffered low level file I/o functions that we are going to use:

open, close, read, write, lseek

int open(const char *file, int o_flag [, int p_mode])

Opens a file with the selected name. Existing files can be opened or new files can be created with this function.

returns -1 on error and sets errno accordingly.

o_flag = O_APPEND | O_CREAT | O_EXCL | O_RDONLY | O_RDWR | O_TRUNC | O_WRONLY
only use the flags that are appropriate for the desired action.

p_mode = S_IWRITE | S_IREAD
These optional flags must be set when a file is being created or you will have neither write nor read access afterwards (you still have delete permission however).

O_APPEND	Position file pointer to end
O_BINARY	Open file in binary mode (do not use with O_TEXT. O_BINARY is the default on UNIX systems.)
O_CREAT	Create file if it does not exist (see O_EXCL)
O_EXCL	Used with O_CREAT, returns an error if file exists
O_RDONLY	Open file for reading only
O_RDWR	Open file for reading and writing

O_TEXT	Open file in text mode (do not use with O_BINARY. O_TEXT this is the default on Windows systems.)
O_TRUNC	Truncates existing file
O_WRONLY	Opens file for writing only

Sample Usages:
Open an existing file for reading in binary mode

```
int hFile;
char* pFilename = "myinputfile.dat";

hFile = open(pFilename, O_BINARY | O_RDONLY);
if (hFile != -1)
   {
   close(hFile);
   hFile = 0;
   }
else
   printf("Error %d Opening File %s for Read\n", errno, pFilename);
```

Create a file for reading and writing in binary mode. The file cannot already exist.

```
int hFile;
char* pFilename = "mynewfile.dat";

hFile = open(pFilename, O_BINARY | O_RDWR | O_CREAT| O_EXCL, S_IWRITE | S_IREAD);
if (hFile != -1)
   {
   close(hFile);
   hFile = 0;
   }
else
   printf("Error %d Creating File %s\n", errno, pFilename);
```

int close(int hFile)

0 if successful -1 otherwise.

int read(int hFile, void *buffer, unsigned int len)
int write(int hFile, void *buffer, unsigned int len)

Read or write len bytes to or from the memory location pointed to by buffer.

returns the number of bytes read/written, which can be different from len of the enf of file is reached, or -1 if there is a catastrophic failure.

hFile is the file handle returned by open.
buffer is a pointer to the memory where data are to be read/written.
len is the number of bytes to be read/written.

off_t lseek(int hFile, off_t offset, int whence)

returns the resulting offset, as measured in bytes from the beginning of the file or (off_t)-1 will be returned if there is an error and errno shall be set to indicate the error, and the file offset shall remain unchanged.

whence is one of the three condition flags: SEEK_SET, SEEK_CUR or SEEK_END

SEEK_SET Start at the beginning of the file and move offset bytes.
SEEK_CUR Start at the current location and move offset bytes.
SEEK_END Start at the end of the file and move offset bytes.

Here is how to get the length of a file without altering the file pointer

```
…
CurrentPos = lseek(hFile, (size_t)0, SEEK_CUR);
length = lseek(hFile, (size_t)0, SEEK_END);
lseek(hFile, CurrentPos, SEEK_SET);
…
```