C Project Starters...

We are going to slowly build up what we need to decompose WAV files and manipulate the bits.

You will need to download the file acts.h from Athena2 and add it to your project.

We need to define a structure that has two things in it (there will be more later). The fist item is the size of the file being loaded and the second is a place holder for the first piece of data that is going to be loaded in.

```
struct mem {size_t size;  // size of the file in bytes
            BYTE data[1]; // place holder for the bytes from the
file
           };
```

In main you will need functions to perform these actions in this order:

- Function 1: Open and Read a file into memory
- Function 2: Data Processing
- Function 3: Saving
- Function 4: Housekeeping

Function 1: returns a `struct mem*`. pass the filename to it and open the file for reading, call a function to get the file size (you need to write it using the notes on file i/O on Athena2), allocate memory to hold the file and read the entire file into memory starting at the location `data[0]`, close the file and return the structure pointer. You will need to test for success every step of the way and handle problems accordingly. C does not throw exceptions except in the case of division by zero. If the function fails it should return a null pointer and not leave any allocated memory behind. The following code stub illustrates how you allocate memory, set the pointer to it and read in the data from the file (assumed to be opened and found to have filesize bytes in it). The details on file i/o are in a document on Athena2.

```
struct mem* pmem;
size_t filesize;
...
hFile = open(...) // Open a file for binary read
if (hFile != -1) // Only use the file handle if the open was
successful
   {
   ...
   pmem = (struct mem*)malloc(filesize + sizeof(struct mem));

   if (pmem != (struct mem*)NIL)
      {
      nbytes = read(hFile, &pmem->data[0], filesize); // load
entire file into memory
```

```
      if (nbytes != filesize) // Error reading the file (don't
care what it was)
         {
         free(pmem);              // give back the memory to the
operating system
         pmem = (struct mem*)NIL; // null out the return value
to signal the failure
         ...
         }
      } // if (pmem != NIL)

   close(hFile);
   } // if (hFIle != -1)
...
```

Function 2: This is the processing function. You will pass it the pointer returned by Function 1 (assuming it did not fail of course). It is just a place holder for now. It will be filled in later. Return an `int`.

Function 3: Pass it a filename to create and the pointer to the mem structure. This function saves the data in memory back to a new file on the disk. See them file i/o document for details on creating new files.

Function 4: Housekeeping. Pass it the pointer returned by Function 1. We are all done so free up the memory allocated in Function 1 (just use the free() function shown above) and print any helpful messages.