Demonstration in Preparation for Home Work 4

Here we demonstrate

- Command-line arguments
- Determining the size of a file
- Allocating memory on the heap to a pointer to hold the entire contents of a file and later, freeing the memory allocated to the above mentioned pointer
- Read the entire file into this area allocated on the heap
- Use pointer arithmetic to access various pieces of information contain in this data read from the file to determine specific details about the data and manipulate the data accordingly using only pointers, indirection (dereference), a quick sample of double indirection to change the address to which a pointer points, and pointer arithmetic

Sounds like quite a plateful but, if we stay focused during the classroom demonstration, these topics will become most clear.

The binary files we will use and access each have similar characteristics and differences based on these characteristics:

1st 2 bytes label the file type. These are ASCII characters which mean:

CH: File that contains a secret message! The length is indicated in the next 4 bytes as an integer, message follows.

IN: File that contains what can be stored as a 2D array of integers. The row and column sizes are indicated in the next 8 bytes as 2 4 byte integers, the 2D data follows.

FL: File that contains what can be stored as a 2D array of floats. The row and column sizes are indicated in the next 8 bytes as 2 4 byte integers, the 2D data follows.

Next 4 or 8 bytes indicates how much data that we need to manipulate exists

Past the above mentioned 4 or 8 bytes is the data we need to manipulate

We need to first, determine the size of the file in order to allocate that exact amount of memory. Previously discussed and demonstrated, the get_file_size function has already been included in the code.

Open the file and pass the FILE object and the file size to a function, say processFileData, that will process the file according to the file type.

Inside processFileData:

Allocate filesize bytes to a void pointer and read the entire file into this allocated memory.

Determine the file type and pass the pointer + 2 (next byte past the 2 ASCIIs) address to the appropriate process. processString, processIntegers, processFloats (this one is almost the same as processInteger and may not have the time to complete though, we can do it quickly with copy/paste and minor alterations).

Free the void pointer.

processString: Use the length information to allocate enough memory for the coded string + 1 byte and using pointer arithmetic and dereferencing, decode the secret message into this allocated memory and print it out. How to decode is purely academic and will be presented in lecture. Once complete, we free the pointer.

processIntegers: Use pointers to get the rows and columns information. Use pointer arithmetic in order to print out and sum the contents of this 2D array of integers. Here we will also demonstrate double indirection where passing a pointer (in essence, an address contained in the pointer) to a function it will returned pointing to a new address, i.e. the function updates not the data to which the pointer points but, the address to which it points.

processFloats: If we get the time, this will be nearly identical to processIntegers.

This document, 3 binary input files, and a skeleton version of our solution is provided on Canvas