Due: 11:59 PM, April 14, 2020

1. Reading Binary Data

Binary files typically contain information that describes their structure. You will write a C program that reads such a binary file and displays its contents. The file is divided into three sections:

a) Section 1 – File Header
The header appears at the beginning of the file. The first 4 bytes of the header contain a file signature "TATA". Immediately following the signature is an integer (4 bytes) giving the number of entries in the offset array (the next section).

b) Section 2 – Offset Array
Immediately following the header is an array of offsets. Each offset is an integer (4 bytes) that gives the offset from the start of the file to the start of each line (the next section).

c) Section 3 – Lines of data
The data in the file is a series of text lines, each preceded by an integer that indicates the length of the line (including the size of the integer). Following the length are the bytes of the file line. The lines are not terminated by null bytes.

A sample file (`Haiku.bin`) is provided in Oaks for testing. You can use the `xxd` utility to display the contents of the file in hex.

Your program will:
a) Take the name of a binary file to process as it's single parameter.
b) Read the contents of the binary file using only the `open()`, `close()`, `lseek()` and `read()` library functions.
c) Print to `stdout` the lines in the file.

I have a larger test file (think more Shakespeare) that I will be using to grade your program so don't depend on their only being a few lines in the file.

I recommend you:
a) Read the header
b) Allocate space for the offset array and read the offset array
c) Loop through the offset array, for each entry:
    a. `lseek()` to the appropriate point in the file
    b. Read the line length field
    c. Read the bytes of the line

Extra Credit (5 points)

Another approach is to memory map the file (as Linux does with shared libraries). Instead of reading the file, use the `mmap()` function to map the file into memory and process.