

LAB SIX

BINARY FILE IO

CS2263, Fall 2021

LEARNING OUTCOMES

At the conclusion of the lab, students should be able to

- Perform sequential and random file IO

SETTING UP

If a file is just a collection of contiguous bytes on a disk, and an array is just a collection of contiguous bytes on a disk, why can't they be treated in the same way? And what is the cost for doing this?

EXERCISE ONE

Create a program based on the `genPointsBin.c` program from lecture that creates a binary file of random integer values and outputs them to a file specified on the command line. The first value in the file should be the number of values in the file (yes, as a binary value), followed by binary integer values (one after another – no spaces, commas, line feeds – just the binary values).

```
$ genIntBin 10000 16ex1.bin
```

SUBMIT:

- A screen shot of the make command output for a successful compile
- A screen shot of the program's successful run for 10,000 values.

EXERCISE TWO

Create a program that reads the binary file from a name specified on the command line into a heap allocated array in memory and sorts it in place. Use whatever sorting algorithm you wish, so long as you write it yourself as a function and name it in your comments. Your function will take in a pointer to the array and it's filled length. Incorporate the technique from lecture of calculating elapsed time and measure the time it takes to sort the integer array. The program should write the values out to a binary file `16ex2.bin` once sorted.

```
$ sortInMemoryIntBin 16ex1.bin 16ex2.bin
```

SUBMIT:

- A screen shot of the make command output for a successful compile.
- A screen shot of the program's successful run.

EXERCISE THREE

Based on the program in Exercise Two, create a program that sorts a binary file (from a name specified on the command line) on the disk instead of in memory, using the same sorting algorithm. Your sorting function will take a pointer to an open file (`FILE*`) and internally will use random file io functions (`fseek()`, `ftell()`, `fread()`, `fwrite()`) to sort the values in the file. Incorporate the technique from L19 of calculating elapsed time and measure the time it takes to sort the file in place. The program should use a different file than what was used in Exercise Two.

```
$ cp 16ex1.bin 16ex3.bin
```

```
$ sortOnDiskIntBin 16ex3.bin
```

SUBMIT:

- A screen shot of the make command output for a successful compile.
- A screen shot of the program's successful run.
- A brief set of thoughts about the relative speeds of running the two sorting programs.

EXERCISE FOUR

Create a program that reads the binary file in the format we've specified from a file specified on the command line and reports it as text to standard out to demonstrate that your file is sorted.

```
$ reportIntBin 16ex3.bin
```

SUBMIT:

- A screen shot of the make command output for a successful compile.
- No screen shot of the program's successful run is required (since it'll be loooong!)

SUBMISSION

Before the due date for this lab, students should submit a single zip or tar file (named *LastName_FirstName_Lab6.zip* or *LastName_FirstName_Lab6.tar*) online to the lms containing:

- the required material for each question (use the headings indicating the question number) in a single pdf file (named *LastName_FirstName_Lab6.pdf*)
- Your source code directory:
 - This should include all of your source files, including any test programs.
 - This should not include object (.o) files and executables. Nobody needs to see those.