ECE 264: Advanced C Programming

Programming Assignment 5 - Files and Memory allocation, Due 7/9/19

1 Learning Goals

You will learn how to:

- Read unsorted data from files
- Write sorted data to another file
- Calculate the amount of memory needed to store the data
- Create structure to store data
- Allocate memory to store the data
- Understand the difference between * and * * in argument
- Use the C built-in qsort function to sort the data
- Use valgrind to check invalid memory access and memory leakCreate a type for functions that have the same types of arguments and return value

Please be aware that this you need to write a lot more for this assignment. Please start early. You are encouraged to study PA01 - PA03 thoroughly before starting this assignment. This assignment is designed with the assumption that you are familiar with the details in PA01 - PA03.

2 Background

2.1 File IO

There are two types of files: text and binary. A text file can be read and modified by using a text editor. A binary file cannot. The two types have advantages and disadvantages. This assignment asks you to write code that can read from and write to text files. In this assignment, each line of the input file has two pieces of information: a unique number (ID) and a name (first name followed by last name).

Before reading from or writing to a file, a program must call fopen to open the file with the proper mode (read, write, read and write, append). It is possible that fopen fails and your program must check that. If fopen fails and your program does not check, your program may enter an unknown state and fail in unexpected ways later.

Your program will use fscanf to read integers and strings. In this assignment, you can assume each person's name contains at most 79 characters (excluding '\0'). Please understand the difference between fscanf and fgets. They stop reading for different reasons.

After reading the input data, sort the data by either the IDs or the names by calling C's qsort function. After sorting the data, write the sorted result to another file. Your program can use fprintf to write the results.

2.2 Calculate the amount of memory needed

Your program needs to calculate number of lines in the input file and allocate enough memory for the data. You can assume that the input file has no empty line and use the number of '\n' to calculate the number of lines.

2.3 Create structure to store IDs and names

Structures allow programs to put related information together. This assignment asks you to create a structure with two attributes:

- an integer
- a string.

You can assume that the string (either first name or last name) is no longer than 80 characters. However, you should NOT write

```
char name[80];
```

Instead, you should create a symbol that defines this maximum length

```
#define NAME_LENGTH 80
```

and use NAME_LENGTH wherever you need to refer to this maximum length.

```
char name[NAME_LENGTH];
```

Why? If, for any reason, you decide to change the maximum length, there is only one place to change. When you write programs, you should expect the programs to change and make changes easy. If you do not define a symbol, then you may need to change multiple locations. The more changes are needed, the more mistakes you will make.

This symbol is defined in constant.h for you to use. It is possible that the grading program will modify the value (for example, define NAME_LENGTH as 160 or 57). If you write 80 anywhere in the program, your program may fail. If you use 80 directly anywhere in the program without creating a symbol, you may *lose up to 20 points* in this assignment.

2.4 Allocate Memory for the Students

You can assume that the input file has no empty line and the last line ends with and new line character ('\n'). Thus, you can count the number of lines to determine the number of students in the file. Your program should allocate an array of the structure type.

To allocate memory, you need to use a pointer. Suppose you want to allocate an array of type and the array has numelem elements.

```
type * ptr;
```

```
ptr = malloc(sizeof(ptr) * numelem);
```

Your program must release allocated memory before it terminates.

```
free (ptr);
```

If your program allocates memory without releasing it, the program has memory leak. Memory leak is a serious problem in programs. Some students believe memory leak is harmless but they are wrong. To ensure that you treat memory leak seriously, you will **lose 1 point** for every bye of leaked memory. In other words, if your program leaks 100 or more bytes, you will receive zero.

Please be aware that calling free does not set the released memory to zero (you should not care about that). Calling free also does not set ptr to NULL. In order words, you should expect the following:

```
free(ptr); // release the memory, does not set ptr to NULL
if (ptr != NULL)
{
    // true
}
```

However, you should **not** use the value of ptr any more. Using the value of ptr will cause segmentation fault.

2.5 Difference between * and * * in argument

Consider the following example:

int x = 5;

If we want to write a function that doubles the value of x and x is an argument, it is necessary to pass the address of x:

```
void f1(int * ptr)
{
    int t = * ptr;
    t = t * 2;
    *ptr = t;
}
f1(& x);
```

If we want to modify the value of a pointer, it is necessary to pass the address of the pointer.

```
int * ptr;
f2(& ptr);
```

In this case, f2's argument is

void f2(int * * addptr) // the address of a pointer

As an example, if a function allocates memory, the address of the allocated memory must be passed back to the caller:

```
int * ptr;
f2(& ptr);

void f2(int * * addptr)
{
    int * ptr = malloc(...);
    * addptr = ptr;
}
```

Adding space between the two * makes no difference:

void f2(int * * addptr)

is exactly the same as

void f2(int ** addptr)

2.6 Use qsort to sort the students

C has a built-in function for quick sort. This function sorts an array. To use qsort, you need to provide four arguments:

- the address of the first element
- the number of elements
- the size of each element
- the function for comparing elements

The comparison function has three steps

- cast the type from void * to the proper type (it is a pointer)
- retrieve the values from the pointer
- compare the values and return negative, zero, or positive numbers

You can find two examples here:

Comparing two elements in an array of integers

https://github.com/yunghsianglu/IntermediateCProgramming/blob/3eee24660f99a641cc2a445733bd154595ff1915/Ch9/comp

Comparing two elements in an array of string

https://github.com/yunghsianglu/IntermediateCProgramming/blob/3eee24660f99a641cc2a445733bd154595ff1915/Ch9/comp

Please use the numeric difference to compare two IDs. Please use strcmp to compare two names;

2.7 Use valgrind to check memory errors

Many types of mistakes may occur when writing a program like PA05. Valgrind is a tool that can check memory errors, such as uninitialized variables, invalid indexes, and memory leaks.

To use a valgrind to check a program foo for memory errors, run:

> valgrind –tool=memcheck –leak-check=full foo

The output will tell you how much memory you leaked (if any), and which calls to malloc were the ones that did not get freed.

This page gives details of the various types of errors you might have.

You must use valgrind to check your program before submission. You will

- lose 1 point for every byte of leaked memory
- lose all points if your program terminates abnormally (e.g., segmentation fault)
- lose 1 point for every error detected by valgrind even though the program continues

Please be aware that if the program has one mistake, it is likely that same mistake is triggered multiple times and valgrind reports the same error multiple times. When this occurs, you will lose one point for every detected error.

2.8 How to write correct programs, faster?

For most ECE 264 students, this assignment may be a significant challenge. You have to pay attention to many details. What you should do is to **plan before coding**. You need to think about how to write the program and how to test your program.

Do **not** assume your program will work. Some students write a lot of code without testing any piece. When they put everything together, the programs do not work and the students do not know what is wrong. Instead, you should **assume your programs have many mistakes** and develop a plan to ensure that every piece is correct before putting them together. You should have a plan before writing the first line of code. If you start coding without a plan, you will spend much more time debugging.

Your plan should guide the order of implementation. Should you implement reading the file first, writing the file first, or sorting the data first? Focus on one piece, fully debug it, before working on the next. If you start writing every function simultaneously, it is possible that none works and you receive no point.

It is very likely that you need to write additional code for testing but the testing code is **not** graded. If you expect that every line of code will be used for grading, it is likely that you do not fully test your program and you will lose points.

There is no definite answer how much additional testing code you need to write. A rule of thumb is that for every line you need to submit, you need to write three to five additional lines for testing.

You will likely need to write 'driver" and 'stub" for testing. Imagine that function A needs function B. If function A is not ready, how can you test function B? You need to write another function that may provide some meaningful data to test B. In this case, you are writing a driver. Similarly, if B does not work yet, how can you test A? You will write a simplified version for B so that you can test A. This is a stub.

You should fully understand the advantage of using Makefile for testing. One common problem is that students do not take advantage of Makefile for testing. You should read Makefile from PA01-04 carefully and think about how to run multiple tests at once by using Makefile.

2.9 Plan, Plan, Plan

The best advice to finish this assignment quickly is to plan before coding. You will save a lot of time if you have a plan. Many students started coding without any plan. They failed. You will not be the first exception.

2.10 Validate your answers

The test case given to you is too large to check by eyes. You should have a plan to check whether your output is correct by using computer programs. There are at least two ways: (1) you can write a function to check and (2) you can use the sort program in Linux to check.

For (1), you can do something like. Suppose numelm is the number of elements and you want the array values to be sorted in the ascending order.

```
for (int ind = 0; ind < (numelm -1); ind ++) // remember -1
{
    if (values[ind] >= values[ind +1])
    {
            // out of order
        }
}
```

For (2), you can use sort command in Linux. Suppose infile is the name of an input file.

> sort infile

sorts the file, line by line.

You can use -k to specify which column to sort.

3 What do you need to do?

You will have to modify the following files:

- student.h: This file defines a structure named Student
- student.c: This file defines several functions (declared in students.h) to:
  - Read a list of Students from a specified input file and store them in an array (you will need to determine how big an array to allocate!)
  - Write an array of Students to a specified output file.
  - Sort arrays of students by ID, first name or last name.
- pa05.c: The main function for pa05. This function should:
  - Read students from the input file into an array of Students (you will need to determine how big an array to allocate!)
  - Sort the students by IDs
  - Write the results to an output file
  - Sort the students by first name
  - Write the results to a second output file
  - Sort the students by last name
  - Write the results to a third output file

  See the pa05.c template file for more details.
- Makefile: You will have to write a Makefile that has four targets:
  - all: This will create an executable called pa05
  - test: This will run three input files (see the template Makefile for details) and then use the diff command in Linux to compare your programs output with the program's output.
  - memory: This will call valgrind on your program to check for memory errors.
  - clean: Remove all .o and executable files

Don't forget to pass in -D flags for TEST_READ, TEST_WRITE, TEST_SORTID, TEST_SORTFIRSTNAME, TEST_SORTLASTNAME so that the code you write in student.c is included.