Out: Jan 23, 2019
Due: Feb 6, 2019, (start of class, paper submission)
34 available marks, marked out of 30.
10% per day late penalty, submission via BlackBoard

CENG 251 Lab 2: Memory

For this lab you are to keep a time log just as you did in Lab 1. This time it is worth 2 marks.
If you completed the lab say so. If you did not list the sections you completed. (2).

Parts I, II and III should all be separate programs listings. Parts that are reused between the programs such as the declaration of the struct should be placed in a separate include file.

Part I: Exploring the Memory Layout of a Struct (9 marks)

In class we explored the example of **struct FLIGHT** using the **sizeof** and **&** operators and the **offset** function. Write a program to accomplish the following:

1. Create and declare your own data structure consisting representing some kind of idea, ie: book, employer, student, product, beverage, planet, game, document, order, schedule……etc . Your data structure should consist of 6 fields with at least 3 different data types, one of which is a character string at least 9 characters long with an odd number of characters. Neither the field before it nor after it should be char or bool. It should not be the last field but it can be the first. (I'll make the reason for this clearer later on.) None of your fields should be pointers. Give appropriate meaningful names to your fields.

   Given these requirements it's unlikely that any two students will have the same declaration. (2)

2. Write a program using the **sizeof** operator to determine the sizeof each of your fields and the size of the whole data structure. **Is the sum of the parts equal to, larger than or smaller than the whole?** (2)

3. Use the **offset** function to determine the offsets of each of the fields in your struct. Use this and the above information to draw a data structure diagram. (2 marks)

4. Recompile your program using the **-fpack-struct** option and redraw the diagram. In words describe where the space saving occurred. (If there is no difference in the size, talk to me. The instructions in #1 are supposed to force a difference in size.) (3)

   To hand in: Program listing, output, data structure diagram, statement regarding sizes.

Part II: Creating a Database from your struct (15 marks)

1. Create a function that generates sets of random data for your struct. The example **createPeople** should give you an idea of how to do this. (3)

2. Create a function that uses **fprintf** to display your struct to the screen. Pass the structure by address to your function as an argument. (No global variables.) Create a small program to test 1 & 2. This should be demonstrated at the start of the 2nd lab. (2)

3. Set an environment variable **NRecords** in the shell using the **export** command and another pair called **MyDB_FP and MyDB_FD** representing 2 file names name. Create a 4th environment variable **DBMODE** and set it to either "APPEND" or "OVERWRITE". Retrieve all 3 values at the start of your program. If any of them are missing, display an error code and quit.

   Hint: put the 4 definitions in a script file and run the script using the **source** command. This way you can set these values quickly each time you log in and you can use the script as documentation for your settings in your report. (1)

4. Creating a database of your datatype (6 marks)

   If **DBMODE** is OVERWRITE then use fopen to open the file represented by **MyDB_FP** in write mode: "w". This will create/replace a new file. If **DBMODE** is APPEND then open the file in append mode "a". Write a main program that retrieves **NRecords** and generates that many sets of your random object, displays them as text on the screen and writes them in binary to the file name stored in your database file using fwrite. When you are done, close the file (3)
   a. test your program in both OVERWRITE and APPEND mode. Report on the resulting file sizes. (1)
   b. In week 2 we will demonstrate the Unix utilities **od** and **hexdump.** Write a short bash shell script with one parameter *n* that will display the fields of the *nth* record, one per line. You will need 6 commands (1 per field) to do this. The output for each field should be the appropriate data type. If extra zeroes are shown at the end of strings, don't worry about it. (2)

5. Write a 2nd loop in your program that uses file descriptors rather than file pointers to open and write the same number of records to the file represented by **MyDB_FD**. The flags for open are O_WRONLY, O_APPEND and O_CREAT. Examine the file sizes and content and report on your success. (3)

Part III: Address Arithmetic/Memory Layout (8 Marks)

Record all gdb commands used. Clearly label which question each set of gdb commands come from. I suggest using gdb's history feature so that you have a record of commands. (The commands used to create your history should also be recorded! Your marks will be based on your answers and your record of the gdb commands used.)

1. Include your data structure declaration from Part I in a copy of addressArithmetic3.c (2 marks)
    a. Create an array of size 1000 of your datatype as a global variable. Initialize the 5th element of the array with a set of sample values for your datatype. In main, as your first executable statement, initialize the 3rd element of the array to a different set of values. Compile to verify that your are using the correct syntax and hand in a screen shot of this part of your code.

    b. Load your program in gdb and set a breakpoint on the 2nd executable statement. run the program and when the program stops display the 2 initialized elements.

2. Display values from the array numbers using subscripts from 6 to 10 and from -1 to -6. Identify from which array (if either) each of these values come from.

    Modify your commands in #2 so that instead of using numbers[i] you use the pointer dereferencing notation *(numbers+i) to refer to the array. Then try the notation i[numbers]. Are the results the same? (2 marks)

3. Continue running your program to the start of the loop. Print out and record the memory locations for the following variables and next to each indicate if they are stored in the stack, heap, text, bss or data sections . At the end write up a short justification of your answers: (4 marks)
    a. The global array of your data type.
    b. numbers, before, after
    c. newstring1, newString2, newString3
    d. fprintf, malloc, i