CSC235 – Final Project (#5), struct

Assignment:	Write a program that will store and retrieve information using an array of structs.
Supplied:	A folder containing a Visual Studio project that is set up for a MASM program. An example input data file.
Location:	**/export/home/public/carelli/csc235/Projects/Project5**
Deliverable:	The Visual Studio solution/project folder containing the completed program.
Due:	The programs MUST be turned in by the assigned date and time using D2L. Late submissions will, in fact, be rejected by D2L, resulting in a grade of zero.

Overview:

In this project, you will complete a program that will read information from a file, load it into an array of structs, and then display the information on the screen.

Your starting point will be a VS project with a pre-defined structure called *olympian* and a main routine that calls a procedure called *loadFile*, which is also pre-defined. *loadFile* will prompt for and read an input file that contains statistics about Olympic athletes. It will store the information read into an array in memory.

Your task, specifically, is to write and call procedures, described below, to load an array of *olympian* structs with the information placed previously into the memory buffer by *loadFile.* You will then output the data in the struct array for display on the terminal.

A properly formatted text file, called *input.txt*, is also provided for initial testing purposes. You will need to write additional tests as described below.

Requirements:

You will need to write a number of procedures. As mentioned, you are supplied with a procedure called *loadFile*. We will go over this procedure in class. You should not modify the data once it is loaded into memory. Just use it as is to populate your structs.

Note that you are *required* to comment your procedures in a manner similar to what is provided - including a header and individual line comments.

All procedures should make use of stack frames to receive input parameters (as discussed in class and in Chapter 8 of the text - see *loadFile*). Where it is necessary to output information to the console, use procedures supplied in the Irvine32 library. Note that this also implies that we are using the STDCALL calling convention, as opposed to the C calling convention (sec 8.2.4).

Following is a list of procedures you need to write and use in your program:

bufferCopy:
This procedure copies bytes from a supplied input BYTE array to a target BYTE array. It continues copying bytes until it encounters a carriage return character (0Dh). In the target array, it substitutes a NULL character for the c/r to produce a well-formatted string, and then, in the source array, skips over the next character, which is assumed to be a line feed character (0Ah). It returns a pointer to the character following the skipped line feed. This will enable successive calls to *bufferCopy* to sequentially step through the file data stored in memory.

Receives:

- pointer to the input BYTE array
- pointer to the output BYTE array
- maximum number of bytes to copy (to avoid a runaway condition)
  The max number of bytes should be set to easily accommodate the maximum length of a line in the input data file (LINESIZE).

Returns (in eax):

- The pointer to the next character in the input array after the carriage return and line feed characters (i.e. the next valid character to be read in the buffer).

loadOlympian:
This procedure reads information from the memory array populated using *loadFile,* and loads it into an *olympian* struct. You should sequentially call *bufferCopy* to transfer each data item into the Olympian struct.
Loading the medal counts will require special attention. You will first need to create and load a local string variable (BYTE array) with the medal count using *bufferCopy*. Once the count is loaded into the local string, *ParseInteger32* (from the Irvine32 library) can be used to convert the medal information in the string to a DWORD that can then be loaded into the struct.

Receives:

- pointer to the beginning of a struct object
- pointer be the beginning of information in the buffer for the next athlete
- maximum number of bytes to read in each transfer (pass to *bufferCopy*).

Returns (in eax):

- Pointer to the next Olympian (athlete) in the buffer

loadAllOlympians:
This procedure should make successive calls to *loadOlympian* to transfer data from the memory buffer into the array of *olympian* structs.
Loading should stop if the next character encountered after an Olympian has been loaded is an asterisk (*). If an asterisk is not encountered after MAXNUM (5) Olympians have been loaded, loading should stop (i.e. no more than MAXNUM Olympians).

Receives:

- pointer to the beginning of the struct array
- pointer to the start of the buffer containing the data read from the file
- Maximum number of bytes to read on each line (pass to *loadOlympian*).

Returns (in eax):

- Number of Olympians read

outputOlympian:
This procedure outputs the contents of one olympian struct to the console in a well formatted manner. The output should look like the example at the end of this document. Note that it will be necessary to compute a total medal count as this is not supplied as an input.

Receives:
- pointer to the beginning of a struct object.

Returns
- There is no return value

Note: strings for output formatting are supplied. *You may use the supplied output formatting strings as globals* (you do NOT need to pass them in as parameters)

outputAllOlympians:
Output the entire array of Olympians to the console by successively calling *outputOlympian.*

Receives:
- pointer to the first Olympian struct.
- Number of Olympians to output

Returns:
- There is no return value

Do not do any looping in the main program! Aside from *loadFile*, the main program should only call *loadAllOlympians* to populate the internal array of structs and *outputAllOlympians* to output the data.

DO NOT use global variables, except for the following:
- pre-defined symbolic constants (for example: MAXNUM, ASTERISK, …)
- pre-defined output formatting strings (for example: outname, outtotal, …)

Input file format:
The format for the input file is extremely important in the assignment since we are working with very rudimentary reading routines. In order to simplify the task of reading the data, each data item is on a separate line. DOS/Windows format is used in the file, which is a simple text file. This means that each line of data is terminated with two characters, a carriage return and a line feed. This is different from the format used in Unix/Linux, in which each data line is terminated with ONLY a carriage return.

Do *not* use a file formatted for Unix – it will not work! (You should create/edit test files with a Windows utility like Notepad – don't copy something from Unix!)

Finally, avoid introducing extra characters (spaces, tabs, etc) and be sure to end the file with an asterisk (*), on a separate line, since this is what is looked for as a terminating character.

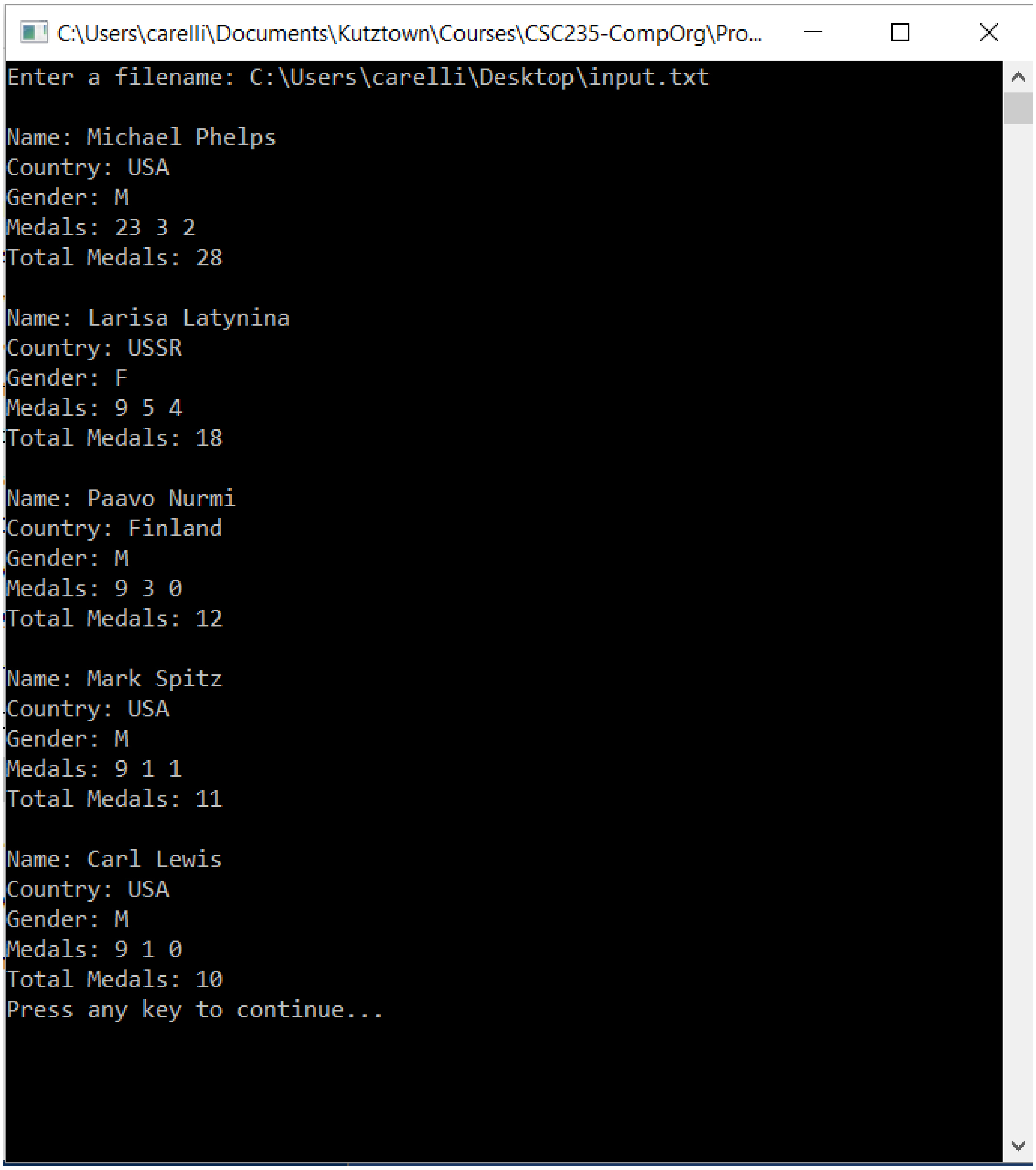
Some tips and suggestions:

- *Build the program up in steps, one procedure at a time*
- *Test each procedure as you write it*
- *Use the debugging facilities available in Visual Studio*
    - *Monitor values in the registers*
    - *Monitor the input buffer by viewing the data stored in memory*
        - *using memory dereferencing (with an ampersand) to access memory locations (Ex: &buffer)*
    - *Monitor the struct array by viewing the data stored in memory (&olist)*
- *Comment as you write!*
    - *It makes it easier to keep track of what you've written*

Grading Considerations:

1. Program must compile
2. Use symbolic constants where appropriate
3. Use Irvine32 procedures for user input and output display
4. Write *all* of the specified procedures
5. Use stack frames to pass information to procedures
    a. With specified inputs and outputs
    b. Make use of a local variable, as directed
    c. No globals (except for output formatting strings)
6. Invoke all procedures as directed (no looping in main)
7. Test thoroughly
    a. With files containing various numbers of Olympians
    b. Make sure loading stops at MAXNUM entries or on a terminating asterisk (*)
8. Be sure to halt execution before terminating (so output can be viewed)
9. Use consistent formatting
    a. With easily readable indentation and code labeling
    b. Follow code formatting example in *loadFile*!
10. Comment liberally!!!
    a. Provide a PROC header section describing information passed to/from the procedure and the registers used
11. Turn in the **entire Visual Studio solution folder**, on time, to D2L
    a. Do **NOT** turn in test files

Example Output:



C:\Users\carelli\Documents\Kutztown\Courses\CSC235-CompOrg\Pro...

```
Enter a filename: C:\Users\carelli\Desktop\input.txt

Name: Michael Phelps
Country: USA
Gender: M
Medals: 23 3 2
Total Medals: 28

Name: Larisa Latynina
Country: USSR
Gender: F
Medals: 9 5 4
Total Medals: 18

Name: Paavo Nurmi
Country: Finland
Gender: M
Medals: 9 3 0
Total Medals: 12

Name: Mark Spitz
Country: USA
Gender: M
Medals: 9 1 1
Total Medals: 11

Name: Carl Lewis
Country: USA
Gender: M
Medals: 9 1 0
Total Medals: 10
Press any key to continue...
```