Assignment 7
C/C++ Programming II

C2A7 General Information

Assignment 7 consists of FOUR (4) exercises:

C2A7E1   C2A7E2   C2A7E3   C2A7E4

All requirements are in this document.

Get a Consolidated Assignment 7 Report (optional)

If you would like to receive a consolidated report containing the results of the most recent version of each exercise submitted for this assignment:

`Send an empty-body email to the assignment checker with the subject line **C2A7_177752_U09845800** and no attachments.

Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report requests as many times as you wish before the assignment deadline.

--- No General Information for This Assignment---

1 **C2A7E1 (6 points – C Program)**

2 Exclude any existing source code files that may already be in your IDE project. Then make a copy of
3 instructor-supplied source code file **C2A7E1_main-Sample.c**, name that copy **C2A7E1_main.c**, and add
4 that copy to the project. This is the only source code file to be used in this exercise.
5
6 This exercise is designed not only to familiarize you with binary tree and hashing concepts but to also
7 illustrate the frustration of dealing with someone else's code.   The code in the renamed
8 instructor-supplied driver file that you added to your IDE project above represents working but slightly
9 modified versions of both the "binary tree" and the "hashing" code from section 15 of the course book.
10 If macro `TREE` is defined the "binary tree" portion of code will be compiled and if not, the "hashing"
11 portion will be compiled instead. Do the following, in order:
12
13   1. Verify that the "binary tree" portion of the code works by ensuring that macro `TREE` is defined
14   then compiling and running the program, noting that the desired input file name must be
15   specified on the command line. Instructor-supplied data file **TestFile1.txt**, which must be placed
16   in the program's "working directory", has been provided for this purpose.
17
18   2. Verify that the "hashing" portion of the code works by commenting out the definition of macro
19   `TREE` then compiling and running the program, noting that in addition to the desired input file
20   name the desired number of bins must be specified after it as an additional command line
21   argument. Test with the same input file as above and a bin count of **10**, as well as with any
22   other desired text files and bin counts.
23
24   3. Combine, modify, add to, and delete the supplied code in any way you deem necessary so
25   that it will perform the same "hashing" operation as before, but will store the words in ordered
26   binary trees like those in the "binary tree" portion of the code instead of in singly linked lists.
27   <mark>**Significant credit will be deducted for unused code that is not deleted from your file.**</mark>   To permit
28   automated testing the following things **must not** be changed:
29   a. The input file name and bin count must still come from the command line.
30   b. The display format (spacing, field width, etc.) must be the same as in the original
31   "hashing" version.
32   c. As in the original "hashing" version, the order of dynamic allocation must remain as
33   follows. There must be no dynamic allocations other than these:
34   1) Allocate memory for a "Table Descriptor" (just once);
35   2) Allocate memory for an empty "Hash Table" (just once);
36   3) If and only if a word is encountered that is not already in the table, allocate
37   memory for a new node **first**, then memory for its data (the string).
38
39   4. As is always the case, **do not** use external variables in this or any other C exercise (and **do not**
40   use non-const external variables in any C++ exercise).
41
42 If your code is working properly the display it produces will be identical to that of the original "hashing"
43 version except for the order of the words in each bin, which will now be in the order dictated by the
44 standard library `strcmp` function. That is, for data file **TestFile1.txt** and a bin count of **10** the display will
45 start exactly as follows:
46   `6 entries for bin 0:`
47   `    1   arguments.`
48   `    1   constants.`
49   `    1   expansion)`
50   `    1   invocation`
51   `    1   occurrence`
52   `    1   parameters`
53   `6 entries for bin 1:`
54   `    6   a`

```
55          1   combination
56          1   definition,
57          1   definition.
58          1   number-sign
59          1   stringizing
60
61
```

62 ## Submitting your solution

63 `Send an empty-body email to the assignment checker with the subject line **C2A7E1_177752_U09845800**
64 and with your modified source code file <u>attached</u>.

65 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
66 *formatting, submission, and assignment checker requirements.*
67

68
69 **Hints:**
70 Whenever you add a new node to a tree you must also increment the **nodes** member of the bin
71 descriptor structure that represents that tree, but how do you access that member? Simply add a 3rd
72 parameter that points to that structure to the **BuildTree** function!
73
74 For my solution I only added/modified about 6 lines of code, deleted about 50 lines of code, and
75 moved a few things around.

1 **C2A7E2** *(4 points – C++ Program)*

2 Exclude any existing source code files that may already be in your IDE project and add two new ones,
3 naming them **C2A7E2_OpenFileBinary.cpp** and **C2A7E2_ListHex.cpp**. Also add instructor-supplied
4 source code file **C2A7E2_main-Driver.cpp**. <u>Do not write a `main` function!</u> `main` already exists in the
5 instructor-supplied file and it will use the code you write.
6
7 File **C2A7E2_OpenFileBinary.cpp** must contain a function named `OpenFileBinary`.
8 `OpenFileBinary` syntax:
9   **void** OpenFileBinary(**const char** *fileName, ifstream &inFile);
10 Parameters:
11   `fileName` – a pointer to the name of the file to be opened
12   `inFile` – a reference to the **ifstream** object to be used to open the file
13 Synopsis:
14   Opens the file named in `fileName` in the read-only <u>binary</u> mode using the `inFile` object. If the
15   open fails an error message is output to **cerr** and the program is terminated with an error exit code.
16   The error message must mention the name of the failing file.
17 Return:
18   **void** if the open succeeds; otherwise, the function does not return.
19
20
21 File **C2A7E2_ListHex.cpp** must contain a function named `ListHex`.
22 `ListHex` syntax:
23   **void** ListHex(ifstream &inFile, **int** bytesPerLine);
24 Parameters:
25   `inFile` – a reference to the **ifstream** object for a file that is open in a readable binary mode
26   `bytesPerLine` – the number of bytes to display on each line while bytes are available
27 Synopsis:
28   Displays the contents of the file in **inFile** as one pair of hexadecimal characters per file byte,
29   zero-filled on the left if necessary to produce the two characters. Pairs are single-space-separated
30   and the number of pairs that are placed on each line is specified by parameter **bytesPerLine**.
31   Pairs are aligned from one line to the next and the last line will contain less than the specified
32   number of pairs per line if EOF is reached prior to the completion of that line. You may assume that
33   a byte consists of 8 bits for this exercise.
34   • **Do not attempt to determine the actual length of the file – it's totally unnecessary.**
35   • **If your algorithm reads 1 byte at a time, don't use an array.**
36   • **Do not use functions** peek, seekp, seekg, tellp, tellg, fseek, ftell, fsetpos, **or** fgetpos.
37 Return:
38   **void**
39
40 The case of the letters does not matter. Typical output from `ListHex` with 16 bytes per line must look
41 like:
42   00 ab 05 72 fe 01 03 67 68 69 20 40 78 0a 0d 02
43   ab 05 72 fe 01 ab 05 72 fe 01 20 40 78 67 68 69
44   fe 01 03 67 68 69 20
45
46 The instructor-supplied driver file that calls `ListHex` requires that you provide the desired input file name
47 followed by the number of bytes per line as command line arguments.
48
49 Manually re-run your program several times, testing with at least instructor-supplied data files **TestFile3.txt**
50 and **TestFile4.bin**, each of which must be placed in the program's "working directory". Try several
51 different bytes-per-line values.
52

53 **Submitting your solution**

54 `Send an empty-body email to the assignment checker with the subject line **C2A7E2_177752_U09845800**
55 and with all three source code files <u>attached</u>.

56 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
57 *formatting, submission, and assignment checker requirements.*

58
59
60 **Hints:**
61   1. Display the value of each and every byte in the file but do not display the values of any bytes
62   that are not actually in the file.
63
64   2. Do not display EOF since it is not a character in the file and cannot be displayed meaningfully.
65
66   3. Use the `setw` and `setfill` I/O manipulators to obtain the correct field widths and fill characters,
67   respectively. `setw` applies only to the next item to be output whereas `setfill` remains in effect
68   until explicitly called again. `setw` is discussed in the course book and information on both is
69   readily available in any C++ text or online.
70
71   4. Due to "sign extension" bytes whose values are from `0x80` through `0xFF` may be printed with
72   unwanted leading `F`s unless they are stored in or typecast to **unsigned char** before printing.
73   Bitwise ANDing the values with `0xFF` prior to printing is another, but less efficient, way to avoid
74   this problem.
75
76   5. Students sometimes ask how to avoid placing a space before the first entry on a line or after the
77   last entry on a line. The answer is to write code to detect when an entry is first or last and do not
78   put a space there.
79
80   6. If you attempt to read the entire file into memory at once your program is not portable since it is
81   possible that an arbitrary file might contain more data than the memory available to your
82   program can hold.

1 **C2A7E3** *(4 points – C Program)*

2 Exclude any existing source code files that may already be in your IDE project and add a new one,
3 naming it **C2A7E3_ReverseEndian.c**. Also add instructor-supplied source code file
4 **C2A7E3_main-Driver.c**. <u>Do not write a `main` function!</u> `main` already exists in the instructor-supplied file
5 and it will use the code you write.

6

7 File **C2A7E3_ReverseEndian.c** must contain a function named `ReverseEndian`.
8 `ReverseEndian` syntax:
9   **void** *ReverseEndian(**void** *ptr, size_t size);
10 Parameters:
11   `ptr` – a pointer to a scalar object whose endianness is to be reversed
12   `size` – the number of bytes in the object
13 Synopsis:
14   Converts any scalar object pointed to by `ptr` from big endian to little endian (or vice versa) by
15   swapping its bytes. `ReverseEndian` cannot detect if the object is actually scalar but it will not
16   produce the desired results if it is not. (See the Hint below if you do not know what a scalar type is.)
17 Return:
18   `ptr`

19

20

21 **Submitting your solution**

22 `Send an empty-body email to the assignment checker with the subject line **C2A7E3_177752_U09845800**
23 and with both source code files <u>attached</u>.

24 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
25 *formatting, submission, and assignment checker requirements.*

26

27

28 **Hints:**
29 What is a "scalar" type? Simply speaking, a "scalar" type is any arithmetic, pointer, or enumeration
30 type. Arrays, Structures, Classes, and Unions are not scalar types.

---

1 **C2A7E4 (6 points – C Program)**

2 Exclude any existing source code files that may already be in your IDE project and add three new ones,
3 naming them **C2A7E4_ReverseEndian.c**, **C2A7E4_ProcessStructures.c**, and
4 **C2A7E4_OpenTemporaryFile.c**. Also add instructor-supplied source code files **C2A7E4_Test-Driver.h** and
5 **C2A7E4_main-Driver.c**. Do not write a `main` function! `main` already exists in the instructor-supplied
6 implementation file and it will use the code you write.

7
8 The purpose of this exercise is to illustrate the endian conversion of arbitrary scalar objects within an
9 aggregate type, such as a structure or class. If it is arbitrarily assumed that a **long** is 4 bytes and a **short**
10 is 2 bytes, a structure defined and initialized as
11
12   **struct** { **long** height; **short** width, depth; } box = {0x01234567L, 0x89ab, 0xcdef};
13
14 and written into a file using
15
16   fwrite(&box, **sizeof**(box), 1, fp);
17
18 will be written exactly how it appears in memory, which is in one of the two following orders, depending
19 upon the machine's endianness. There can be an arbitrary amount of implementation-dependent
20 padding after any members:

	Big Endian	Little Endian
21 | | | |
22 | | | |
23 | First byte in memory & in the file: | 01 | 67 |
24 | | 23 | 45 |
25 | | 45 | 23 |
26 | | 67 | 01 |
27 | | ----- possible padding here ----- | |
28 | | 89 | ab |
29 | | ab | 89 |
30 | | ----- possible padding here ----- | |
31 | | cd | ef |
32 | | ef | cd |
33 | | ----- possible padding here ----- | |

34
35
36 A structure of data type **struct** Test is what you will be using in this exercise. It is defined as
37   **struct** Test { **float** flt; **double** dbl; **void** *vp; };
38 in instructor-supplied header file
39   **C2A7E4_Test-Driver.h**
40 Any file that uses this data type must include this header file using **#include**. Do not define this data
41 type separately from this header file.

42
43
44 File **C2A7E4_ReverseEndian.c** must contain a copy of the same `ReverseEndian` function you wrote for
45 the previous exercise.

46
47 File **C2A7E4_OpenTemporaryFile.c** must contain a function named `OpenTemporaryFile`.
48 `OpenTemporaryFile` syntax:
49   FILE *OpenTemporaryFile(**void**);
50 Parameters:
51   none
52 Synopsis:
53   Opens a temporary file using the standard library `tmpfile` function. If the open fails an error
54   message is output to `stderr` and the program is terminated with an error exit code.

55   Return:
56   A pointer to the open file if the open succeeds; otherwise, the function does not return.
57
58   File   **C2A7E4_ProcessStructures.c**   must   contain   functions   named   `ReverseMembersEndian`,
59   `ReadStructures`, and `WriteStructures`.
60
61   `ReverseMembersEndian` syntax:
62   **struct** Test *ReverseMembersEndian(**struct** Test *ptr);
63   Parameters:
64   `ptr` – a pointer to the structure whose members' endiannesses are to be reversed
65   Synopsis:
66   Calls function `ReverseEndian` once for each member of the structure in `ptr`, thereby converting
67   each from big endian to little endian or vice versa. A loop cannot be used to access the members
68   nor can `ReverseEndian` reverse the endianness of the entire structure at once.
69   Return:
70   `ptr`
71
72   `ReadStructures` syntax:
73   **struct** Test *ReadStructures(**struct** Test *ptr, size_t count, FILE *fp);
74   Parameters:
75   `ptr` – a pointer to where the structure(s) that are read will be placed in memory
76   `count` – the number of structures to read
77   `fp` – a pointer to an open binary file containing the structure(s) to be read.
78   Synopsis:
79   Using <u>no loops</u> and <u>no recursion</u> it reads `count` structures from `fp` and stores them in memory starting
80   at address `ptr`. If `count` structures cannot be read an error message is output to `stderr` and the
81   program is terminated with an error exit code.
82   Return:
83   `ptr` if `count` structures are read; otherwise, the function displays an error message and terminates
84   the program with an error code.
85
86   `WriteStructures` syntax:
87   **struct** Test *WriteStructures(**const struct** Test *ptr, size_t count, FILE *fp);
88   Parameters:
89   `ptr` – a pointer to where the structure(s) that are written will come from in memory
90   `count` – the number of structures to write
91   `fp` – a pointer to an open binary file into which the structure(s) will be written.
92   Synopsis:
93   Using <u>no loops</u> and <u>no recursion</u> it reads `count` structures from memory starting at address `ptr` and
94   writes them to the file `fp`. If `count` structures cannot be written an error message is output to `stderr`
95   and the program is terminated with an error exit code.
96   Return:
97   `ptr` if `count` structures are written; otherwise, the function displays an error message and terminates
98   the program with an error code.
99
100
101   • Do <u>not</u> use dynamic memory allocation.
102   • Do <u>not</u> make any assumptions about the number of bytes in any data type.
103   • Do <u>not</u> make any assumptions about the presence, absence, value, or amount of padding.
104   • Please answer the following questions about your results and place these answers as comments in
105   the title block of the file containing the `ReverseMembersEndian` function:
106   1. Were the results you got correct for your implementation?
107   2. How many padding bytes were in your structure?
108

109 **Submitting your solution**

110 \`Send an empty-body email to the assignment checker with the subject line **C2A7E4_177752_U09845800**
111 and with all five source code files <u>attached</u>.

112 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
113 *formatting, submission, and assignment checker requirements.*

114
115
116 **Hints:**

117
118 **Having trouble getting the** `tmpfile` **function to work?**
119 If you are running Windows Vista or newer the failure of `tmpfile` is probably because your program
120 lacks administrative privileges. It is important to realize that being logged onto the computer with
121 administrative privileges does not automatically bestow those privileges on programs you run. Instead, it
122 merely permits you to bestow them. If you are running your program from within the Visual Studio IDE it
123 will not have administrative privileges unless Visual Studio itself does. To accomplish this start Visual
124 Studio by right-clicking its icon or Start Menu item and selecting "Run as administrator". If you are
125 running your executable program directly from its own icon or .exe file, right-click that icon or `.exe` file
126 and select "Run as administrator".