Assignment 8
C/C++ Programming II

C2A8 General Information

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

Get a Consolidated Assignment 8 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 email to the assignment checker with the subject line **C2A8_ID**, where **ID** is your 9-character UCSD student ID. 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.

1 <mark>**C2A8E1** *(10 points – C++ Program)*</mark>

2 Exclude any existing source code files that may already be in your IDE project and add a two new ones,
3 naming them **C2A8E1_OpenFiles.cpp** and **C2A8E1_MergeAndDisplay.cpp**. Also add instructor-supplied
4 source code file **C2A8E1_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

8 File **C2A8E1_OpenFiles.cpp** must contain a function named `OpenFiles`.
9 `OpenFiles` syntax:
10   `ifstream *OpenFiles(char * const fileNames[], size_t count);`
11 Parameters:
12   `fileNames` – a pointer to the first element in an array representing the names of text files to be
13   opened. The array has the following standard ragged array format:
14   `char *fileNames[] = { "fileA", "fileB", etc. };`
15   `count` – the number of elements in `fileNames`
16 Synopsis:
17   Dynamically creates an array of `ifstream` objects having `count` elements then uses those objects
18   to open the files named in `fileNames`, in order. All opens are in the read-only text mode. If any
19   open fails all previously opened files are explicitly closed, the dynamic allocation is deleted, an error
20   message is output to `cerr`, and the program is terminated with an error exit code. The error
21   message must mention the name of the failing file. If `count` is zero an error message to that effect is
22   output to `cerr` and the program is terminated with an error exit code.
23 Return:
24   a pointer to the first entry in the `ifstream` array if `count` is non-zero and all opens succeed;
25   otherwise, the function does not return.

26
27

28 File **C2A8E1_MergeAndDisplay.cpp** must contain a function named `MergeAndDisplay`.
29 `MergeAndDisplay` syntax:
30   `void MergeAndDisplay(ifstream files[], size_t count);`
31 Parameters:
32   `files` – a pointer to the first element in an array of `ifstream` objects having `count` elements, where
33   each object represents a text file open in the text mode for reading.
34   `count` – the number of elements in the array in `files`
35 Synopsis:
36   Proceeding in order from the first file specified in `files`, the first line in each file is read and
37   displayed, followed by the second line in each, followed by the third, etc. When the end of any file
38   is reached that file is closed and the process continues using only the remaining open files until all
39   files have finally been closed. Empty lines are displayed as empty lines. Empty files are simply
40   closed and ignored.
41 Return:
42   **void**

43
44

45 1. Functions `OpenFiles` and `MergeAndDisplay` must be able to handle any number of files specified
46   by their `count` parameter.
47 2. You may assume that no line in a file will contain more than 511 characters.
48 3. Do <u>not</u> display anything other than the exact contents of the files, i.e., no file names, line numbers,
49   extra spaces, extra blank lines, etc.
50 4. Do <u>not</u> attempt to store the entire contents of any file at once.
51 5. Do <u>not</u> attempt to read a file after reaching its EOF.
52 6. Do <u>not</u> attempt to use data you "think" you read from a file before verifying that the read was
53   actually successful.
54

        Page 1 of 2 of C2A8E1

1   Manually run your program twice, specifying the names of the instructor-supplied data files listed below
2   on the command line in the order shown. Each must be placed in the program's "working directory".
3   DO NOT prompt the user for the file names or place them in your program code.
4
5   Command line file names for test 1:   **mFile1.txt mFile2.txt mFile3.txt mFile4.txt mFile5.txt**
6   Command line file names for test 2:   **mFile3.txt mFile2.txt mFile1.txt**
7
8   Example:
9   If the command line specifies files **f1 f2 f3** and those files contain the following text:
10
	f1	f2	f3
Line 1:	Hello from	Bah bah black	Now is
Line 2:	the	sheep	the
Line 3:	other side of	<EOF>	time for all
Line 4:	*<blank line>*		good men and
Line 5:	the universe<EOF>		<EOF>

18   the display would be as follows, where ***<blank line>*** represents an actual blank line:
19
20   Hello from
21   Bah bah black
22   Now is
23   the
24   sheep
25   the
26   other side of
27   time for all
28   *<blank line>*
29   good men and
30   the universe
31
32
33   ## Submitting your solution

34   Send all three source code files to the Assignment Checker with the subject line **C2A8E1_ID**, where *ID* is
35   your 9-character UCSD student ID.

36   *See the course document titled "Preparing and Submitting Your Assignments" for additional exercise*
37   *formatting, submission, and Assignment Checker requirements.*
38
39
40   **Hints:**
41   C++ does not allow copies to be made of **ifstream** objects. Dynamically create an array of **ifstream**
42   objects with one element for each file to open. These objects may also be used in conjunction with the
43   **is_open** function to determine if a file is still open.

Page 2 of 2 of C2A8E1

1 **C2A8E2** *(10 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 **C2A8E2_OpenFileBinary.c** and **C2A8E2_DisplayModifiedSingleReals.c**. Also add
4 instructor-supplied source code file **C2A8E2_main-Driver.c**. <u>Do not write a `main` function!</u> `main` already
5 exists in the instructor-supplied file and it will use the code you write.

6

7 A budding young computer scientist working at the local mini-mart has developed a slightly modified
8 version of the IEEE 754 "Single Real" floating point format described in notes 16.10A–16.10C of the course
9 book. The only difference between his version and the original is that the "Exponent" field occupies 9
10 bits and the "Fraction" field occupies 22 bits. These changes (shown in **bold**) are reflected in the
11 following modified tables:

Modified Single Real (32 Bits)

30	21	0
9-Bit Exponent	**22**-Bit Fraction	

Sign of Fraction
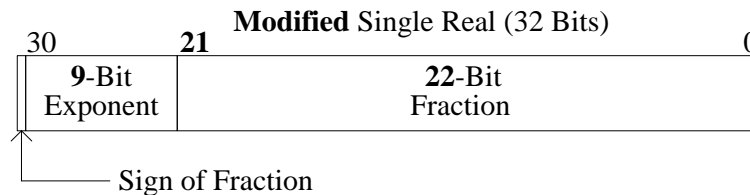
Modified Single Real				
Normalized Numbers (~limits: $\pm$**1.5*10^{77}**, $\pm$**3.4*10^{-77}**)			Zeros	
Bias of e	$+$**255**		Range of e	0
Range of e	0 < e < **511**		Mantissa	0.f = 0.0
Range of f	Any value			
Mantissa	1.f		Infinities	
Value	$(-1)^s$ * 1.f * $2^{(e-255)}$		Range of e	**511**
Denormalized Numbers (~limits: $\pm$**8.2*10^{-84}**)			Mantissa	0.f = 0.0
Bias of e	$+$**254**		NANs	
Range of e	0		Range of e	**511**
Range of f	Non-0		Range of f	Non-0
Mantissa	0.f			
Value	$(-1)^s$ * 0.f * $2^{(e-254)}$			

38 In order to test this modified format two instructor-supplied data files have been supplied, each of which
39 must be placed in the program's "working directory". Each contains an assortment of 32-bit patterns,
40 where these patterns represent various combinations of normalized numbers, denormalized numbers,
41 zeros, infinities, and not-a-numbers (NANs).

42

43

44 File **C2A8E2_OpenFileBinary.c** must contain a function named `OpenFileBinary`.
45 `OpenFileBinary` syntax:
46   `FILE *OpenFileBinary(const char *fileName);`
47 Parameter:
48   `fileName` – a pointer to the name of the file to be opened
49 Synopsis:
50   Opens the file named in `fileName` in the read-only binary mode. If the open fails an error message
51   is output to `stderr` and the program is terminated with an error exit code. The error message must
52   mention the name of the failing file.

1 Return:
2   A pointer to the **FILE** structure for the open file if the open succeeds; otherwise, the function does
3   not return.
4
5 File **C2A8E2_DisplayModifiedSingleReals.c** must contain function **DisplayModifiedSingleReals**.
6 **DisplayModifiedSingleReals** syntax:
7   **void DisplayModifiedSingleReals(FILE *inFile);**
8 Parameter:
9   **inFile** – a pointer to a **FILE** structure representing an open readable binary file
10 Synopsis:
11   This function assumes a byte is 8 bits and that the file in **inFile**:
12   1. was written in "<u>big endian</u>" format;
13   2. contains successive 32-bit patterns, each representing a "Modified Single Real".
14   This function displays an aligned table in which each 32-bit pattern is represented by an 8-character
15   hexadecimal value (letters may be uppercase or lowercase) followed by what that value represents
16   if interpreted as a "Modified Single Real". That representation will <u>always be preceded by a plus</u>
17   <u>sign or a minus sign</u> as appropriate. The possible representations are:
18   1. If a Normalized Number, a Denormalized Number, or a Zero is represented its magnitude will
19   be displayed in scientific notation (using **printf**'s **%e** conversion specification) followed by
20   the word **Normal**, **Denormal**, or **Zero**, as appropriate;
21   2. If an infinity is represented **INF** will be displayed;
22   3. If a not-a-number is represented **NAN** will be displayed.
23   If the file ends with an incomplete pattern (1, 2, or 3 bytes) the exact message **Unexpected EOF** will
24   be displayed at that point instead of the incomplete pattern.
25 Return: **void**
26
27 1. Do <u>not</u> attempt to obtain a count of the total number of bytes in the file.
28 2. Do <u>not</u> attempt to read the entire contents of the file at once.
29 3. Do <u>not</u> make any assumptions about or attempt to determine the machine's "endianness"; a
30   properly written program does not need to know.
31 4. Do <u>not</u> make any assumptions about the maximum number of bytes in any data types other than the
32   **char** types. For example, you may not assume type **long** has only 4 bytes.
33 5. Although you may assume a byte is 8 bits, if you actually need to represent that number use the
34   standard library **CHAR_BIT** macro, not something you define.
35
36 Manually run your program twice – once with instructor-supplied input file **TestFile7.bin** and once with
37 instructor-supplied input file **TestFile8.bin**, each of which must be placed in the program's "working
38 directory". Specify the desired file name on the command line – DO NOT prompt the user for it or place
39 it in your code.
40
41 Your display must be in the format shown below, where the number of exponent digits actually
42 displayed (typically 2 or 3) will vary by implementation. This example represents the first 12 patterns from
43 file **TestFile7.bin**. The EOF message at the end of the display must occur if and only if a file ends with an
44 incomplete pattern:
45
46   0xffc00001 -NAN
47   0x7fffffff +NAN
48   0xffc00000 -INF
49   0x7fc00000 +INF
50   0xffbfffff -1.157921e+77   Normal
51   0x7fbfffff +1.157921e+77   Normal
52   0x80400000 -3.454467e-77   Normal
53   0x00400000 +3.454467e-77   Normal
54   0x80000001 -8.236092e-84   Denormal

```
0x003fffff  +3.454467e-77  Denormal
0x000d0a00  +7.037971e-78  Denormal
0x80000000  -0.000000e+00  Zero
…
Unexpected EOF
```

Submitting your solution

Send all three source code files to the Assignment Checker with the subject line **C2A8E2_ID**, where *ID* is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

Hints:

Read each 4-byte group from the input file into a 4-element **unsigned char** array and use a 4-iteration loop to place those 4 bytes into the appropriate bytes of a single **unsigned long** variable. Do all necessary masking and testing on that variable.