# Assignment 8 C/C++ Programming II

**C2A8** General Information

# Assignment 8 consists of TWO (2) exercises:

C2A8E1 C2A8E2

All requirements are in this document.

1	C2A8 General Information, continued
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14 15	
15	
16 17	
18	
19	
20	
20	No. Company la forma article for Their Appingues and
21	No General Information for This Assignment
22	
23	
22 23 24 25 26 27 28 29	
25	
26	
2/	
28 20	
∠7 3∩	
30 31	
31 32 33 34	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42 43	
43 4.4	Cot a Consolidated Assignment & Report (entional)
44 45	Get a Consolidated Assignment 8 Report (optional)
45 44	If you would like to receive a consolidated report containing the results of the most recent version or
46 47	each exercise submitted for this assignment:
47 40	`Send an empty-body email to the assignment checker with the subject line C2A8_164440_U09339367
48 40	and no attachments.
49 50	Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report
50	requests as many times as you wish before the assignment deadline.

# C2A8E1 (10 points – C++ Program)

Exclude any existing source code files that may already be in your IDE project and add a two new ones, naming them C2A8E1\_OpenFiles.cpp and C2A8E1\_MergeAndDisplay.cpp. Also add instructor-supplied source code file C2A8E1\_main-Driver.cpp. Do not write a main function! main already exists in the instructor-supplied file and it will use the code you write.

5 6 7

8

9

11 12

13 14

15

16

17 18

19

20

21

22

23

1

3

4

File **C2A8E1\_OpenFiles.cpp** must contain a function named **OpenFiles**.

OpenFiles syntax:

```
ifstream *OpenFiles(char * const fileNames[], size_t count);
```

10 Parameters:

**fileNames** – a pointer to the first element in an array representing the names of text files to be opened. The array has the following standard ragged array format:

```
char *fileNames[] = { "fileA", "fileB", etc. };
```

count – the number of elements in fileNames

Synopsis:

Dynamically creates an array of **ifstream** objects having **count** elements then uses those objects to open the files named in **fileNames**, in order. All opens are in the read-only text mode. If any open fails all previously opened files are explicitly closed, the dynamic allocation is deleted, an error message is output to **cerr**, and the program is terminated with an error exit code. The error message must mention the name of the failing file. If **count** is zero an error message to that effect is output to **cerr** and the program is terminated with an error exit code.

Return:

a pointer to the first entry in the **ifstream** array if **count** is non-zero and all opens succeed; otherwise, the function does not return.

24252627

28

29

30

31

32

33

34 35

36 37

38

39

40

41

43

44 45

46

47 48 File C2A8E1\_MergeAndDisplay.cpp must contain a function named MergeAndDisplay.

MergeAndDisplay syntax:

```
void MergeAndDisplay(ifstream files[], size t count);
```

Parameters:

files – a pointer to the first element in an array of ifstream objects having count elements, where each object represents a text file open in the text mode for reading.

count – the number of elements in the array in files

Synopsis:

Proceeding in order from the first file specified in files, the first line in each file is read and displayed, followed by the second line in each, followed by the third, etc. When the end of any file is reached that file is closed and the process continues using only the remaining open files until all files have finally been closed. Empty lines are displayed as empty lines. Empty files are simply closed and ignored.

Return:

void

- 42
  - Do not attempt to use data you "tried" to read from a file before verifying the read was successful.
     Functions OpenFiles and MergeAndDisplay must be able to handle any number of files specified
  - by their count parameter.
  - 3. You may assume that no line in a file will contain more than 511 characters.
  - 4. Do <u>not</u> display anything other than the exact contents of the files, i.e., no file names, no line numbers, no extra spaces, no extra blank lines, etc.
- 49 5. Do not attempt to store the entire contents of any file at once.
- 6. Do <u>not</u> attempt to read a file again after reaching its end.
- 7. Do <u>not</u> attempt to determine the number or lines or bytes in any file.
- 8. Do <u>not</u> use the peek, seekg, seekp, tellg, or tellp functions.

Manually run your program twice, specifying the names of the instructor-supplied data files listed below on the command line in the order shown. Each must be placed in the program's "working directory". DO NOT prompt the user for the file names or place them in your program code.

f2

3 4 5

6

7 8

9

10 11

12

13

14

15

16

1 2

> Command line file names for test 1: Command line file names for test 2:

mFile1.txt mFile2.txt mFile3.txt mFile4.txt mFile5.txt

mFile3.txt mFile2.txt mFile1.txt

### Example:

If the command line specifies files f1 f2 f3 and those files contain the following text:

	f1
Line 1:	Hello from
Line 2:	the
Line 3:	other side of
Line 4:	<blank line=""></blank>

Bah bah black Now is sheep the <EOF> time for all good men and

<E0F> Line 5: the universe<EOF>

17 18

the display would be as follows, where *<blank line>* represents an actual blank line:

19

22

23

24

25

26

27 28

29

30

20 Hello from 21 Bah bah black

> Now is the sheep

the other side of time for all <blank line> good men and the universe

31 32 33

34

35

36

37

## **Submitting your solution**

`Send an empty-body email to the assignment checker with the subject line C2A8E1\_164440\_U09339367 and with all three source code files attached.

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

38 39 40

41

42 43

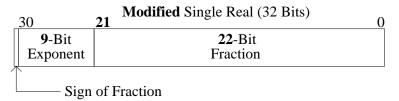
#### Hints:

C++ does not allow copies to be made of ifstream objects. Dynamically create an array of ifstream objects with one element for each file to open. These objects may also be used in conjunction with the is\_open function to determine if a file is still open.

# C2A8E2 (10 points – C Program)

Exclude any existing source code files that may already be in your IDE project and add two new ones, naming them **C2A8E2\_OpenFileBinary.c** and **C2A8E2\_DisplayModifiedSingleReals.c**. Also add instructor-supplied source code file **C2A8E2\_main-Driver.c**. Do not write a main function! main already exists in the instructor-supplied file and it will use the code you write.

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



Modified Single Real							
Normalized Numbers							
(~limits: ±1.5*10 <sup>77</sup> , ±3.4*10 <sup>-77</sup> )			Zeros				
Bias of e	+255		Range of e	0			
Range of e	0 < e < <b>511</b>		Mantissa	0.f = 0.0			
Range of f	Any value						
Mantissa	1.f		Infinities				
Value	(-1)s * 1.f * 2(e- <b>255</b> )		Range of e	511			
Denormalized Numbers			Mantissa	0.f = 0.0			
(~limit	s: ± <b>8.2*10</b> -84)						
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 * O.f * 2(e- <b>254</b> )						

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

File **C2A8E2\_OpenFileBinary.c** must contain a function named **OpenFileBinary**. **OpenFileBinary** syntax:

FILE \*OpenFileBinary(const char \*fileName);
Parameter:

**fileName** – a pointer to the name of the file to be opened Synopsis:

Opens the file named in **fileName** in the read-only binary mode. If the open fails an error message is output to **stderr** and the program is terminated with an error exit code. The error message must mention the name of the failing file.

Return:

A pointer to the FILE structure for the open file if the open succeeds; otherwise, the function does not return.

3 4 5

> 6 7

8

9

10

11

12

13

14

15

16

17

18 19

20 21

22

23

24

25

26 27

28

29

30 31

32

33

34 35

36

37 38

39 40

1

2

File C2A8E2\_DisplayModifiedSingleReals.c must contain function DisplayModifiedSingleReals.

DisplayModifiedSingleReals syntax:

void DisplayModifiedSingleReals(FILE \*inFile);

Parameter:

inFile – a pointer to a FILE structure representing an open readable binary file Synopsis:

This function assumes a byte is 8 bits and that the file in inFile:

- 1. was written in "big endian" format.
- 2. contains successive 32-bit patterns, each representing a "Modified Single Real".

This function displays an aligned table in which each 32-bit pattern is represented by an 8-character hexadecimal value (letters may be uppercase or lowercase) followed by what that value represents if interpreted as a "Modified Single Real". That representation will <u>always be preceded by a plus sign or a minus sign as appropriate</u>. The possible representations are:

- 1. If a Normalized Number, a Denormalized Number, or a Zero is represented its magnitude will be displayed in scientific notation (using printf's %e conversion specification) followed by the word **Normal**, **Denormal**, or **Zero**, as appropriate.
- 2. If an infinity is represented **INF** will be displayed.
- 3. If a not-a-number is represented **NAN** will be displayed.

If the file ends with an incomplete pattern (1, 2, or 3 bytes) the exact message **Unexpected EOF** will be displayed at that point instead of the incomplete pattern.

Return: void

- 1. Do <u>not</u> attempt to obtain a count of the total number of bytes in the file.
- 2. Do not attempt to read the entire contents of the file at once.
- 3. Do <u>not</u> make any assumptions about or attempt to determine the machine's "endianness"; a properly written program does not need to know.
- 4. Do <u>not</u> make any assumptions about the maximum number of bytes in any data types other than the **char** types. For example, you may not assume type **long** has only 4 bytes.
- 5. If you use a structure or a class (neither of which is necessary), do <u>not</u> make any assumptions about padding or whether bit fields are allocated left-to-right or right-to-left.
- 6. Although you may assume a byte is 8 bits, if you actually need to represent that number use the standard library CHAR BIT macro, not something you define.

Manually run your program twice – once with instructor-supplied input file **TestFile7.bin** and once with instructor-supplied input file **TestFile8.bin**, each of which must be placed in the program's "working directory". Specify the desired file name on the command line – DO NOT prompt the user for it or place it in your code.

41 42 43

44

45

Your display must be in the format shown below, where the number of exponent digits actually displayed (typically 2 or 3) will vary by implementation. This example represents the first 12 patterns from file **TestFile7.bin**. The EOF message at the end of the display must occur if and only if a file ends with an incomplete pattern:

46 47

```
48
           0xffc00001
                       -NAN
           0x7fffffff
49
                       +NAN
           0xffc00000
50
                       -INF
           0x7fc00000
51
                       +INF
52
           0xffbfffff
                      -1.157921e+77 Normal
53
           0x7fbfffff +1.157921e+77
                                      Normal
54
           0x80400000 -3.454467e-77
                                      Normal
```

#### Personalized C2A8 requirements exclusively for Phillip Ward (U09339367) C/C++ Programming II (Section 164440)

```
Normal
1
          0x00400000
                     +3.454467e-77
2
         0x80000001
                     -8.236092e-84
                                     Denormal
3
         0x003fffff
                     +3.454467e-77
                                     Denormal
4
          0x000d0a00
                     +7.037971e-78
                                    Denormal
5
         0x80000000
                     -0.000000e+00 Zero
6
7
         Unexpected EOF
8
```

9

11

12

13

# **Submitting your solution**

`Send an empty-body email to the assignment checker with the subject line **C2A8E2\_164440\_U09339367** and with all three source code files <u>attached</u>.

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

14 15 16

17

18

19 20

#### Hints:

Read each 4-byte group from the input file into a 4-element **unsigned char** array and use a 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.