Consolidated Assignment 7 Report

This report contains the graded results for the newest of each exercise submitted to the assignment checker prior to 8/27/2020 4:08:00 AM PDT.

Student Name: Shaun Chemplavil
Student ID: U08713628
Contact e-mail: shaun.chemplavil@gmail.com
C/C++ Programming II (Section 149123)

```
Submitted:
   Exercise 1: 8/18/2020 7:32:59 PM PDT
   Exercise 2: 8/19/2020 10:02:05 AM PDT
   Exercise 3: 8/19/2020 1:36:50 PM PDT
   Exercise 4: 8/22/2020 12:05:16 PM PDT
```

Score (out of 20 possible): 20

```
THIS WAS SENT FROM A NOTIFICATION-ONLY ADDRESS THAT CANNOT ACCEPT INCOMING MAIL.
For help please contact the instructor at the email address provided on the "Home" page
of the course's Canvas website.  The assignment checker DOES NOT GRADE your submissions
but merely reports on issues so you can correct them and resubmit, thereby avoiding
unnecessary credit loss.  ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the
assignment deadline based solely upon the NEWEST submission of each exercise.  BE WARY
of correcting minor issues after the deadline because a late deduction will usually be
much greater than a minor issue deduction.


   From: Shaun Chemplavil <mailto:shaun.chemplavil@gmail.com>
   Subject: C2A7E1_U08713628
   Submitted: 8/18/2020 7:32:59 PM PDT
   Course: C/C++ Programming II (Section 149123)
   Student's name: Shaun Chemplavil
   Contact email: shaun.chemplavil@gmail.com
   Student ID: U08713628
   Assignment 7, Exercise 1  (C2_001205940M02005X75205)
   Exercise point value: 6
   File submitted:
      C2A7E1_main.c

"Compile-time" results:
   No "compile-time" issues;
"Run-time" results:
   Program ran - No errors detected during preliminary testing (SEE ATTACHMENT);
```

1 80

```c
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C / C++ Programming II : Dynamic Memory and File I / O Concepts
// 149123 Raymond L.Mitchell, Jr., M.S.
// 08 / 18 / 2020
// C2A7E1_main.c
// Win10
// Visual C++ 19.0
//
//This program will run at the command line with two space separated arguments
// first argument specifies a text file name, and the second specifies the
// how many bins should be used within a HashTable that contains strings within
// the file (bin # = string length % # of bins), within each bin a binary tree
// is used to organize the nodes so that they are ordered in alphabetical order
// according to the string it contains
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LINE_LEN 256                    // size of input buffer
#define BUFFMT "%255"                   // field width for input buffer scan

void *SafeMalloc(size_t size)
{
   void *vp;

   if ((vp = malloc(size)) == NULL)
   {
      fputs("Out of memory\n", stderr);
      exit(EXIT_FAILURE);
   }
   return(vp);
}

FILE *OpenFile(const char *fileName)
{
   FILE *fp;

   if ((fp = fopen(fileName, "r")) == NULL)
   {
      fprintf(stderr, "File \"%s\" didn't open.\n", fileName);
      perror(fileName);
      exit(EXIT_FAILURE);
   }
   return fp;
}

#define MIN_ARGS 3                         // fewest command line arguments
#define FILE_ARG_IX 1                      // index of file name argument
#define BINS_ARG_IX 2                      // index of bin count argument

typedef struct Node NODE;
struct Node                                // type of each list node
{
   char *strng;                            // string for this node
   size_t count;                           // occurrences of this string
   NODE  *left, *right;                             // next node in list
};
```

```

typedef struct                         // type of table array elements
{
   size_t nodes;                       // # of list nodes for this bin
   NODE *root;                         // root node in this bin's list
} BIN;

typedef struct                         // type of hash table descriptor
{
   size_t bins;                        // bins in hash table
   BIN *firstBin;                      // first bin
} TABLE;

//
// BuildTree will search the binary tree at pNode for a node representing the
// string in str.  If found, its string count will be incremented.  If not
// found, a new node for that string will be created, put in alphabetical
// order, and its count set to 1.  A pointer to the node for string str is
// returned.
//
NODE *BuildTree(NODE *pNode, char *str, BIN *bp)
{
   if (pNode == NULL)                                    // string not found
   {
      size_t length = strlen(str) + 1;                   // length of string

      pNode = (NODE *)SafeMalloc(sizeof(NODE));          // allocate a node
      pNode->strng = (char *)SafeMalloc(length);
      memcpy(pNode->strng, str, length);                 // copy string
      pNode->count = 1;                                  // 1st occurrence
      pNode->left = pNode->right = NULL;                 // no subtrees
      ++bp->nodes;                                       // update bin node cnt
   }
   else
   {
      int result = strcmp(str, pNode->strng);            // compare strings

      if (result == 0)                                   // new str == current
         ++pNode->count;                                 // increment occurrence
      else if (result < 0)                               // new str < current
         pNode->left = BuildTree(pNode->left, str, bp);  // traverse left
      else                                               // new str > current
         pNode->right = BuildTree(pNode->right, str, bp);// traverse right
   }
   return(pNode);
}

// PrintTree recursively prints the binary tree in pNode alphabetically.
void PrintTree(const NODE *pNode)
{
   if (pNode != NULL)                                 // if child exists
   {
      PrintTree(pNode->left);                         // traverse left
      printf("%4d  %s\n", (int)pNode->count, pNode->strng);
      PrintTree(pNode->right);                        // traverse right
   }
}

// FreeTree recursively frees the binary tree in pNode.
void FreeTree(NODE *pNode)
```

```
{
   if (pNode != NULL)                                  // if child exists
   {
      FreeTree(pNode->left);                           // traverse left
      FreeTree(pNode->right);                          // traverse right
      free(pNode->strng);                              // free the string
      free(pNode);                                     // free the node
   }
}

int HashFunction(const char *key, size_t bins)  // get bin value from key
{
   return((int)(strlen(key) % bins));           // value = char count % bins
}

// CreateTable creates and initializes the hash table and its bins
TABLE *CreateTable(size_t bins)
{
   TABLE *hashTable;
   BIN  *end;

   hashTable = (TABLE *)SafeMalloc(sizeof(TABLE));          // alloc desc struct
   hashTable->bins = bins;                                  // how many bins
   // alloc bins
   hashTable->firstBin = (BIN *)SafeMalloc(bins * sizeof(BIN));
   end = hashTable->firstBin + bins;                        // end of bins

   for (BIN *bin = hashTable->firstBin; bin < end; ++bin)   // initialize bins
   {
      bin->nodes = 0;                                       // no list nodes
      bin->root = NULL;                                     // no list
   }
   return(hashTable);
}

//
// BuildList calls BuildTree, which ultimately orders all nodes within the bin
// that is passed to it, using the string that is also passed to it
//
void BuildList(BIN *bp, char *str)
{
   bp->root = BuildTree(bp->root, str, bp);
}

// PrintTable prints the hash table.
void PrintTable(const TABLE *hashTable)
{
   BIN  *end;

   end = hashTable->firstBin + hashTable->bins;             // end of bins
   for (BIN *bin = hashTable->firstBin; bin < end; ++bin)   // visit bins
   {
      printf("%d entries for bin %d:\n",
         (int)bin->nodes,
         (int)(bin - hashTable->firstBin));

      // Print all nodes in bin
      PrintTree(bin->root);
   }
}
```

```c

// FreeTable frees the hash table.
void FreeTable(TABLE *hashTable)
{
   BIN *bin, *end;

   end = hashTable->firstBin + hashTable->bins;       // end of bins
   for (bin = hashTable->firstBin; bin < end; ++bin)  // visit bins
      FreeTree(bin->root);                            // free all nodes in bin

   free(hashTable->firstBin);                         // free all bins
   free(hashTable);                                   // free table descriptor
}

int main(int argc, char *argv[])
{
   char buf[LINE_LEN];                             // word string buffer
   char fileName[LINE_LEN];                        // file name buffer
   int howManyBins;                                // number of bins to create
   TABLE *hashTable;                               // pointer to hash table
   FILE *fp;

   // Read file name from command line.
   if (argc < MIN_ARGS || sscanf(argv[FILE_ARG_IX], BUFFMT "s", fileName) != 1)
   {
      fprintf(stderr, "No file name specified on command line\n");
      return EXIT_FAILURE;
   }
   fp = OpenFile(fileName);

   // Read bin count from command line.
   if (sscanf(argv[BINS_ARG_IX], "%d", &howManyBins) != 1)
   {
      fprintf(stderr, "No bin count specified on command line\n");
      return EXIT_FAILURE;
   }
   hashTable = CreateTable((size_t)howManyBins);   // allocate table array

   //
   // The following loop will read one string at a time from stdin until
   // EOF is reached.  For each string read the BuildList function will
   // be called to update the hash table.
   //
   while (fscanf(fp, BUFFMT "s", buf) != EOF)      // get string from file
   {
      // Find appropriate bin.
      BIN *bin = &hashTable->firstBin[HashFunction(buf, (size_t)howManyBins)];
      BuildList(bin, buf);                         // put string in list
   }
   fclose(fp);
   PrintTable(hashTable);                          // print all strings
   FreeTable(hashTable);                           // free the table
   return(EXIT_SUCCESS);
}
```

```
********** C2 ASSIGNMENT 7 EXERCISE 1 AUTOMATIC PROGRAM RUN RESULTS **********

************* THE RESULTS BELOW HAVE BEEN PARTIALLY CHECKED AND **************
*************   NO ERRORS WERE FOUND.  HOWEVER, THIS DOES NOT   **************
*************  NECESSARILY MEAN THAT THERE ARE NO ERRORS.  THE  **************
*************  INSTRUCTOR WILL DO A MORE THOROUGH CHECK DURING  **************
*************                  MANUAL GRADING.                  **************


----------------------------- PURPOSE OF 1ST RUN ----------------------------
Verify binary tree content display for 10 bins.
------------------------------ START OF 1ST RUN -----------------------------

The assignment checker has split your display in half for compactness...

6 entries for bin 0:                   9 entries for bin 5:
   1  arguments.                          1  Thus,
   1  constants.                          1  White
   1  expansion)                          1  first
   1  invocation                          4  macro
   1  occurrence                          1  marks
   1  parameters                          3  space
6 entries for bin 1:                      2  token
   6  a                                   1  which
   1  combination                         4  white
   1  definition,                      11 entries for bin 6:
   1  definition.                         1  (after
   1  number-sign                         5  actual
   1  stringizing                         2  formal
12 entries for bin 2:                     1  macros
   1  If                                  1  occurs
   1  It                                  1  passed
   1  as                                  2  single
   2  by                                  2  space.
   1  concatenated                        6  string
   1  if                                  2  tokens
   5  in                                  1  within
   7  is                               5 entries for bin 7:
   3  it                                  2  between
   4  of                                  1  comment
   1  or                                  2  literal
   3  to                                  2  reduced
9 entries for bin 3:                      1  treated
   1  "stringizing"                    10 entries for bin 8:
   1  (#)                                 1  adjacent
   1  Any                                 4  argument
   3  The                                 1  converts
   3  and                                 1  enclosed
   1  any                                 1  ignored.
   1  automatically                       2  literal.
  13  the                                 1  literals
   1  two                                 2  operator
9 entries for bin 4:                      1  precedes
   1  each                                1  replaces
   1  from                             7 entries for bin 9:
   1  last                                1  argument,
   2  only                                1  following
   1  take                                2  parameter
   1  that                                1  preceding
   1  then                                1  quotation
   1  used                                 2  resulting
   2  with                                 1  separated
------------------------------- END OF 1ST RUN -------------------------------

----------------------------- PURPOSE OF 2ND RUN -----------------------------
Verify binary tree content display for 5 bins.
------------------------------ START OF 2ND RUN ------------------------------

The assignment checker has split your display in half for compactness...

15 entries for bin 0:                      3  it
   1  Thus,                                2  literal
   1  White                                4  of
   1  arguments.                           1  or
   1  constants.                           2  reduced
   1  expansion)                           3  to
   1  first                                1  treated
   1  invocation                        19 entries for bin 3:
   4  macro                                1  "stringizing"
   1  marks                                1  (#)
   1  occurrence                           1  Any
   1  parameters                           3  The
   3  space                                1  adjacent
   2  token                                3  and
   1  which                                1  any
   4  white                                4  argument
17 entries for bin 1:                      1  automatically
   1  (after                               1  converts
   6  a                                    1  enclosed
   5  actual                               1  ignored.
   1  combination                          2  literal.
   1  definition,                          1  literals
   1  definition.                          2  operator
   2  formal                               1  precedes
   1  macros                               1  replaces
   1  number-sign                         13  the
   1  occurs                               1  two
   1  passed                            16 entries for bin 4:
   2  single                               1  argument,
   2  space.                               1  each
   6  string                               1  following
   1  stringizing                          1  from
   2  tokens                               1  last
   1  within                               2  only
17 entries for bin 2:                      2  parameter
   1  If                                   1  preceding
   1  It                                   1  quotation
   1  as                                   2  resulting
   2  between                              1  separated
   2  by                                   1  take
   1  comment                              1  that
   1  concatenated                         1  then
   1  if                                   1  used
   5  in                                   2  with
   7  is
------------------------------- END OF 2ND RUN -------------------------------

----------------------------- PURPOSE OF 3RD RUN -----------------------------
Verify binary tree content display for 1 bin.
------------------------------ START OF 3RD RUN ------------------------------
```

```
The assignment checker has split your display in half for compactness...

84 entries for bin 0:                      2  literal
   1  "stringizing"                        2  literal.
   1  (#)                                  1  literals
   1  (after                               4  macro
   1  Any                                  1  macros
   1  If                                   1  marks
   1  It                                   1  number-sign
   3  The                                  1  occurrence
   1  Thus,                                1  occurs
   1  White                                4  of
   6  a                                    2  only
   5  actual                               2  operator
   1  adjacent                             1  or
   3  and                                  2  parameter
   1  any                                  1  parameters
   4  argument                             1  passed
   1  argument,                            1  precedes
   1  arguments.                           1  preceding
   1  as                                   1  quotation
   1  automatically                        2  reduced
   2  between                              1  replaces
   2  by                                   2  resulting
   1  combination                          1  separated
   1  comment                              2  single
   1  concatenated                         3  space
   1  constants.                           2  space.
   1  converts                             6  string
   1  definition,                          1  stringizing
   1  definition.                          1  take
   1  each                                 1  that
   1  enclosed                            13  the
   1  expansion)                           1  then
   1  first                                3  to
   1  following                            2  token
   2  formal                               2  tokens
   1  from                                 1  treated
   1  if                                   1  two
   1  ignored.                             1  used
   5  in                                   1  which
   1  invocation                           4  white
   7  is                                   2  with
   3  it                                   1  within
   1  last
------------------------------- END OF 3RD RUN -------------------------------

----------------------------- PURPOSE OF 4TH RUN -----------------------------
Verify that program detects an input file open failure.
------------------------------ START OF 4TH RUN ------------------------------

File "bad//file//b" didn't open.
bad//file//b:No such file or directory



------------------------------- END OF 4TH RUN -------------------------------

----------------------------- PURPOSE OF 5TH RUN -----------------------------
```

```
Verify that program detects an input file open failure.
------------------------------ START OF 5TH RUN ------------------------------

File "bad//file//a" didn't open.
bad//file//a:No such file or directory



------------------------------- END OF 5TH RUN -------------------------------

----------------------------- PURPOSE OF 6TH RUN -----------------------------
Verify that program detects a memory allocation failure.
-------------------------- CODE CHANGES FOR 6TH RUN --------------------------
Intentionally induced malloc failure.
------------------------------ START OF 6TH RUN ------------------------------

Out of memory



------------------------------- END OF 6TH RUN -------------------------------
```

```
THIS WAS SENT FROM A NOTIFICATION-ONLY ADDRESS THAT CANNOT ACCEPT INCOMING MAIL.
For help please contact the instructor at the email address provided on the "Home" page
of the course's Canvas website.  The assignment checker DOES NOT GRADE your submissions
but merely reports on issues so you can correct them and resubmit, thereby avoiding
unnecessary credit loss.  ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the
assignment deadline based solely upon the NEWEST submission of each exercise.  BE WARY
of correcting minor issues after the deadline because a late deduction will usually be
much greater than a minor issue deduction.


   From: Shaun Chemplavil <mailto:shaun.chemplavil@gmail.com>
   Subject: C2A7E2_U08713628
   Submitted: 8/19/2020 10:02:05 AM PDT
   Course: C/C++ Programming II (Section 149123)
   Student's name: Shaun Chemplavil
   Contact email: shaun.chemplavil@gmail.com
   Student ID: U08713628
   Assignment 7, Exercise 2  (C2_001351768M02005X97351)
   Exercise point value: 4
   Files submitted:
      C2A7E2_main-Driver.cpp
      C2A7E2_OpenFileBinary.cpp
      C2A7E2_ListHex.cpp

"Compile-time" results:
   No "compile-time" issues;
"Run-time" results:
   Program ran - No errors detected during preliminary testing (SEE ATTACHMENT);
```

```
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C / C++ Programming II : Dynamic Memory and File I / O Concepts
// 149123 Raymond L.Mitchell, Jr., M.S.
// 08 / 19 / 2020
// C2A7E2_ListHex.cpp
// Win10
// Visual C++ 19.0
//
// File contains ListHex function, which display the characters of a file
// as hexadecimal pairs, a new line is created once the number of bytesPerLine
// has been met
//

#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

void ListHex(ifstream &inFile, int bytesPerLine)
{
   int hexBytes = 0;

   // Set persistent I/O manipulators
   cout << setfill('0') << hex;

   do
   {
      char buf;
      inFile.get(buf);

      if (inFile.gcount())
      {
         // If not the first character of the line, place a preceding space
         // this avoids having a trailing whitespace character
         if (++hexBytes % bytesPerLine != 1)
            cout << ' ';

         // protect against sign extension bytes by typecasting to unsigned char
         // to display hexadecimal additional typecast to int
         cout << setw(2) << (int)(unsigned char)buf;

         // keep track of the number of characters we've displayed
         if (!(hexBytes % bytesPerLine))
            cout << '\n';
      }
   } while (inFile.good());
}
```

```
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C / C++ Programming II : Dynamic Memory and File I / O Concepts
// 149123 Raymond L.Mitchell, Jr., M.S.
// 08 / 18 / 2020
// C2A7E2_OpenFileBinary.cpp
// Win10
// Visual C++ 19.0
//
// File contains OpenFileBinary function, which opens the file (specified via a
// pointer to the name of filename) in read-only binary mode , using
// the ifstream reference which is also passed to it
//

#include <iostream>
#include <fstream>

using namespace std;

void OpenFileBinary(const char *fileName, ifstream &inFile)
{
   // open fileName in "read" mode using the ifstream object inFile
   inFile.open(fileName, ios_base::binary);
   if (!inFile.is_open())
   {
      {
         cerr << "\"" << fileName << "\" :File access error!\n";
         exit(-1);
      }
   }
}
```

```
********** C2 ASSIGNMENT 7 EXERCISE 2 AUTOMATIC PROGRAM RUN RESULTS **********

************* THE RESULTS BELOW HAVE BEEN PARTIALLY CHECKED AND **************
*************   NO ERRORS WERE FOUND.  HOWEVER, THIS DOES NOT   **************
*************  NECESSARILY MEAN THAT THERE ARE NO ERRORS.  THE  **************
*************  INSTRUCTOR WILL DO A MORE THOROUGH CHECK DURING  **************
*************                  MANUAL GRADING.                  **************


----------------------------- PURPOSE OF 1ST RUN -----------------------------
Verify the display of hexadecimal file bytes.
--------------------- COMMAND LINE ARGUMENTS FOR 1ST RUN ---------------------
TestFile3.txt  16
------------------------------ START OF 1ST RUN ------------------------------

Hex dump of file TestFile3.txt with 16 bytes per line:
66 6f 72 20 28 69 20 3d 20 30 3b 20 69 20 3c 20
52 45 43 5f 4e 4f 3b 20 2b 2b 69 29 09 2f 2a 20
66 6f 72 20 66 69 72 73 74 20 75 6e 6e 65 65 64
65 64 20 72 65 63 6f 72 64 73 20 2a 2f 0d 0a 09
69 66 20 28 66 73 63 3f 61 6e 66 28 66 70 2c 20
22 25 2a 5b 5e 5c 6e 5d 25 2a 63 22 29 20 3d 3d
20 45 4f 46 29 20 7b 20 2f 2a 20 72 65 61 64 20
61 6e 64 20 74 68 72 6f 77 20 61 77 61 79 20 2a
2f 0d 0a 09 20 20 66 70 75 74 73 28 22 55 6e 65
78 70 65 63 74 65 64 20 45 4f 46 5c 6e 22 2c 20
73 74 64 65 72 72 29 3b 09 2f 2a 20 74 68 65 72
65 20 69 73 20 6e 6f 20 72 65 63 2e 6f 72 64 20
52 45 43 5f 4e 4f 20 2a 2f 0d 0a 20 20 7d 0d 0a
0d 0a

------------------------------- END OF 1ST RUN -------------------------------

----------------------------- PURPOSE OF 2ND RUN -----------------------------
Verify the display of hexadecimal file bytes.
--------------------- COMMAND LINE ARGUMENTS FOR 2ND RUN ---------------------
TestFile4.bin  16
------------------------------ START OF 2ND RUN ------------------------------

Hex dump of file TestFile4.bin with 16 bytes per line:
ff c0 00 01 7f ff ff ff ff c0 00 00 7f c0 00 00
ff bf ff ff 7f bf ff ff 80 40 00 00 00 40 00 00
80 00 00 01 00 3f ff ff 00 0d 0a 00 80 00 00 00
00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0 00 00 00 0e 00 00 00 00 0e ba 1f 21 cd 09 b4
44 20 6e 69 6d 20 53 4f 2e 65 64 6f b4 0a 0d 0d
00 00 0d 0a ba 03 7d 00 54 62 39 58 54 62 39 0b
ff da bc d8 7f e0 2a c6 ff c0 00 00 7f c0 00 00
54 62 38 0b 63 69 52 0b 54 62 39 68 00 00 00 0b
50 00 00 00 01 4c 00 45 0e 7f 00 05 00 00 3a 54
80 00 00 00 00 e0 00 00 ff da 21 77 80 00 00 06
5f 74 73 e5 64 70 2e 63 0d 0a 00 0e ba 1f ff c0
00 01 7f ff ff ff 01 4c 00 45 7f c0 00 00 ff bf
ff ff 80 40 00 00 00 40 00 00 80 00 00 01 00 3f
ff ff 00 00 00 0b 7f c0 00 00 00 00 00 00 80 00
00 00 7f bf ff ff 00 40 00 00 00 00 00 00 7f e0
2a c6 00 00 00 00 00 00 00 00 d0 00 00 00 0e 00
00 00 64 70 2e 63 21 cd 09 b4 44 20 6e 69 6d 20
53 4f b4 0a 0d 0a 00 00 00 00 00 00 3a 54 ba 03
7d 00 54 62 39 58 54 62 39 0b ff da bc d8 ff c0
```

```
00 00 54 62 38 0b 63 69 52 0b 54 62 39 68 50 00
00 00 2e 65 64 6f 0e 7f 00 05 80 00 00 00 00 e0
00 00 ff da 21 77 80 00 00 06 5f 74 73 e5 ff c0
00 00

------------------------------- END OF 2ND RUN -------------------------------

----------------------------- PURPOSE OF 3RD RUN -----------------------------
Verify the display of hexadecimal file bytes.
--------------------- COMMAND LINE ARGUMENTS FOR 3RD RUN ---------------------
TestFile1.txt  25
------------------------------ START OF 3RD RUN ------------------------------

Hex dump of file TestFile1.txt with 25 bytes per line:
54 68 65 20 6e 75 6d 62 65 72 2d 73 69 67 6e 20 6f 72 20 22 73 74 72 69 6e
67 69 7a 69 6e 67 22 20 6f 70 65 72 61 74 6f 72 20 28 23 29 20 63 6f 6e 76
65 72 74 73 20 6d 61 63 72 6f 0d 0a 70 61 72 61 6d 65 74 65 72 73 20 28 61
66 74 65 72 20 65 78 70 61 6e 73 69 6f 6e 29 20 74 6f 20 73 74 72 69 6e 67
20 63 6f 6e 73 74 61 6e 74 73 2e 20 49 74 20 69 73 20 75 73 65 64 0d 0a 6f
6e 6c 79 20 77 69 74 68 20 6d 61 63 72 6f 73 20 74 68 61 74 20 74 61 6b 65
20 61 72 67 75 6d 65 6e 74 73 2e 20 49 66 20 69 74 20 70 72 65 63 65 64 65
73 20 61 20 66 6f 72 6d 61 6c 0d 0a 70 61 72 61 6d 65 74 65 72 20 69 6e 20
74 68 65 20 6d 61 63 72 6f 20 64 65 66 69 6e 69 74 69 6f 6e 2c 20 74 68 65
20 61 63 74 75 61 6c 20 61 72 67 75 6d 65 6e 74 20 70 61 73 73 65 64 0d 0a
62 79 20 74 68 65 20 6d 61 63 72 6f 20 69 6e 76 6f 63 61 74 69 6f 6e 20 69
73 20 65 6e 63 6c 6f 73 65 64 20 69 6e 20 71 75 6f 74 61 74 69 6f 6e 20 6d
61 72 6b 73 20 61 6e 64 0d 0a 74 72 65 61 74 65 64 20 61 73 20 61 20 73 74
72 69 6e 67 20 6c 69 74 65 72 61 6c 2e 20 54 68 65 20 73 74 72 69 6e 67 20
6c 69 74 65 72 61 6c 20 74 68 65 6e 20 72 65 70 6c 61 63 65 73 0d 0a 65 61
63 68 20 6f 63 63 75 72 72 65 6e 63 65 20 6f 66 20 61 20 63 6f 6d 62 69 6e
61 74 69 6f 6e 20 6f 66 20 74 68 65 20 73 74 72 69 6e 67 69 7a 69 6e 67 20
6f 70 65 72 61 74 6f 72 0d 0a 61 6e 64 20 66 6f 72 6d 61 6c 20 70 61 72 61
6d 65 74 65 72 20 77 69 74 68 69 6e 20 74 68 65 20 6d 61 63 72 6f 20 64 65
66 69 6e 69 74 69 6f 6e 2e 0d 0a 0d 0a 57 68 69 74 65 20 73 70 61 63 65 20
70 72 65 63 65 64 69 6e 67 20 74 68 65 20 66 69 72 73 74 20 74 6f 6b 65 6e
20 6f 66 20 74 68 65 20 61 63 74 75 61 6c 20 61 72 67 75 6d 65 6e 74 0d 0a
61 6e 64 20 66 6f 6c 6c 6f 77 69 6e 67 20 74 68 65 20 6c 61 73 74 20 74 6f
6b 65 6e 20 6f 66 20 74 68 65 20 61 63 74 75 61 6c 20 61 72 67 75 6d 65 6e
74 20 69 73 20 69 67 6e 6f 72 65 64 2e 0d 0a 41 6e 79 20 77 68 69 74 65 20
73 70 61 63 65 20 62 65 74 77 65 65 6e 20 74 68 65 20 74 6f 6b 65 6e 73 20
69 6e 20 74 68 65 20 61 63 74 75 61 6c 20 61 72 67 75 6d 65 6e 74 20 69 73
0d 0a 72 65 64 75 63 65 64 20 74 6f 20 61 20 73 69 6e 67 6c 65 20 77 68 69
74 65 20 73 70 61 63 65 20 69 6e 20 74 68 65 20 72 65 73 75 6c 74 69 6e 67
20 73 74 72 69 6e 67 20 6c 69 74 65 72 61 6c 2e 0d 0a 54 68 75 73 2c 20 69
66 20 61 20 63 6f 6d 6d 65 6e 74 20 6f 63 63 75 72 73 20 62 65 74 77 65 65
6e 20 74 77 6f 20 74 6f 6b 65 6e 73 20 69 6e 20 74 68 65 20 61 63 74 75 61
6c 0d 0a 61 72 67 75 6d 65 6e 74 2c 20 69 74 20 69 73 20 72 65 64 75 63 65
64 20 74 6f 20 61 20 73 69 6e 67 6c 65 20 77 68 69 74 65 20 73 70 61 63 65
2e 20 54 68 65 20 72 65 73 75 6c 74 69 6e 67 0d 0a 73 74 72 69 6e 67 20 6c
69 74 65 72 61 6c 20 69 73 20 61 75 74 6f 6d 61 74 69 63 61 6c 6c 79 20 63
6f 6e 63 61 74 65 6e 61 74 65 64 20 77 69 74 68 20 61 6e 79 20 61 64 6a 61
63 65 6e 74 0d 0a 73 74 72 69 6e 67 20 6c 69 74 65 72 61 6c 73 20 66 72 6f
6d 20 77 68 69 63 68 20 69 74 20 69 73 20 73 65 70 61 72 61 74 65 64 20 6f
6e 6c 79 20 62 79 20 77 68 69 74 65 20 73 70 61 63 65 2e 0d 0a 0d 0a

------------------------------- END OF 3RD RUN -------------------------------

----------------------------- PURPOSE OF 4TH RUN -----------------------------
Verify that program detects an input file open failure.
--------------------- COMMAND LINE ARGUMENTS FOR 4TH RUN ---------------------
```

```
bad//file//a  5
------------------------------ START OF 4TH RUN ------------------------------

"bad//file//a" :File access error!



------------------------------- END OF 4TH RUN -------------------------------

----------------------------- PURPOSE OF 5TH RUN -----------------------------
Verify that program detects an input file open failure.
--------------------- COMMAND LINE ARGUMENTS FOR 5TH RUN ---------------------
bad//file//b  5
------------------------------ START OF 5TH RUN ------------------------------

"bad//file//b" :File access error!



------------------------------- END OF 5TH RUN -------------------------------
```

```
THIS WAS SENT FROM A NOTIFICATION-ONLY ADDRESS THAT CANNOT ACCEPT INCOMING MAIL.
For help please contact the instructor at the email address provided on the "Home" page
of the course's Canvas website.  The assignment checker DOES NOT GRADE your submissions
but merely reports on issues so you can correct them and resubmit, thereby avoiding
unnecessary credit loss.  ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the
assignment deadline based solely upon the NEWEST submission of each exercise.  BE WARY
of correcting minor issues after the deadline because a late deduction will usually be
much greater than a minor issue deduction.


   From: Shaun Chemplavil <mailto:shaun.chemplavil@gmail.com>
   Subject: C2A7E3_U08713628
   Submitted: 8/19/2020 1:36:50 PM PDT
   Course: C/C++ Programming II (Section 149123)
   Student's name: Shaun Chemplavil
   Contact email: shaun.chemplavil@gmail.com
   Student ID: U08713628
   Assignment 7, Exercise 3  (C2_001703744M02005X16703)
   Exercise point value: 4
   Files submitted:
      C2A7E3_main-Driver.c
      C2A7E3_ReverseEndian.c

"Compile-time" results:
   No "compile-time" issues;
"Run-time" results:
   Program ran - No errors detected during preliminary testing (SEE ATTACHMENT);
```

```
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C / C++ Programming II : Dynamic Memory and File I / O Concepts
// 149123 Raymond L.Mitchell, Jr., M.S.
// 08 / 19 / 2020
// C2A7E3_ReverseEndian.c
// Win10
// Visual C++ 19.0
//
// File contains ReverseEndian function, which converts any scalar object
// pointed to it from big endian to little endian (or vice versa)
//

#include <stdlib.h>

void *ReverseEndian(void *ptr, size_t size)
{
   // Swap contents at each byte
   for (char *head = (char *)ptr, *tail = head + size - 1;
      tail > head; --tail, ++head)
   {
      // store contents at head pointer
      char temp = *head;
      *head = *tail;
      *tail = temp;
   }

   //Now contents at the memory addresses have been swapped, so return original
   // address (with new content)
   return(ptr);
}
```

```
********** C2 ASSIGNMENT 7 EXERCISE 3 AUTOMATIC PROGRAM RUN RESULTS **********

************* THE RESULTS BELOW HAVE BEEN PARTIALLY CHECKED AND **************
*************   NO ERRORS WERE FOUND.  HOWEVER, THIS DOES NOT   **************
*************  NECESSARILY MEAN THAT THERE ARE NO ERRORS.  THE  **************
*************  INSTRUCTOR WILL DO A MORE THOROUGH CHECK DURING  **************
*************                  MANUAL GRADING.                  **************


--------------------------------- START OF RUN --------------------------------

ReverseEndian succeeded for type "char"
ReverseEndian succeeded for type "short"
ReverseEndian succeeded for type "long"
ReverseEndian succeeded for type "float"
ReverseEndian succeeded for type "double"
ReverseEndian succeeded for type "void pointer"
ReverseEndian succeeded for type "char pointer"
ReverseEndian succeeded for type "int pointer"



---------------------------------- END OF RUN ---------------------------------
```

```
   From: Shaun Chemplavil <mailto:shaun.chemplavil@gmail.com>
   Subject: C2A7E4_U08713628
   Submitted: 8/22/2020 12:05:16 PM PDT
   Course: C/C++ Programming II (Section 149123)
   Student's name: Shaun Chemplavil
   Contact email: shaun.chemplavil@gmail.com
   Student ID: U08713628
   Assignment 7, Exercise 4  (C2_001405604M02005X65405)
   Exercise point value: 6
   Files submitted:
      C2A7E4_ReverseEndian.c
      C2A7E4_Test-Driver.h
      C2A7E4_main-Driver.c
      C2A7E4_OpenTemporaryFile.c
      C2A7E4_ProcessStructures.c

"Compile-time" results:
   No "compile-time" issues;
"Run-time" results:
   Program ran - No errors detected during preliminary testing (SEE ATTACHMENT);
```

```
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C / C++ Programming II : Dynamic Memory and File I / O Concepts
// 149123 Raymond L.Mitchell, Jr., M.S.
// 08 / 22 / 2020
// C2A7E4_OpenTemporaryFile.c
// Win10
// Visual C++ 19.0
//
// File contains OpenTemporaryFile function, which opens temporary file
// and returns the pointer to that temporary file
//

#include <stdio.h>
#include <stdlib.h>

FILE *OpenTemporaryFile(void)
{
   FILE *source;

   // open fileName in "read" mode, return error if failed
   if ((source = tmpfile()) == NULL)
   {
      fprintf(stderr, "Temporary File failed to open\n");
      exit(EXIT_FAILURE);
   }

   return(source);
}
```

```c
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C / C++ Programming II : Dynamic Memory and File I / O Concepts
// 149123 Raymond L.Mitchell, Jr., M.S.
// 08 / 22 / 2020
// C2A7E4_ProcessStructures.c
// Win10
// Visual C++ 19.0
//
// File contains the functions ReverseMembersEndian, ReadStructures, and
// WriteStructures
//   ReverseMembersEndian: reverses each member of the Test structure passed to
//                         it
//   ReadStructures:  Reads a Test Structure within a temporary file and stores
//                    it at the pointer location passed to it
//   WriteStructures:  Writes a Test Structure to a temporary file and stores
//                    from the pointer location passed to it
// 1. Were the results you got correct for your implementation? Yes
// 2. How many padding bytes were in your structure ?  8 padding bytes
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "C2A7E4_Test-Driver.h"

void *ReverseEndian(void *ptr, size_t size);

struct Test *ReverseMembersEndian(struct Test *ptr)
{
   // Reverse Endian of each structure member individually
   ReverseEndian((void *)&ptr->flt, sizeof(ptr->flt));
   ReverseEndian((void *)&ptr->dbl, sizeof(ptr->dbl));
   ReverseEndian(&ptr->vp, sizeof(ptr->vp));

   return(ptr);
}

struct Test *ReadStructures(struct Test *ptr, size_t count, FILE *fp)
{
   // Read 'count' Test structures from contiguous memory within temporary file
   if (fread(ptr, sizeof(*ptr) * count, 1, fp) != 1)
   {
      fprintf(stderr, "Failed to read structures from temporary file\n");
      exit(EXIT_FAILURE);
   }

   return(ptr);
}

struct Test *WriteStructures(const struct Test *ptr, size_t count, FILE *fp)
{
   // Write 'count' Test structures in contiguous memory within temporary file
   if (fwrite(ptr, sizeof(*ptr) * count, 1, fp) != 1)
   {
      fprintf(stderr, "Failed to write structures to temporary file\n");
      exit(EXIT_FAILURE);
   }

```

```
    return((struct Test *)ptr);
}
```

```c
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C / C++ Programming II : Dynamic Memory and File I / O Concepts
// 149123 Raymond L.Mitchell, Jr., M.S.
// 08 / 20 / 2020
// C2A7E4_ReverseEndian.c
// Win10
// Visual C++ 19.0
//
// File contains ReverseEndian function, which converts any scalar object
// pointed to it from big endian to little endian (or vice versa)
//

#include <stdlib.h>

void *ReverseEndian(void *ptr, size_t size)
{
   // Swap contents at each byte
   for (char *head = (char *)ptr, *tail = head + size - 1;
      tail > head; --tail, ++head)
   {
      // store contents at head pointer
      char temp = *head;
      *head = *tail;
      *tail = temp;
   }

   //Now contents at the memory addresses have been swapped, so return original
   // address (with new content)
   return(ptr);
}
```

```
********** C2 ASSIGNMENT 7 EXERCISE 4 AUTOMATIC PROGRAM RUN RESULTS **********

************* THE RESULTS BELOW HAVE BEEN PARTIALLY CHECKED AND **************
*************   NO ERRORS WERE FOUND.  HOWEVER, THIS DOES NOT    **************
*************  NECESSARILY MEAN THAT THERE ARE NO ERRORS.  THE   **************
*************  INSTRUCTOR WILL DO A MORE THOROUGH CHECK DURING   **************
*************                  MANUAL GRADING.                   **************


---------------------------- PURPOSE OF 1ST RUN ----------------------------
Verify the endian reversal of structure members.
----------------------------- START OF 1ST RUN -----------------------------

IMPORTANT:
The results displayed below are what any correctly written code for this
exercise will produce when run on my system.  However, because type
widths and padding are implementation dependent the results on your system
might differ yet still be correct.  In the output below the only padding is
the second group of 4 bytes in each element.  The easiest way to spot padding
is to look for the three reversed byte sequences in any of the elements.
Anything that is not part of these sequences is padding.


Structure bytes before (1st line) & after (2nd line) reversal:

Element 0:
   cd cc bc 41 ff ff ff ff 5f 07 ce 19 51 da 3b bf 45 23 00 00 00 00 00 00
   41 bc cc cd ff ff ff ff bf 3b da 51 19 ce 07 5f 00 00 00 00 00 00 23 45
Element 1:
   00 00 00 40 f7 7f 00 00 00 00 00 00 00 00 f0 3f 00 00 00 00 00 00 00 00
   40 00 00 00 f7 7f 00 00 3f f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Element 2:
   00 00 c0 c0 f7 7f 00 00 66 66 66 66 66 66 0a 40 00 00 00 00 00 00 00 00
   c0 c0 00 00 f7 7f 00 00 40 0a 66 66 66 66 66 66 00 00 00 00 00 00 00 00


PLEASE BE SURE YOU HAVE ANSWERED THE FOLLOWING QUESTIONS:
   1. Were the results you got correct for your implemenation?
   2. How many padding bytes were in your structure?


------------------------------- END OF 1ST RUN -------------------------------

---------------------------- PURPOSE OF 2ND RUN ----------------------------
Verify that program detects a temporary file open failure.
------------------------- CODE CHANGES FOR 2ND RUN -------------------------
Intentionally induced tmpfile failure.
----------------------------- START OF 2ND RUN -----------------------------

Temporary File failed to open



------------------------------- END OF 2ND RUN -------------------------------
```