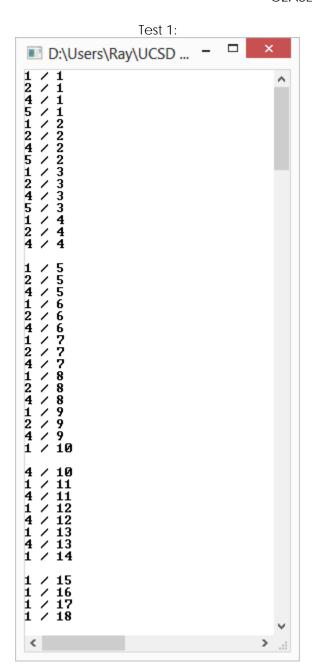
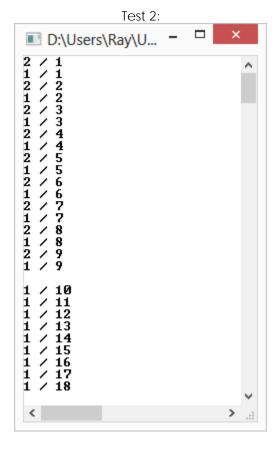
```
1
    Exercise 1 (10 points - C++ Program)
2
      3
4
5
    // ...the usual title block Student/Course/Assignment/Compiler information goes here...
7
    // This file contains function OpenFiles, which opens one or more specified
8
    // files.
9
10
11
    #include <cstdlib>
12
    #include <fstream>
    #include <iostream>
13
14
    using std::cerr;
15
    using std::exit;
16
    using std::ifstream;
17
    using std::nothrow;
18
19
    //
20
    // Use separate ifstream objects to open the number of files specified by
    // <count> having the names specified in <fileNames>. Each ifstream object
21
    // is kept in a dynamically-allocated array and a pointer to the first element
22
    // of that array is returned. If a file fails to open close all previously
23
    // opened files, output an error message, and terminate the program with
24
25
    // an error code.
26
    //
27
    ifstream *OpenFiles(char * const fileNames[], size_t count)
28
29
       if (count == 0)
30
          cerr << "No files specified!\n";</pre>
31
                                                     // syntax error message
                                                     // error termination
32
          exit(EXIT FAILURE);
33
       }
34
35
       ifstream *files;
36
       if ((files = new (nothrow) ifstream[count]) == NULL)// array of ifstream
37
       {
38
          cerr << "new out of memory\n";</pre>
39
          exit(EXIT_FAILURE);
                                                     // error termination
40
       }
41
42
       for (size_t which = 0; which < count; ++which)</pre>
                                                    // for each file
43
          44
          if (!files[which].is open())
                                                     // test the open
45
46
            cerr << "File \"" << fileNames[which] << "\" didn't open.\n";</pre>
47
48
            for (size_t ix = 0; ix < which; ++ix) // for each open file</pre>
49
               files[ix].close();
                                                     // close the file
50
            delete[] files;
                                                     // delete dynamic alloc
51
            exit(EXIT FAILURE);
                                                     // error termination
52
          }
53
       }
54
       return files;
55
56
```

```
1
2
      3
4
     // ...the usual title block Student/Course/Assignment/Compiler information goes here...
5
     //
     // This file contains function MergeAndDisplay, which displays the lines from
     // one or more text files interleaved with each other.
7
8
     //
9
10
     #include <fstream>
11
     #include <iostream>
12
     using std::cout;
13
     using std::ifstream;
14
15
16
     // For each of the <count> text files in <files> read and display the first
17
     // line from the first file, the first line from the second file, the first
     // line from the third file, etc. Then read and display the second line from
18
     // the first file, the second line from the second file, the second line from
19
20
     // the third file, etc. Continue this process until all lines from all files
     // have been displayed. If a file runs out of lines simply close that file
21
22
     // and continue on with the remaining files.
23
     //
24
     void MergeAndDisplay(ifstream files[], size_t count)
25
26
       const int BUFSIZE = 512;
                                                      // size of input buffer
27
28
       for (size t openFiles = count; openFiles;)
                                                 // while any file is open
29
          for (size_t ix = 0; ix < count; ++ix)</pre>
30
                                                      // for each file
31
             if (files[ix].is_open())
32
                                                      // is unread data in file
33
34
                char buf[BUFSIZE];
                                                      // input buffer
35
                if (!files[ix].getline(buf, BUFSIZE)) // read file line; chk EOF
36
37
                   files[ix].close();
                                                      // close the file
38
                   --openFiles;
                                                      // reduce open file count
39
                }
40
                                                      // read a line from file
                else
41
                   cout << buf << '\n';</pre>
                                                      // write line to output
42
             }
43
          }
44
        }
45
     }
```

C2A8E1 Screen Shots are on the next page...

## C2A8E1 Screen Shots





```
Exercise 2 (10 points - C Program)
1
 2
           3
 4
5
      * ...the usual title block Student/Course/Assignment/Compiler information goes here...
6
7
      * This file contains function OpenFileBinary, which opens the specified file
      * in the binary read-only mode.
8
9
10
11
     #include <stdio.h>
12
     #include <stdlib.h>
13
14
15
     * Open the file named in <fileName> in the "read only" binary mode and return
      * its FILE pointer if the open succeeds. If it fails display an error message
16
17
      * and terminate the program with an error code.
18
19
     FILE *OpenFileBinary(const char *fileName)
20
21
       FILE *inFile;
22
23
       /* Open the file named in <fileName> in the read-only binary mode. */
       if ((inFile = fopen(fileName, "rb")) == NULL)
24
25
26
          /* Print an error message and terminate with an error code. */
          fprintf(stderr, "File \"%s\" didn't open.\n", fileName);
27
28
          exit(EXIT FAILURE);
29
       }
30
       return inFile;
31
     }
32
      33
34
35
       ...the usual title block Student/Course/Assignment/Compiler information goes here...
36
37
      * This file contains functions:
          DisplayModifiedSingleReal: Reads groups of 4 bytes from an already open
38
39
             binary file and stores each group in an integer that's at least 32
             bits wide. The file bytes are assumed to be in big endian order;
40
          DisplayModifiedSingleReals: Interprets the least significant 32-bits of
41
             its integer parameter as floating point value in "Modified Single
42
43
             Real" format; displays the bit pattern and interpreted value.
      */
44
45
46
     #include <limits.h>
47
     #include <math.h>
     #include <stdio.h>
48
49
50
     #define MASK USED BITS OxFFFFFFFuL /* mask over all used bits */
51
     #define SIGN MASK 0x8000000uL /* sign field bit mask */
                                     /* exponent field bit mask */
/* fraction field bit mask */
     #define EXP MASK 0x7FC00000uL
52
     #define FRAC MASK 0x003FFFFFuL
53
     #define FRAC BITS 22
                                      /* bits in fraction field */
54
55
     #define EXP_NBIAS 255
                                      /* normalized number exponent bias */
56
    #define EXP DBIAS 254
                                      /* denormalized number exponent bias */
                                      /* exponent maximum value */
57
     #define EXP_MAX 511
```

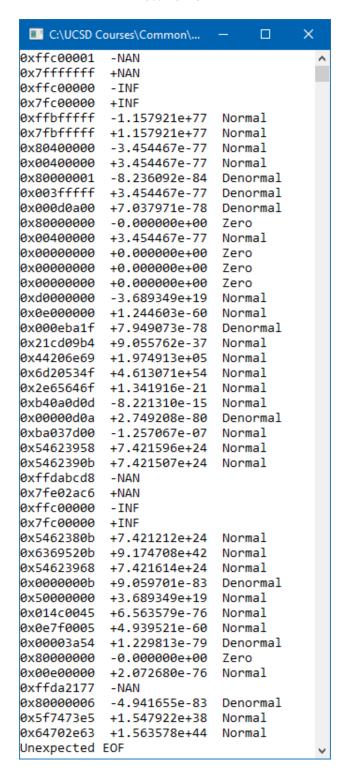
```
1
 2
                                         /* # of bytes to analyze per pattern */
     #define PATTERN BYTES 4
 3
                                         /* radix of representation */
     #define BASE 2.0
 4
5
 6
      * Analyze the bit pattern in <pattern>, treating it as the representation
7
      * of a "Modified Single Real" as described in the requirements for this
8
      * exercise. If the pattern represents either a normalized number, a
      * denormalized number, or a zero, display which one of those it is
9
10
      * along with the numeric value it represents. If the pattern represents
      * an infinity or a NAN simply display which one of those it is.
11
12
      */
13
     static void DisplayModifiedSingleReal(unsigned long pattern)
14
15
        int isNegative = !!(SIGN_MASK & pattern);
                                                             /* neg/pos */
        int exponent = (EXP MASK & pattern) >> FRAC BITS; /* exponent field val */
16
17
        long fraction = FRAC_MASK & pattern;
                                                            /* fraction field val */
18
19
        printf("%s", isNegative ? "-" : "+");
                                                            /* display the sign */
20
21
        if (exponent == 0 && fraction == 0)
                                                            /* test for zero */
22
           printf("%e Zero\n", 0.0);
23
        else if (exponent == EXP_MAX && fraction == 0)
                                                            /* test for infinity */
24
           printf("INF\n");
        else if (exponent == EXP MAX && fraction != 0)
                                                            /* test for NAN */
25
           printf("NAN\n");
26
27
                                                             /* is norm/denorm */
        else
28
        {
29
           double result = fraction * pow(BASE, -FRAC BITS);/* fract. part val */
30
           if (exponent != 0)
                                                             /* if is norm */
31
                                                             /* add implicit 1 */
32
              ++result;
33
              result *= pow(BASE, exponent - EXP_NBIAS);
                                                            /* calc mantissa */
              printf("%e Normal\n", result);
34
35
           }
36
                                                            /* else is denorm */
           else
37
           {
              result *= pow(BASE, exponent - EXP_DBIAS); /* calc mantissa */
38
39
              printf("%e Denormal\n", result);
40
           }
41
        }
42
     }
43
44
45
      * Read successive 32-bit patterns from the file in <inFile>. Each
      * of these patterns represents a "Modified Single Real" as described
46
      * in the requirements for this exercise and is stored in the file in
47
48
      * big endian order. Call function DisplayModifiedSingleReal for each
49
      * pattern to display its format (normalized, denormalized, etc.), and
      * also its value if it is not an infinity or a NAN.
50
51
     void DisplayModifiedSingleReals(FILE *inFile)
52
53
54
        for (;;)
55
56
           /* Array to hold pattern bytes. */
57
           unsigned char patternArray[PATTERN BYTES];
```

```
1
            * Read the next pattern. This cannot be placed directly into a scalar
 2
 3
            * type due to the implementation-dependent widths of such types and
            * because this algorithm must handle the big endian data from the
 4
5
            * file independent of the endianness of the machine running this code.
 6
            */
 7
           size_t bytesRead = fread(patternArray, 1, PATTERN_BYTES, inFile);
8
           /* If complete pattern was read. */
           if (bytesRead == PATTERN_BYTES)
9
10
           {
11
              unsigned long result = 0;
              int byteIx, shiftIx = PATTERN_BYTES - 1;
12
13
              /* Shift each byte into the correct position in result */
              for (byteIx = 0; byteIx < PATTERN_BYTES; ++byteIx, --shiftIx)</pre>
14
15
                  result |= patternArray[byteIx] << (CHAR_BIT * shiftIx);</pre>
              printf("0x%08lx ", result & MASK_USED_BITS);
16
              DisplayModifiedSingleReal(result);
17
18
19
           /* Else, partial or no pattern was read. */
20
           else
21
           {
22
              if (bytesRead > 0)
23
                  printf("Unexpected EOF\n");
24
              break;
25
           }
26
        }
27
     }
```

C2A8E2 Screen Shots are on the next page...

## C2A8E2 Screen Shots

TestFile7.bin TestFile8.bin



C:\UCSD C	ourses\Commo	- 🗆	×
	+7.949073e-78	Denormal	^
0xffc00001			
0x7fffffff	+NAN		
0x014c0045	+6.563579e-76	Normal	
0x7fc00000	+INF		
0xffbfffff	-1.157921e+77	Normal	
0x80400000	-3.454467e-77	Normal	
0x00400000	+3.454467e-77	Normal	
0x80000001	-8.236092e-84	Denormal	
0x003fffff	+3.454467e-77	Denormal	
0x0000000b	+9.059701e-83	Denormal	
0x7fc00000	+INF		
0x00000000	+0.000000e+00	Zero	
0x80000000	-0.000000e+00	Zero	
0x7fbfffff	+1.157921e+77	Normal	
0x00400000	+3.454467e-77	Normal	
0000000000	+0.000000e+00	Zero	
0x7fe02ac6	+NAN		
00000000x6	+0.000000e+00	Zero	
00000000x6	+0.000000e+00	Zero	
0000000000	-3.689349e+19	Normal	
	+1.244603e-60	Normal	
0x64702e63	+1.563578e+44	Normal	
0x21cd09b4		Normal	
0x44206e69		Normal	
0x6d20534f	+4.613071e+54	Normal	
0xb40a0d0a	-8.221305e-15	Normal	
0x00000000	+0.000000e+00	Zero	
0x00003a54	+1.229813e-79	Denormal	
0xba037d00	-1.257067e-07	Normal	
0x54623958		Normal	
0x5462390b		Normal	
0xffdabcd8		1101 1102	
0xffc00000			
	+7.421212e+24	Normal	
	+9.174708e+42		
0x54623968	+7.421614e+24	Normal	
0x50000000	+3.689349e+19	Normal	
0x2e65646f	+1.341916e-21	Normal	
0x0e7f0005	+4.939521e-60	Normal	
0x80000000	-0.000000e+00	Zero	
0x00e00000	+2.072680e-76	Normal	
0xffda2177	-NAN	MOLINAT	
0x1Tua21// 0x80000006	-4.941655e-83	Denormal	
0x80000000 0x5f7473e5	-4.941655e-83 +1.547922e+38	Normal	
0x5f7473e5 0xffc00000	+1.54/922e+38 -INF	MOLINAT	
DYLLCARARA	- TIME		