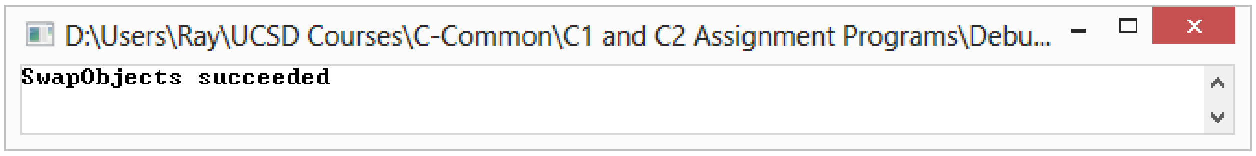
## Exercise 1 *(4 points – C Program)*

```c
/*
 * ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains function SwapObjects, which swaps the contents of the
 * objects specified by its first two parameters.
 */

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

/*
 * Swap the object in <pa> with the object of the same size in <pb>.  The
 * number of bytes in each object is specified by <size>.
 */
void SwapObjects(void *pa, void *pb, size_t size)
{
   void *ptr;

   /*
    * Dynamically allocate enough memory to hold one object.  Terminate the
    * the program with an error message and code if it fails.
    */
   if ((ptr = malloc(size)) == NULL)
   {
      fputs("Out of memory\n", stderr);
      exit(EXIT_FAILURE);
   }

   /*
    * Do the standard 3-step swap using memcpy to copy an entire object
    * at once.  Free the dynamically-allocated memory when finished.
    */
   memcpy(ptr, pa, size);     /* save object *pa to temporary */
   memcpy(pa, pb, size);      /* write object *pb onto object *pa */
   memcpy(pb, ptr, size);     /* write saved object onto object *pb */
   free(ptr);                 /* free dynamically-allocated memory */
}
```

C2A5E1 Screen Shot


D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Debu...

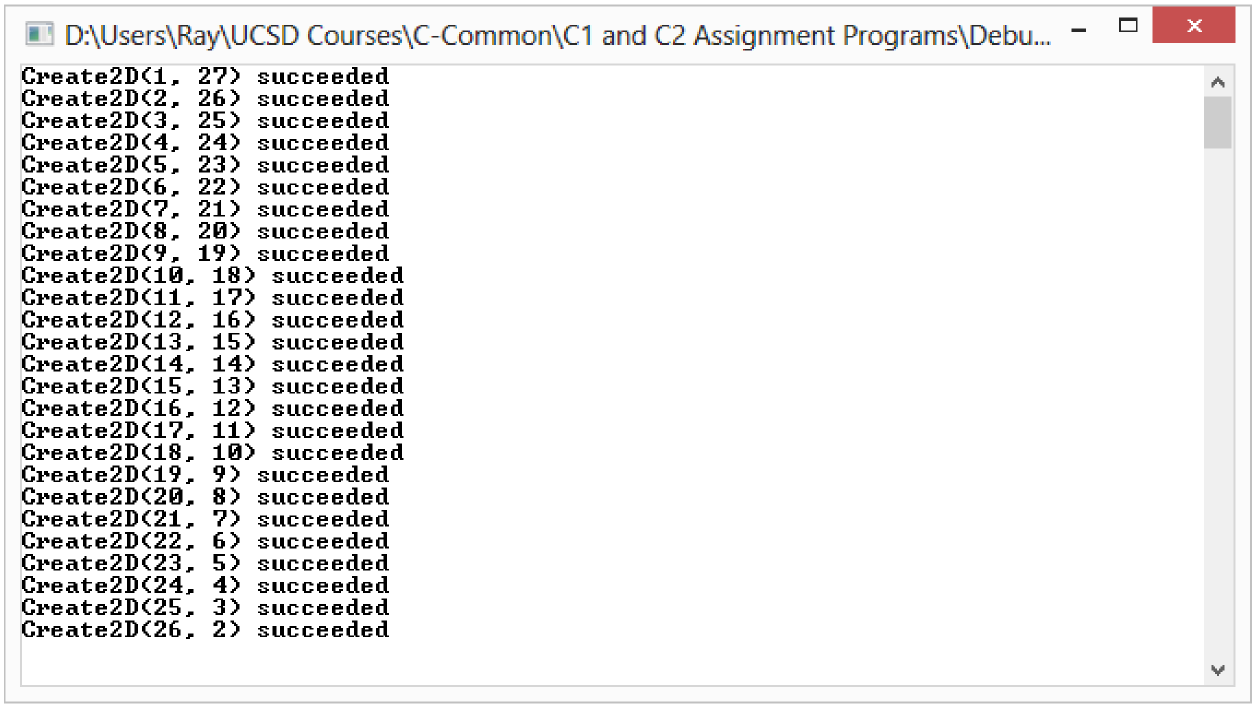
```
SwapObjects succeeded
```

## Exercise 2 *(6 points – C Program)*

```c
/*
 * ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains functions:
 *    Create2D: Dynamically creates and returns access to a 2D array of Type
 *              having the dimension values specified by its two parameters.
 *    Free2D: Frees any array created by the Create2D function, above.
 */

#include <stdio.h>
#include <stdlib.h>
#include "C2A5E2_Type-Driver.h"

/*
 * Dynamically allocate memory for a 2D pointer array of type <Type> having
 * <rows> rows and <cols> columns.  Initialize all row pointers to point to
 * the first element in each row.  Return a pointer to the first row pointer.
 * The array will then be usable by the caller as p[row][col], where <p> is
 * the returned pointer.  Since this algorithm mixes data types within the
 * same allocation block, memory alignment issues are possible on some
 * implementations.
 */
Type **Create2D(size_t rows, size_t cols)
{
   Type **pStart, **pCur, **pEnd, *pRow;

   /*
    * Dynamically allocate memory for all row pointers & rows in the array at
    * once.  Terminate the program with an error message and code if it fails.
    */
   if (!(pStart = (Type **)malloc(
      rows * sizeof(Type *) + rows * cols * sizeof(Type))))
   {
      fputs("malloc out of memory\n", stderr);
      exit(EXIT_FAILURE);
   }

   /* Initialize row pointers to point to first element in each row. */
   for (pRow = (Type *)(pEnd = pStart + rows), pCur = pStart;
      pCur < pEnd; ++pCur, pRow += cols)
   {
      *pCur = pRow;
   }
   return(pStart);
}

/*
 * Free the block of dynamically allocated memory pointed to by
 * parameter <p>.
 */
void Free2D(void *p)
{
   free(p);
}
```

C2A5E2 Screen Shot is on the next page...

C2A5E2 Screen Shot

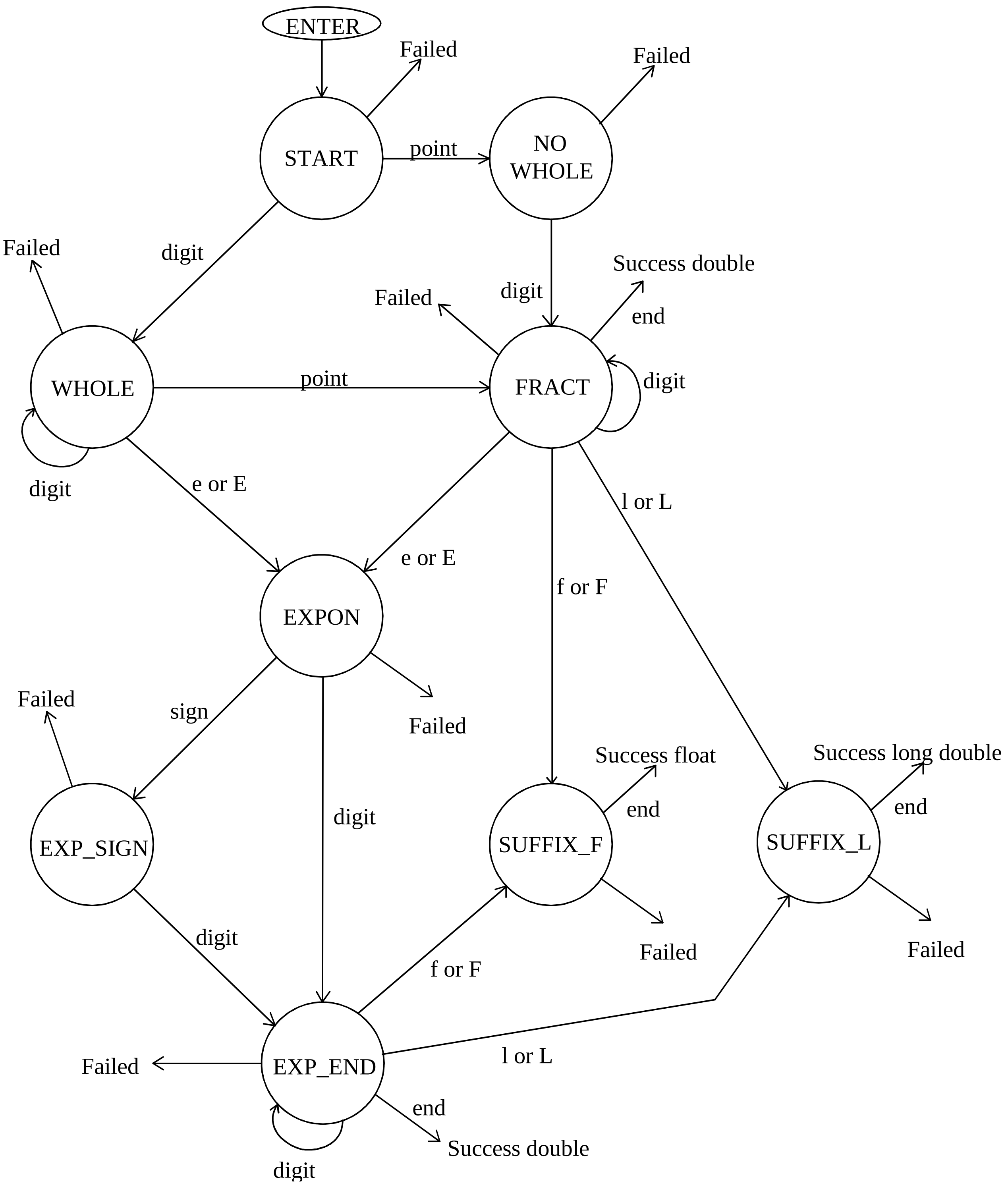

D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Debu...

```
Create2D(1, 27) succeeded
Create2D(2, 26) succeeded
Create2D(3, 25) succeeded
Create2D(4, 24) succeeded
Create2D(5, 23) succeeded
Create2D(6, 22) succeeded
Create2D(7, 21) succeeded
Create2D(8, 20) succeeded
Create2D(9, 19) succeeded
Create2D(10, 18) succeeded
Create2D(11, 17) succeeded
Create2D(12, 16) succeeded
Create2D(13, 15) succeeded
Create2D(14, 14) succeeded
Create2D(15, 13) succeeded
Create2D(16, 12) succeeded
Create2D(17, 11) succeeded
Create2D(18, 10) succeeded
Create2D(19, 9) succeeded
Create2D(20, 8) succeeded
Create2D(21, 7) succeeded
Create2D(22, 6) succeeded
Create2D(23, 5) succeeded
Create2D(24, 4) succeeded
Create2D(25, 3) succeeded
Create2D(26, 2) succeeded
```

## Exercise 3 *(5 points – Diagram only – No program required)*

This diagram assumes:

a. The next character is available as each state is entered;
b. "Failed" indicates an exit where the string was not a floating literal;
c. "Success ..." indicates an exit where the string was a floating literal of the specified type;
d. "end" means the end of the string was reached



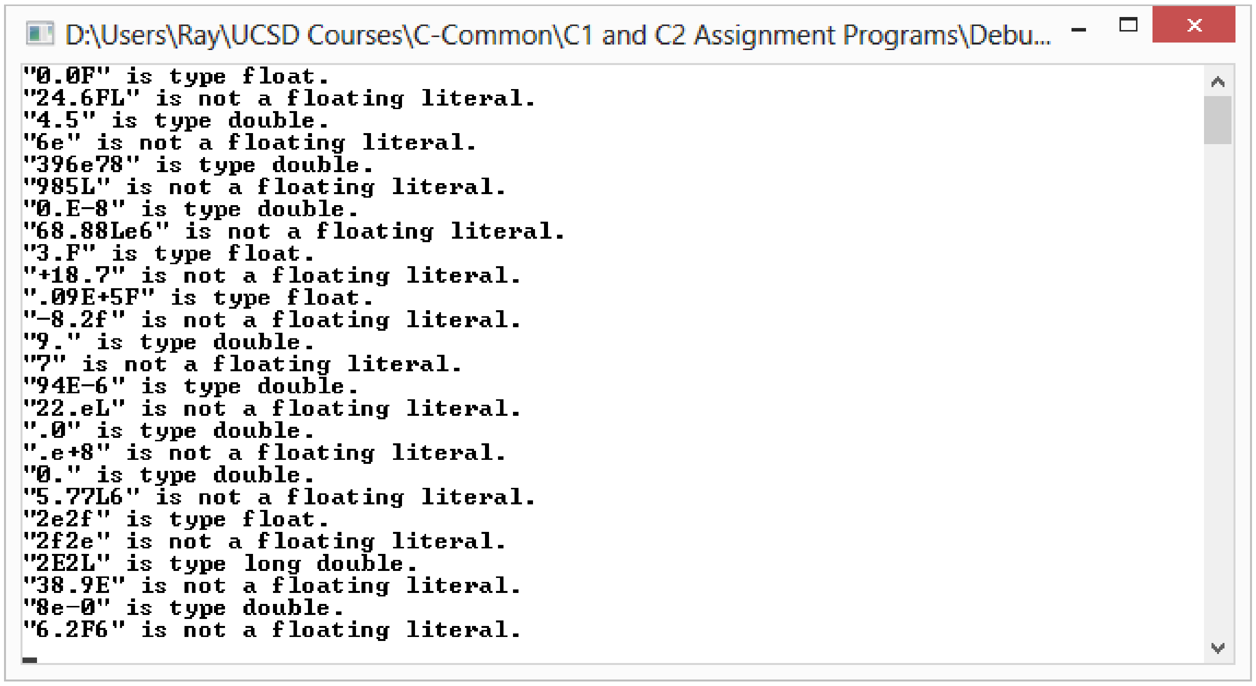
## Exercise 4 *(5 points – C++ Program)*

```
   *************************************** FILE C2A5E4_OpenFile.cpp ***************************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function OpenFile, which opens in the read-only mode the
// file specified by its first parameter using the object specified by its
// second parameter.
//

#include <fstream>
#include <iostream>
#include <cstdlib>

//
// Open the file named in <fileName> using the object referenced by
// <inFile>.  If it fails display an error message and terminate the
// program with an error code.
//
void OpenFile(const char *fileName, std::ifstream &inFile)
{
   // Open file for read only.
   inFile.open(fileName);
   // If open fails print an error message and terminate with an error code.
   if (!inFile.is_open())
   {
      std::cerr << "File \"" << fileName << "\" didn't open.\n";
      std::exit(EXIT_FAILURE);
   }
}

   ************************************* FILE C2A5E4_DetectFloats.cpp *************************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains functions:
//    DetectFloats: Determines if its parameter string is a floating literal;
//    IsFloat: Determines if floating literal's suffix is for float;
//    IsLDouble: Determines if floating literal's suffix is for long double;
//    IsExponent: Determines if floating literal contains an exponent;
//    IsSign: Determines if floating literal's exponent contains a sign;
// It also contains a definition of enumerated type enum States; used in the
// state machine.
//

#include <cctype>
#include "C2A5E4_StatusCode-Driver.h"

//
// Definition of enumerations returned by the state machine from function
// DetectFloats to indicate the result of its analysis of its parameter
// string.
//
enum States
{
   START, WHOLE, NO_WHOLE, FRACT, EXPON, EXP_SIGN, EXP_END,
   SUFFIX_F, SUFFIX_L
};

//
// Inline functions to test for suffix and exponent characters.
//
static inline bool IsFloat(int ch)    { return ch == 'f' || ch == 'F'; }
static inline bool IsLDouble(int ch)  { return ch == 'l' || ch == 'L'; }
static inline bool IsExponent(int ch) { return ch == 'e' || ch == 'E'; }
static inline bool IsSign(int ch)     { return ch == '+' || ch == '-'; }

//
// Implement a state machine to detect if the string in <chPtr> represents
// a legal floating point literal according to the definition in the
// C and C++ language standards documents.  Return a code indicating the
// result of each analysis.
//
StatusCode DetectFloats(const char *chPtr)
{
   States state = START;                      // machine state

   for (;; ++chPtr)
   {
      switch (state)                          // go to indicated state
      {
         case START:                          // looking for initial '.' or digit
            if (*chPtr == '.')                // decimal point
               state = NO_WHOLE;              // set up for new state
            else if (std::isdigit(*chPtr)) // digit
               state = WHOLE;                 // set up for new state
            else
               return(NOTFLOATING);
            break;
         case NO_WHOLE:                       // found initial '.'
            if (std::isdigit(*chPtr))         // digit
               state = FRACT;                 // set up for new state
            else
               return(NOTFLOATING);
            break;
         case WHOLE:                          // found initial digit
            if (!std::isdigit(*chPtr))        // not a digit
               if (*chPtr == '.')             // decimal point
                  state = FRACT;              // set up for new state
               else if (IsExponent(*chPtr))// 'e' or 'E'
                  state = EXPON;              // set up for new state
               else
                  return(NOTFLOATING);
            break;
         case FRACT:                          // doing fractional part
            if (!std::isdigit(*chPtr))        // not a digit
               if (*chPtr == '\0')            // end of string
                  return(TYPE_DOUBLE);        // string is double
               else if (IsFloat(*chPtr))      // 'f' or 'F'
                  state = SUFFIX_F;           // set up for new state
               else if (IsLDouble(*chPtr)) // 'l' or 'L'
                  state = SUFFIX_L;           // set up for new state
               else if (IsExponent(*chPtr))// 'e' or 'E'
```

```
                  state = EXPON;           // set up for new state
               else
                  return(NOTFLOATING);
            break;
         case EXPON:                       // doing exponent part
            if (std::isdigit(*chPtr))      // is a digit
               state = EXP_END;            // set up for new state
            else if (IsSign(*chPtr))       // '+' or '?'
               state = EXP_SIGN;           // set up for new state
            else
               return(NOTFLOATING);
            break;
         case EXP_SIGN:
            if (std::isdigit(*chPtr))      // is a digit
               state = EXP_END;            // set up for new state
            else
               return(NOTFLOATING);
            break;
         case EXP_END:
            if (!std::isdigit(*chPtr))     // not a digit
               if (*chPtr == '\0')         // end of string
                  return(TYPE_DOUBLE);     // string is double
               else if (IsFloat(*chPtr))   // 'f' or 'F'
                  state = SUFFIX_F;        // set up for new state
               else if (IsLDouble(*chPtr)) // 'l' or 'L'
                  state = SUFFIX_L;        // set up for new state
               else
                  return(NOTFLOATING);
            break;
         case SUFFIX_F:
            if (*chPtr == '\0')            // end of string
               return(TYPE_FLOAT);
            else
               return(NOTFLOATING);
         case SUFFIX_L:
            if (*chPtr == '\0')            // end of string
               return(TYPE_LDOUBLE);
            else
               return(NOTFLOATING);
      }
   }
}
```

C2A5E4 Screen Shot is on the next page…

C2A5E4 Screen Shot


D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Debu...

```
"0.0F" is type float.
"24.6FL" is not a floating literal.
"4.5" is type double.
"6e" is not a floating literal.
"396e78" is type double.
"985L" is not a floating literal.
"0.E-8" is type double.
"68.88Le6" is not a floating literal.
"3.F" is type float.
"+18.7" is not a floating literal.
".09E+5F" is type float.
"-8.2f" is not a floating literal.
"9." is type double.
"7" is not a floating literal.
"94E-6" is type double.
"22.eL" is not a floating literal.
".0" is type double.
".e+8" is not a floating literal.
"0." is type double.
"5.77L6" is not a floating literal.
"2e2f" is type float.
"2f2e" is not a floating literal.
"2E2L" is type long double.
"38.9E" is not a floating literal.
"8e-0" is type double.
"6.2F6" is not a floating literal.
```