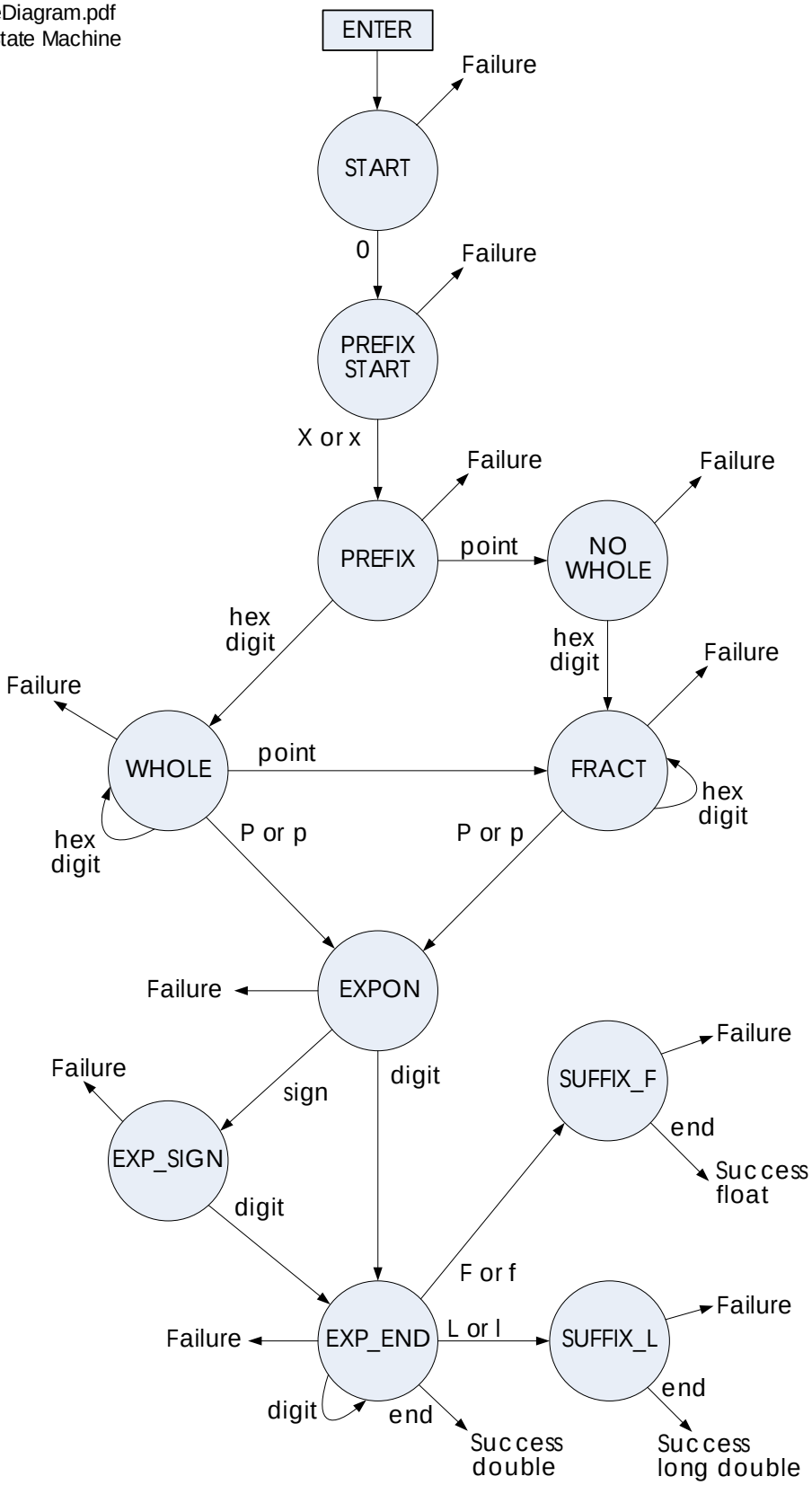
```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A5E1_SwapObjects.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function SwapObjects, which swaps the contents 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)
{
   //
   // Dynamically allocate enough memory to hold one object.  Terminate the
   // program with an error message and code if it fails.
   //
   void *ptr;
   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
}
```

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A5E2_Create2D.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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)
{
   //
   // 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.
   //
   Type **pS = (Type **)malloc(rows * (sizeof(Type *) + cols * sizeof(Type)));
   if (!pS)
   {
      fputs("malloc out of memory\n", stderr);
      exit(EXIT_FAILURE);
   }

   // Initialize row pointers to point to first element in each row.
   Type **end = pS + rows, *pCol = (Type *)end;
   for (Type **pRow = pS; pRow < end; ++pRow, pCol += cols)
      *pRow = pCol;

   return pS;
}

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

Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming II
Section 888888, Ray Mitchell
January 1, 2018
C2A5E3_StateDiagram.pdf
DetectFloats State Machine

1. The next character is available as each state is entered;
2. “Failure” returns 0; “Success” returns the indicated data type value;
3. “end” means the end of the string was reached.



```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming II
5   // Section 149123, Ray Mitchell
6   // June 25, 2019
7   // C2A5E4_DetectFloats.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains functions:
12  //    DetectFloats: Determines if its parameter string is a hexadecimal
13  //       floating literal.
14  //    IsFloat: Determines if floating literal's suffix is for float.
15  //    IsLDouble: Determines if floating literal's suffix is for long double.
16  //    IsExponent: Determines if a floating literal exponent indicator.
17  //    IsSign: Determines if a floating literal exponent sign.
18  // It also contains a definition of enumerated type enum States used in the
19  // state machine.
20  //
21
22  #include <cctype>
23  #include "C2A5E4_StatusCode-Driver.h"
24  using std::isdigit;
25  using std::isxdigit;
26
27  //
28  // Definition of machine state enumerations.
29  //
30  enum States
31  {
32     START, PREFIX_START, PREFIX, WHOLE, NO_WHOLE, FRACT, EXPON, EXP_SIGN,
33     EXP_END, SUFFIX_F, SUFFIX_L
34  };
35
36  //
37  // Inline functions to test for special characters.
38  //
39  static inline bool IsFloat(int ch)    { return ch == 'f' || ch == 'F'; }
40  static inline bool IsLDouble(int ch)  { return ch == 'l' || ch == 'L'; }
41  static inline bool IsExponent(int ch) { return ch == 'p' || ch == 'P'; }
42  static inline bool IsSign(int ch)     { return ch == '+' || ch == '-'; }
43  static inline bool IsHexPrefixEnd(int ch) { return ch == 'x' || ch == 'X'; }
44
45  //
46  // Implements a state machine to detect if the string in <chPtr> represents a
47  // hexadecimal floating point literal according to the definition in the C and
48  // C++ language standards documents.  Returns a code indicating the result of
49  // each analysis.
50  //
51  StatusCode DetectFloats(const char *chPtr)
52  {
53     States state = START;                     // machine state
54
55     for (;; ++chPtr)
56     {
57        switch (state)                         // go to indicated state
58        {
59           case START:                         // looking for hex prefix start
60              if (*chPtr == '0')               // hex prefix start
61                 state = PREFIX_START;         // set up for new state
```

```
62              else
63                 return(NO_MATCH);
64              break;
65           case PREFIX_START:                  // looking for hex prefix end
66              if (IsHexPrefixEnd(*chPtr))     // hex prefix end
67                 state = PREFIX;              // set up for new state
68              else
69                 return(NO_MATCH);
70              break;
71           case PREFIX:                        // found entire prefix
72              if (*chPtr == '.')               // radix point
73                 state = NO_WHOLE;            // set up for new state
74              else if (isxdigit(*chPtr))       // hex digit
75                 state = WHOLE;               // set up for new state
76              else
77                 return(NO_MATCH);
78              break;
79           case NO_WHOLE:                      // found initial radix point
80              if (isxdigit(*chPtr))            // hex digit
81                 state = FRACT;               // set up for new state
82              else
83                 return(NO_MATCH);
84              break;
85           case WHOLE:                         // found initial digit
86              if (!isxdigit(*chPtr))           // not a hex digit
87                 if (*chPtr == '.')            // radix point
88                    state = FRACT;             // set up for new state
89                 else if (IsExponent(*chPtr))  // 'p' or 'P'
90                    state = EXPON;             // set up for new state
91                 else
92                    return(NO_MATCH);
93              break;
94           case FRACT:                         // doing fractional part
95              if (!isxdigit(*chPtr))           // not a hex digit
96                 if (*chPtr == '\0')           // end of string
97                    return(TYPE_DOUBLE);       // string is double
98                 else if (IsExponent(*chPtr))  // 'p' or 'P'
99                    state = EXPON;             // set up for new state
100                else
101                   return(NO_MATCH);
102             break;
103          case EXPON:                         // doing exponent part
104             if (isdigit(*chPtr))             // is a digit
105                state = EXP_END;              // set up for new state
106             else if (IsSign(*chPtr))         // '+' or '-'
107                state = EXP_SIGN;             // set up for new state
108             else
109                return(NO_MATCH);
110             break;
111          case EXP_SIGN:
112             if (isdigit(*chPtr))             // is a digit
113                state = EXP_END;              // set up for new state
114             else
115                return(NO_MATCH);
116             break;
117          case EXP_END:
118             if (!isdigit(*chPtr))            // not a digit
119                if (*chPtr == '\0')           // end of string
120                   return(TYPE_DOUBLE);       // string is double
121                else if (IsFloat(*chPtr))     // 'f' or 'F'
```

```
122                     state = SUFFIX_F;           // set up for new state
123                  else if (IsLDouble(*chPtr))    // 'l' or 'L'
124                     state = SUFFIX_L;           // set up for new state
125                  else
126                     return(NO_MATCH);
127               break;
128            case SUFFIX_F:
129               if (*chPtr == '\0')               // end of string
130                  return(TYPE_FLOAT);
131               else
132                  return(NO_MATCH);
133            case SUFFIX_L:
134               if (*chPtr == '\0')               // end of string
135                  return(TYPE_LDOUBLE);
136               else
137                  return(NO_MATCH);
138         }
139      }
140   }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming II
5   // Section 149123, Ray Mitchell
6   // June 25, 2019
7   // C2A5E4_OpenFile.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function OpenFile, which opens a specified file in the
12  // read-only mode.
13  //
14
15  #include <fstream>
16  #include <iostream>
17  #include <cstdlib>
18
19  //
20  // Function OpenFile opens the file named in <fileName> in the read-only mode
21  // using the object referenced by <inFile>.  Upon failure an error message is
22  // displayed and the program is terminated with an error code.
23  //
24  void OpenFile(const char *fileName, std::ifstream &inFile)
25  {
26     // Open file for read only.
27     inFile.open(fileName);
28     // If open fails print an error message and terminate with an error code.
29     if (!inFile.is_open())
30     {
31        std::cerr << "File \"" << fileName << "\" didn't open.\n";
32        std::exit(EXIT_FAILURE);
33     }
34  }
```