```c
/* secant_colebrook.h
 * Ryan Jensen
 * 2013-11-15 (created)
 * 2013-11-18 (last modified)
 * This header file contains the secant_colebrook() function.
 * This function will solve the Colebrook equation for the friction factor f
 * numerically.
*/

//used for the nice math functions like log10() and sqrtf()
#include <math.h>

// this is a flag that denotes success in finding a zero
#define SECANT_COLEBROOK_SUCCESS 1.0f
// this is a flag that denotes a failure to find a zero
#define SECANT_COLEBROOK_FAILURE 0.0f

/// this function solves the Colebrook equation for the friction factor, f.
/// the return value is a pointer to two floating point values.
/// the first floating point number is the friction factor f
/// the second floating point number is an error flag.
float * secant_colebrook(float Re, float D, float epsilon){

    ///------------------------------------------------------------------------
    ///     SET UP VARIABLES
    ///------------------------------------------------------------------------

    // this is where we store the frictionFactor.
    // ffact[0] is the first guess.
    // ffact[1] is the second guess.
    // ffact[2] is the next guess.
    static float ffact[3];
    // this stores the difference between one side and the other side of the
    // colebrook equation. Lets call it y.
    // y[0] is for the first guess.
    // y[1] is for the second guess.
    // y[2] is for the next guess.
    float y[3];
    // this stores the slope of the line between.
    float slope;

    // this is the f_tol for relative error in our value.
    float f_tol = 0.01;
    //this stores the relative error at the current iteration
    float relErr;
    // this keeps track of how many times the function has gone through the loop.
    int iter;
    //this is the maximum number of times that you can go through the while() loop.
    int maxItt = 200;

    // define an array of floating point values with two elements.
    // the first element will store the friction factor.
    // the second value will store the success value.
    static float returnValues[2];

    ///------------------------------------------------------------------------
    ///     EVALUATE THE ROOT OF THE COLEBROOK EQUATION
    ///------------------------------------------------------------------------

    //initial guesses for the friction factor
    ffact[0] = 0.1;
    ffact[1] = 0.01;
    //initial value to make sure we can enter the loop
    ffact[2] = 10;
    // start iter at 0
    iter = 0;
```

```
67      // make sure the relative error is greater than f_tol to enter the loop
68      relErr = 100*f_tol;
69
70      while(relErr > f_tol && iter < maxItt && ffact[2] >= 0.0){
71
72          // evaluate the Colebrook equation at point 0
73          y[0]= (1/sqrtf(ffact[0]));
74          y[0]+= 2*log10((epsilon/(3.7f * D))+(2.51f/(Re*sqrtf(ffact[0]))));
75          // evaluate the Colebrook equation at point 1
76          y[1]= (1/sqrtf(ffact[1]));
77          y[1]+= 2*log10((epsilon/(3.7f * D))+(2.51f/(Re*sqrtf(ffact[1]))));
78
79          // if the slope is 0 or infinite, break and return a failure flag.
80          if(ffact[1] == ffact[0] || y[1] == y[0])break;
81
82          // calculate the slope
83          slope = (y[1]-y[0])/(ffact[1]-ffact[0]);
84
85          // calculate where the next x value will be located
86          ffact[2] = ffact[0] - y[0]/slope;
87
88
89          //store the second guess in the first guess's spot
90          ffact[0] = ffact[1];
91          // store new ffact value in the second's guess spot
92          ffact[1] = ffact[2];
93          //increment the iteration counter.
94          iter++;
95
96          //calculate relative error
97          relErr = (ffact[1]-ffact[0])/ffact[1];
98          // take absolute value of relative error calculation
99          if(relErr < 0) relErr *= -1.0f;
100
101     }
102
103     ///------------------------------------------------------------------------
104     ///     DETERMINE RETURN VALUES AND EXIT FUNCTION
105     ///------------------------------------------------------------------------
106
107     // the program has failed if any of the following are true:
108     //      - the relative error is still larger than the f_tol
109     //      - the loop went through too many iter
110     //      - the slope of the line if infinite or zero
111     //      - the ffact value that will be returned is negative
112     if(relErr>f_tol || iter>=maxItt || y[0]==y[1] || ffact[2]<0){
113         // return the value of the invalid friction factor (whatever it may be)
114         returnValues[0] = ffact[2];
115         // set success flag to FAILURE
116         returnValues[1] = SECANT_COLEBROOK_FAILURE;
117         return returnValues;
118     }
119     // you must have succeeded if nothing went wrong.
120     else{
121         // return the value of the correctly calculated friction factor
122         returnValues[0] = ffact[2];
123         // set success flag to SUCCESS
124         returnValues[1] = SECANT_COLEBROOK_SUCCESS;
125         return returnValues;
126     }
127     // the function should never get to this point.
128     // but if it does, return a failure flag.
129     returnValues[0] = 0.0f;
130     returnValues[1] = SECANT_COLEBROOK_FAILURE;
131     return returnValues;
132 }
```