```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

char Alloy[20];                 // The alloy in question
char Element[20];               // The element in question
double D0_Min, D0_Max;          // Bounds for the infinite diffusivity (D0)
double Ac_Min, Ac_Max;          // Bounds for the activation energy (Ac) in kcal/mol
double D0_Step;                 // The step change for infinite diffusivity
double Ac_Step;                 // The step change for activation energy in kcal/mol
unsigned int D0_Dimension;      // The number of D0 values to use, calculated from the bounds and
                                        // the step size of D0
unsigned int Ac_Dimension;      // The number of Ac values to use, calculated from the bounds and
                                        // the step size of Ac
double Conc_Min, Conc_Max;      // The minimum and maximum (bounding) concentrations for the element
                                        // in question in atomic percent
double delta_C, Cavg;           // Values used in the diffusion equation
unsigned int NumDatasets;       // The number of data sets to analyze
unsigned char LastControlSet;   // The index of the last dataset that was not aged - used for
                                        // constructing the Effective Starting Times

struct DataSet
{
        unsigned long Seconds;          // The number of seconds for which the
                                                // diffusion couple was heated
        unsigned int Temperature;       // The temperature at which the diffusion couple was heated
                                                // in Kelvin
        unsigned int NumberOfDataPoints;// The number of data points in this data set
        double Cmin;
        double Cmax;
        double Cavg;
        double DeltaC;
        double X_Values[255];           // The distances from the interface in microns
        double Y_Values[255];           // The point concentration of the element
                                                // in question in atomic percent
};

struct DataSet CurrentData[10];
```

```
const double R = 0.001986;              // Gas constant in kcal/K-mol

void GetGlobalParameters(void);         // Get the global parameters (value bounds) for the current
                                                // diffusion problem
void LoadDataIntoMemory(void);          // Load the data set from the input file
                                                // into memory for processing
double DC(double, double, unsigned int);        // Calculate diffusion constant (D) from a given D0,
                                                        // Ac and temperature
double PointConc(double, double, unsigned long, unsigned char);
                        // Calculate the theoretical element concentration from
                        // a given D, distance and time

int main (void)
{
        unsigned int i, j, l, m, n;             // Dummy variables for loops
        unsigned long k;
        double Local_D0, Local_Ac, Local_D;
                                        // Local values of D0 and Ac, used for computing each step
        unsigned long Local_Time;       // Local copy of the time in seconds.  Used in computations
        double RMS_Sum;         // The total RMS value for matching the theory with
                                        // the current experiment
        double Min_RMS;         // Keeps track of the minimum RMS value, used to find the best time
        unsigned long MinimumTime;      // Keeps track of the minimum time value reached in the
                                                // first step
        double D_Scale;         // Temporary diffusion constant multiplied by a power of ten
                                        // specified below
        double SinglePointMSE;          // The result of a single MSE calculation for one point
        unsigned char AbortLoopFlag;    // If this flag is set, it will stop going through the
                                                // EffectiveStartingTime loop
        double AbsoluteMinimumRMS;      // This is the absolute minimum value of the total RMS
        double AbsoluteMinimumX, AbsoluteMinimumY;
                                // These are the coordinates (in D0-Ac space) of the minimum point
        unsigned long AbsoluteMinimumTime;

        FILE *fp = fopen("output-time.csv", "w");
        FILE *fp_results = fopen("output_results.csv", "w");
        FILE *fp_gnuplot = fopen("output_gnuplot.dat", "w");

        LoadDataIntoMemory();
```

```
GetGlobalParameters();

unsigned long EffectiveStartingTimes[D0_Dimension+1] [Ac_Dimension+1][NumDatasets];
        // Effective starting times
double RMS_Map[D0_Dimension+1][Ac_Dimension+1];
        // This is the final output map of RMS values, in D0 / Ac space.
        // This will be output to a file to help the user figure out
        // where the best fit values are for the entire problem.

// First generate an "effective control aging time" for each combination
// of D0 and Ac for each data set. This generates a three dimensional
// array of "effective control aging times", with the X Y and Z axes
// being D0, Ac and Dataset Number.  Each of these values (in seconds)
// will then be added to the recorded time to get an effective aging
// time.  This is because making the diffusion couples causes some
// diffusion to happen, so that the initial concentration profile is
// not a perfect step function.

printf("\n\nReady to run simulation with Cmin = %f, Cmax = %f", Conc_Min, Conc_Max);
printf("\nYou have %u datasets.\n", NumDatasets);
printf("\nD0_Dimension = %u, Ac_Dimension = %u\n", D0_Dimension, Ac_Dimension);
getchar();


for (l=0; l<NumDatasets; l++)
{
        if (CurrentData[l].Seconds == 0) LastControlSet = l;

        for (i=0; i<=D0_Dimension; i++)         // Loop over specified values of D0
        {
                for (j=0; j<=Ac_Dimension; j++) // Loop over specified values of Ac
                {
                        AbortLoopFlag = 0;

                        Local_D0 = D0_Min + ((double)i * D0_Step);
                        Local_Ac = Ac_Min + ((double)j * Ac_Step);

                        Local_D = DC(Local_D0, Local_Ac, CurrentData[l].Temperature);
                        D_Scale = Local_D * pow(10, 13);
```

```
//                              printf("\nD0 = %.3lf, Ac = %.3lf, D = %.3lf * 10^-13, #points = %u,\
                           =>   ", Local_D0, Local_Ac, D_Scale, CurrentData[l].NumberOfDataPoints);

                                EffectiveStartingTimes[i][j][l] = 0;

                                if (CurrentData[l].Seconds == 0)
        // Only compute effective starting times if this dataset is a control (no aging)
                                {

                                        for (k=3600; k<=360000000; k+=3600)
        // Step through in units of 0.01 hours, start at 0.01 hours to avoid divide by zero errors
                                        {

                                                RMS_Sum = 0;

                                                if (AbortLoopFlag == 0)
                                                {

                                                        for (m=0; \
                                                        m<CurrentData[l].NumberOfDataPoints; m++)
        // Computes the RMS for the curve at the specified time.  Loops through data points.
                                                        {
// printf("X = %lf cm, ", CurrentData[l].X_Values[m]);
        // (DEBUG) Shows the current X-coordinate
                                        SinglePointMSE = sqrt(pow((CurrentData[l].Y_Values[m] - \
                                        PointConc(Local_D, CurrentData[l].X_Values[m], k, l)), 2));
// printf("X_Ex = %lf, X_Th = %lf, time = %lu seconds, MSE = %lf\n", CurrentData[l].Y_Values[m], \
        PointConc(Local_D, CurrentData[l].X_Values[m], k), k, SinglePointMSE);
// printf("%f\n", SinglePointMSE);
                                                                RMS_Sum += SinglePointMSE;
// getchar();

                                                        }
// getchar();

// printf("when time = %lu seconds, MSE_Sum = %f\n", k, RMS_Sum);

                                                        if ((k == 3600) || (RMS_Sum < Min_RMS))
        // Tag this time and keep its RMS if it's at a minimum - finds the best fit
        // for the "effective aging time" from making the diffusion couple
                                                        {
                                                                Min_RMS = RMS_Sum;
//                                                              printf("%lf\n", RMS_Sum);
                                                                MinimumTime = k;
                                                        }
                                                        if (RMS_Sum > (1.1 * Min_RMS)) \
                                                         AbortLoopFlag = 1;
                                                }
                                        }

                                }
                                if (l == LastControlSet)
                                {
                                        EffectiveStartingTimes[i][j][l] = MinimumTime;
        // Locks the effective starting time in for this combination of D0, Ac and DataSet
                                }

                                else
        // Use the control dataset's effective aging time to add to the experiment's time
                                {
                                        EffectiveStartingTimes[i][j][l] = \
                                         EffectiveStartingTimes[i][j][LastControlSet];
        // Locks the effective starting time in for this combination
        // of D0, Ac and DataSet.  Uses the control dataset.
                                }
// printf("Minimum Time = %lu seconds.", MinimumTime);
// getchar();
                        }
                if (CurrentData[l].Seconds == 0) printf("\nEnd D0=%f Step\n", Local_D0);
// getchar();
                }
        if (CurrentData[l].Seconds == 0) printf("\nEnd %uC Control Dataset\n", \
         CurrentData[l].Temperature);
// getchar();
        }

        printf("\n\nEnd Starting Time Calculation\n\n");
        for (l=0; l<NumDatasets; l++)
        {
                fprintf(fp, "/=========");
                for (m=0; m<=Ac_Dimension; m++) fprintf(fp, "==========");
                fprintf(fp, "\\\n");
                fprintf(fp, "|         | Dataset %u has %u points at %u degrees C\n", l, \
                 CurrentData[l].NumberOfDataPoints, CurrentData[l].Temperature);
                fprintf(fp, "|=========|");
                for (m=0; m<Ac_Dimension; m++) fprintf(fp, "==========");
                fprintf(fp, "=========|\n");
                fprintf(fp, "| D0 / Ac |");
                for (m=0; m<=Ac_Dimension; m++) fprintf(fp, "  %5.2f  |", Ac_Min + \
                 ((double)m * Ac_Step));
                fprintf(fp, "\n");
                fprintf(fp, "|=========|");
                for (m=0; m<Ac_Dimension; m++) fprintf(fp, "==========");
                fprintf(fp, "=========|\n");
                for (i=0; i<=D0_Dimension; i++)
                {
                        fprintf(fp, "| %6.3f  |", D0_Min + ((double)i * D0_Step));
                        for (j=0; j<=Ac_Dimension; j++)
                        {
                                fprintf(fp, "%8lu |", EffectiveStartingTimes[i][j][l]);
// printf("%lu, ", EffectiveStartingTimes[i][j][l]);
                        }

                        fprintf(fp, "\n");
// printf("\n");
                }

                fprintf(fp, "\\=========");
                for (m=0; m<=Ac_Dimension; m++) fprintf(fp, "==========");
                fprintf(fp, "/\n\n");
// printf("\n\n");
// getchar();

        }
```

```c
        fclose(fp);

        AbsoluteMinimumRMS = 1000;

        for (i=0; i<=D0_Dimension; i++)         // Loop over specified values of D0
        {
                for (j=0; j<=Ac_Dimension; j++) // Loop over specified values of Ac
                {
                        Local_D0 = D0_Min + ((double)i * D0_Step);
                        Local_Ac = Ac_Min + ((double)j * Ac_Step);


                        RMS_Sum = 0;
        // Reset the RMS sum before testing each combination of D0 and Ac
                        RMS_Map[i][j] = 0;

                        for (l=0; l<NumDatasets; l++)
        // Computes the total RMS sum for each combination of D0 and Ac over all datasets.  This
        // makes it so the fit is universal to the whole set of data, and not just to one experiment.
                        {
                                Local_D = DC(Local_D0, Local_Ac, CurrentData[l].Temperature);
        // Compute the diffusion constant for this combination of D0 and Ac

                                if (CurrentData[l].Seconds == 0)
        // Only compute an RMS value for non-control datasets (aged samples only)
                                {
                                        Local_Time = EffectiveStartingTimes[i][j][l];
                                }

                                else
        // Only compute an RMS value for non-control datasets (aged samples only)
                                {
                                        for (m=0; m<CurrentData[l].NumberOfDataPoints; m++)
        // Loop over the individual datapoints
                                        {
                                                RMS_Sum += sqrt(pow((CurrentData[l].Y_Values[m] - \
                                                 PointConc(Local_D, CurrentData[l].X_Values[m], \
                                                 (CurrentData[l].Seconds + Local_Time), l)), 2));
// printf("%f\n", RMS_Sum);
```

```c
                    }
                }
// printf("When D0 = %f and Ac = %f and T = %u, RMS_Sum = %f\n", Local_D0, Local_Ac, CurrentData
[l].Temperature, RMS_Sum);
            }

            RMS_Map[i][j] = RMS_Sum;
    // Locks the total RMS (summed over all datasets) for this combination of D0 and Ac
            if (RMS_Sum < AbsoluteMinimumRMS)
            {
                AbsoluteMinimumRMS = RMS_Sum;
                AbsoluteMinimumX = Local_D0;
                AbsoluteMinimumY = Local_Ac;
                AbsoluteMinimumTime = EffectiveStartingTimes[i][j][0];
            }
        }
// printf("\n");
// getchar();
    }

    fprintf(fp_results, "RMS = %f at D0 = %f and Ac = %f,\nTime = %lu seconds.\n\n", \
     AbsoluteMinimumRMS, AbsoluteMinimumX, AbsoluteMinimumY, AbsoluteMinimumTime);

    for (i=0; i<=D0_Dimension; i++)
    {
        for (j=0; j<=Ac_Dimension; j++)
        {
            fprintf(fp_results, "%f,", RMS_Map[i][j]);
            if (RMS_Map[i][j] < (1.1 * AbsoluteMinimumRMS)) fprintf(fp_gnuplot, \
             "%f %f %f\n", (D0_Min + (i * D0_Step)), (Ac_Min + (j * Ac_Step)), \
             RMS_Map[i][j]);
    // The above line only prints values close to the minimum,
    // used for making the plot more dramatic

// printf("%f, ", RMS_Map[i][j]);
        }

        fprintf(fp_results, "\n");
        fprintf(fp_gnuplot, "\n");
```

```c
// printf("\n");
        }

        fprintf(fp_results, "\n\n\n");
        printf("\n\nFinished\n\n");

        fclose(fp_results);
        fclose(fp_gnuplot);

        return 0;
}

void GetGlobalParameters(void)
{
        system("clear");

        printf("\n\nMike's Diffusion Data Fitting Program");
        printf("\n\nYou have %d datasets for %s in %s", NumDatasets, Element, Alloy);

        printf("\n\nPlease enter the following parameters for the infinite diffusivity (D0)\n");
        printf("\nMinimum value of D0: ");
        scanf("%lf", &D0_Min);
        printf("\nMaximum value of D0: ");
        scanf("%lf", &D0_Max);
        printf("\nStep size for D0: ");
        scanf("%lf", &D0_Step);

        D0_Dimension = ceil((D0_Max - D0_Min) / D0_Step);

        printf("\nMinimum value of Ac: ");
        scanf("%lf", &Ac_Min);
        printf("\nMaximum value of Ac: ");
        scanf("%lf", &Ac_Max);
        printf("\nStep Size for Ac: ");
        scanf("%lf", &Ac_Step);

        Ac_Dimension = ceil((Ac_Max - Ac_Min) / Ac_Step);
}
```

```c
void LoadDataIntoMemory(void)
{
        printf("\n\nLoading files (debug)\n\n");

        FILE *fp = fopen("input-Si-T91-700.csv", "r");
        double TempArray[10];
        unsigned int l, j;
        double dummy;
        unsigned long hours;
        printf("\nStill loading");

        if (fp == 0)
        {
                printf("Could not open input file.  Possible problems:\n\n");
                printf("-Did you make an input file called \"input.csv\"?\n");
                printf("-Is it in the same directory as this program?\n\n");
        }

        else    // Start parsing the input file
        {
                fscanf(fp, "%u,,\n", &NumDatasets);
                fscanf(fp, "%s,,\n", Alloy);
                fscanf(fp, "%s,,\n", Element);
                fscanf(fp, ",,\n");

                for (l=0; l<NumDatasets; l++)
                {
                        fscanf(fp, "%u,%u,%lu", &CurrentData[l].NumberOfDataPoints, \
                         &CurrentData[l].Temperature, &CurrentData[l].Seconds);
                        fscanf(fp, "%lf,%lf,", &CurrentData[l].Cmin, &CurrentData[l].Cmax);
                        CurrentData[l].Cavg = (CurrentData[l].Cmin + CurrentData[l].Cmax) / 2;
                        CurrentData[l].DeltaC = CurrentData[l].Cmax - CurrentData[l].Cmin;

                        for (j=0; j<CurrentData[l].NumberOfDataPoints; j++)
                        {
                                fscanf(fp, "%lf,%lf,\n", &CurrentData[l].X_Values[j], \
                                 &CurrentData[l].Y_Values[j]);
                                CurrentData[l].X_Values[j] /= 10000;
                        }
                        fscanf(fp, ",,\n");
                }

                fclose(fp);

                printf("Your alloy is %s\n", Alloy);
                printf("Your element is %s\n", Element);
                printf("You have %d datasets\n", NumDatasets);

/*              for (l=0; l<NumDatasets; l++)
                {
        // This function displays the file contents to ensure they made it to memory.
        // It's usually only executed in DEBUG mode.
                        hours = CurrentData[l].Seconds / 3600;
                        printf("\n\nThis dataset was run at %d degrees Celsius for %lu hours\n", \
                         CurrentData[l].Temperature, hours);
                        printf("This dataset contains %d points\n", \
                         CurrentData[l].NumberOfDataPoints);
                        for (j=0; j<CurrentData[l].NumberOfDataPoints; j++) printf("%f microns, \
                         %f wt percent Si\n", CurrentData[l].X_Values[j], \
                         CurrentData[l].Y_Values[j]);
                        getchar();
                }       */
        }
}

double DC(double Current_D0, double Current_Ac, unsigned int Current_Temperature)
{
        double Current_D;

        Current_D = Current_D0;
        Current_D *= exp(((-Current_Ac) / (R * (Current_Temperature + 273.15))));

        return Current_D;
}

double PointConc(double Current_D, double Current_X, unsigned long Current_Time, unsigned char l)
{
        double Concentration;

        Concentration = (CurrentData[l].DeltaC / 2);
        Concentration *= erf(Current_X / sqrt(4*Current_D*Current_Time));
        Concentration += CurrentData[l].Cavg;

        return Concentration;
}
```

Example Input File

```
6               Number of Datasets (positive integer)
T91             Name of alloy (string)
Silicon         Name of diffusing species (string)
                Blank line required
31,700,0        (# of datapoints),(Temperature in C),(Aging time in hours)
0.6846,3.8980,  (Min concentration in at. %),(max concentration in at. %),
-244.59,0.67,   (distance in microns),(concentration in atomic percent),
-232.59,0.69,   (distance in microns),(concentration in atomic percent),
-220.59,0.68,   (distance in microns),(concentration in atomic percent),
...                     Continue value pairs here. Note that you need a comma
More Values...          at the end of each pair before the next line. If you are
...                     having trouble, check the Windows/Linux line feed
-10.59,0.92,            character issue (0x0a vs. 0x0d)
-3.59,1.31,
                Datasets separated by blank line
Next dataset... Continue exactly as above, separating datasets by blank lines.
                Note that the data must be presented with the minimum
                concentration on the left. Reverse your data ahead of time.
```