File I/O

The following program reads integer values from the file `input.dat` and copies only the positive values to the file `output.dat`. This program is provided to you as a reference to remind you of the syntax used when working with files.

NOTICE: `fscanf` has similar behaviour to `scanf` but reads from a filestream (`FILE*`) instead of the keyboard

```
/*
 * Author:  Joe Gear
 * Date:    June 20, 2009
 * Purpose: copies only the positive values from file INPUTFILE to file OUTPUTFILE
 */

#include <stdio.h>

#define INPUTFILE "input.dat"
#define OUTPUTFILE "output.dat"

int main( void ) {
    FILE* in_file;
    FILE* out_file;
    char next_val;

    /* open input file and check */
    in_file = fopen( INPUTFILE, "r" );
    if( in_file == NULL ) {
        printf( "Error opening input file - program terminating...\n" );
        return 1;
    }
    /* open output file and check */
    out_file = fopen( OUTPUTFILE, "w" );

    if( out_file == NULL ) {
        printf( "Error opening output file - program terminating...\n" );
        fclose( in_file );  /* close the file we've already opened */
        return 1;
    }

    /* copy positive values from input file to output file */
    while( fscanf( in_file, "%d", &next_val ) == 1 ) {
        if( nextVal > 0 ) {
            fprintf( outFile, "%d\n", nextVal );
        }
    }

    /* close files and finish */
    fclose( in_file );
    fclose( out_file );
    printf( "positive numbers copied from input file to output file.\n" );

    return 0;
}
```

1. Complete the function `count_above` according to the given documentation. Test your function from `main`.

```
#include <stdio.h>

#define INPUTFILE "input.dat"
int count_above();

int main( void ) {
    int count = count_above();
    printf("count: %d\n", count);


    return 0;
}

/*
 * Purpose: prompts user for a threshold value and counts the number of entries
 *    in INPUTFILE that are greater than the given threshold value
 * Parameters: None
 * Returns: int, the count, -1 if error opening INPUTFILE or reading from user
 */
int count_above() {
    FILE* in_file;
    double next_val;
    double threshold_val;
    int count_above = 0;

    in_file = fopen( INPUTFILE, "r" );

    if( in_file == NULL ) {
        printf( "Error opening input file\n" );
        return -1;
    } else {
        printf("enter a threshold value: \n");
        if (scanf("%lf", &threshold_val) != 1) {
            printf( "Error getting input from user\n" );
            return -1;
        }
        while (fscanf(in_file, "%lf", &next_val) == 1) {
            if (next_val > threshold_val) {
                count_above++;
            }
        }
        fclose( in_file );
    }
    return count_above;
}
```

2. In this problem we will assume that we have a file called `yvr_temperatures08.dat` containing the temperature at Vancouver International Airport at midnight, 6am, noon and 6pm on given days of the year for 2008. The meteorologists at the airport attempt to record data every day but technical problems mean that data for some days of the year could go missing. We will write a program that reads the file and reports the average temperature at midnight over the year.

   Assume that the format of the file is as follows:
   - data for each day is on a line of its own
   - the first entry is an integer from 1 to 366 (2008 was a leap year) representing the day of the year
   - the next four entries are doubles representing the temperature at midnight, 6am, noon and 6pm, in that order

   Here's a sample data file:

   ```
   1 2.4 2.3 3.5 5.4
   2 3.6 2.1 5.7 5.0
   4 1.1 1.2 2.3 2.2
   5 2.5 1.3 2.4 2.1
   6 2.0 1.1 2.3 2.0
   …
   366 1.2 0.3 2.4 2.1
   ```

   Note that for this particular file, data was not recorded on day 3 of the year. You can assume that if data is included for a particular day, the temperatures at all four times of the day are provided.

   a) Sketch out an algorithm for solving this problem. Given that reading data from a file is a relatively slow operation (as compared to reading data that's stored in memory) you should develop an algorithm that reads each piece of data from the file only once using `fscanf` to read formatted data.

   b) Design a function called `analyze_temps` that takes a valid `FILE*` as an argument that is a pointer to an open file in the format described above. The function should implement your algorithm from part a and you should call the function from your `main`.

   c) Add to your the program written in part (b) so that the `analyze_temps` function instead of just reporting the average temperature at midnight it also reports the minimum and maximum temperatures recorded at noon over the year. Again, do not attempt to read a particular piece of data from the file more than once, as this is a time consuming operation as compared to reading data stored in main memory.

```c
#include <stdio.h>

#define DATA_PER_LINE 5
#define FILENAME "yvr_temperatures08.dat"

void analyze_temps(FILE* infile);

int main( void ) {
    FILE* infile;
    infile = fopen( FILENAME, "r" );

    if( infile == NULL ) {
        printf( "Error opening input file - program terminating!\n" );
        return 1;
    } else {
        analyze_temps(infile);
        fclose( infile );
        return 0;
    }
}

/*
 * Purpose: analyzes temperature data in FILENAME
 * Parameters: FILE* infile - a pointer to a valid open file
 */
void analyze_temps(FILE* infile) {
    int count = 0;
    int day;
    double six_am, noon, six_pm, midnight;
    double avg, sum, min, max;

    /* repeatedly attempt to scan a line of data from file */
    while( fscanf( infile, "%d %lf %lf %lf %lf",
                   &day, &midnight, &six_am, &noon, &six_pm ) == DATA_PER_LINE ) {

        if (count==0) {
            min = noon;
            max = noon;
        } else {
            if (noon<min) {
                min = noon;
            } else if (noon>max) {
                max = noon;
            }
        }
        sum += midnight;
        count++;
    }
    if (count==0) {
        printf("no readings in the file\n");
    } else {
        avg = sum / count;
        printf( "Average temperature at midnight was: %.1f degrees C \n", avg );
        printf( "Coldest day at noon was: %.1f degrees C\n", min);
        printf( "Warmest day at noon was: %.1f degrees C \n", max);
    }
}
```

4. Assuming the constant OUTPUT_FILE is defined as a symbolic constant, complete the function write_sine_table below that takes a positive integer N. The function should write to the file a table of sine values for each of the following values: $\pi/N,\ 2\pi/N,\ 3\pi/N,\ ...,\ \pi$.
   For example, if the function is called with an *N* of 5, it should write the following table of values to the file sineTable.dat:

```
     x   sin(x)
 0.628  0.5878
 1.257  0.9511
 1.885  0.9511
 2.513  0.5878
 3.142  0.0000
```

   Your table of values should be formatted exactly as shown here.
   Hint: be sure to use a variable of type int to control the loop. Recall that computation with doubles is not exact so, in general, we should avoid using such variables to control a loop as we may end up executing the loop one too many or one too few times.
   You can use the sin function in math.h …

```
double sin(double x)
The sin() function returns the sine of an argument (angle in radians).
```

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

#define OUTPUT_FILE "sine_table.dat"
#define PI acos(-1.0)

void write_sine_table(int n);

int main( void ) {
    write_sine_table(5);

    return 0;
}

/*
 * Purpose: write a table of sine values with n rows to OUTPUT_FILE
 * Parameters: int n - number of rows in the table
 */
void write_sine_table(int n) {
    double x, sin_x;
    int count;
    FILE *outfile;

    /* open file for writing */
    outfile = fopen(OUTPUT_FILE, "w");

    if (outfile == NULL) {
        printf("Error opening output file -- program terminating\n");
    } else {
        /* print headings */
        fprintf( outfile, "    x  sin(x)\n" );

        /* compute and then write the N sine values to the file */
        for(count=1; count<=n; count++) {
            x = count * PI / n;
            sin_x = sin( count * PI / n );
            fprintf( outfile, "%.3f %.4f\n", x, sin_x );
        }
        fclose( outfile );
    }
}
```