7.3. Project File LEW2.CPP

This file contains the code that reads and checks the input data and prints the output files.

Includes:

STDIO.H
STDLIB.H

Defines:

MAXLAYERS - the maximum number of reflecting layers (or reflected rays seen by the receiver) in the ionosphere that the program will handle.
DATA - the number of real data points in the output data streams. Two successive data points represent a complex number. The first is the real part and the second is the imaginary part.

Structures:

ray_path - structure that contains all input and computed variables characteristic of a path. The elements of **ray_path** are given on p. 28.
compute - structure that contains all the variables specific to the computations or not specific to an individual path. The elements of **compute** are given on p. 29.

String type:

STRING - used for handling file names of input and output files.

Files:

innyfile - **pointer** to the input file.
datyfile - **pointer** to the first output file. This file will contain all the input and computed parameters.
bigfile - **pointer** to the second output file. This file will contain the transfer function.

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

#define MAXLAYERS 3
#define DATA 4096

typedef struct ray_path
{
        float path_Distance, center_freq, penetrate_freq, thick_scale, maxD_hgt;
        float peak_amplitude, sigma_tau, sigma_c, sigma_D, fds, fdl;
        double tau_c, sigma_f, slp, tau_L, tau_U, tau_l, alpha, sigma_l, lambda;
};

typedef struct compute
{
        int layers, slices, seed;
        float delth, afl;
        double delt, big_el;
};

typedef char *STRING;

FILE *innyfile, *datyfile, *bigfile;
```

7.3.1. Function **void init**

Description:

This function is called by **main**. **Init** opens the input file and calls **input_data** to read all the data into the variables and to check the data for compatibility with the model and for preventing division by zero, logarithms of nonpositive numbers, etc. Note that *delta_t* is multiplied by 10^{-6}. This is done to place *delta_t* in the proper units of seconds. For consistency, all times are input in microseconds. **Init** finishes by closing the input file. **Init** is in the file LEW2.CPP.

Variables passed to **init**:

arg_num - **integer**, the number of arguments in the command line, 4.
inny - **STRING**, the name of the input file.
ci - **compute**, structure containing the computation parameters.
pi - **ray_path**, structure containing the path parameters.

Functions called:

input_data - reads data from the input file and checks some input data. **Init** passes *ci*, a single element array of **compute**, and *pi*, an array of **ray_path**, by reference. **Input_data** is in LEW2.CPP.
exit - termination library function, requires STDLIB.H.
fopen - library function that opens files, requires STDIO.H.
fclose - library function that closes files, requires STDIO.H.

```
void init(int arg_num, STRING inny, struct compute ci[1], struct ray_path pi[MAXLAYERS])
{
                /* Function Prototype */

        void input_data(compute[], ray_path[]);

                /* Code */

        if (arg_num !=4)
        {
                printf("\n Error in function init! \n");
                printf("\n Is command line correct?: lews infile outfile1 outfile2 \n");
                printf("\n Program will terminate! \n");
                exit(0);
        }

        if ((innyfile = fopen(inny,"r")) == NULL)
        {
                printf("\n Error in function init! \n");
                printf("\n Input file cannot be opened! \n");
                printf("\n Terminating program! \n");
                exit(0);
        }

        input_data(ci, pi);

        ci[0].delta_t *= 1.0E-6;

        if (fclose(innyfile) == EOF)
        {
                printf("\n Error in function init! \n");
                printf("\n Cannot close the input file! \n");
                printf("\n Terminating program \n");
                exit(0);
        }

        return;

}  /* End of init */
```

7.3.2. Function void **input_data**

Description:

Input_data reads data from the file specified by the second argument on the command line. This function also checks input data values to prevent data that would violate the model and that would cause run time errors due to division by zero, logarithms of nonpositive numbers, square roots of negative numbers, etc. **Input_data** also checks that the pseudorandom number generator seed is in the proper range. The input file has already been opened by **init** and the input file name is a global variable.

Variables passed to **input_data**:

cii - **compute**, structure containing the computation parameters, an array of length 1.
pii - **ray_path**, structure containing the path parameters, an array of length MAXLAYERS.

Local variable:

j - **integer**, used to count through the layers of the input data file.

Functions called:

fscanf - library function reads from files, requires STDIO.H.
printf - library function prints to the executing window, requires STDIO.H.
exit - library termination function, requires STDLIB.H.

```
void input_data(struct compute cii[1], struct ray_path pii[MAXLAYERS])
{
                /* Variable */

        int j;

                /* Code */

        fscanf(innyfile, "%d%f%f%d%d", &cii[0].slices, &cii[0].delta_t, &cii[0].afl,
                &cii[0].layers, &cii[0].seed);

        for (j = 0; j < cii[0].layers; j++)
        {
                fscanf(innyfile, "%f%f%f%f%f%f%f%f%f%f%f", &pii[j].path_Distance,
                        &pii[j].center_freq, &pii[j].penetrate_freq, &pii[j].thick_scale,
                        &pii[j].maxD_hgt, &pii[j].peak_amplitude, &pii[j].sigma_tau,
                        &pii[j].sigma_c, &pii[j].sigma_D, &pii[j].fds, &pii[j].fdl);

                /* Input data checking */

                if (pii[j].peak_amplitude == 0.0)
                {
                        printf("\n Error in function input_data!");
                        printf("\n Division by zero coming!");
                        printf("\n Peak_amplitude, A, must be greater than 0!");
                        printf("\n Program will terminate!");
                        printf("\n Correct the input file!");
                        exit(0);
                }

                if ((pii[j].sigma_c == 0.0) || (pii[j].sigma_c >= (pii[j].sigma_tau / 2)))
                {
                        printf("\n Error in function input_data!");
                        printf("\n Division by zero warning!");
                        printf("\n Sigma_c must be greater than 0 and less than half sigma_tau!");
                        printf("\n Program will terminate!");
                        printf("\n Correct the input file!");
                        exit(0);
                }
                if ((cii[0].afl <= 0.0) || (cii[0].afl >= 1.0))
                {
                        printf("\n Error in function input_data!");
                        printf("\n Square root of a negative number warning!");
                        printf("\n Afl must be between 0 and 1!");
                        printf("\n Program will terminate!");
                        printf("\n Correct the input file!");
                        exit(0);
                }

                if (pii[j].penetrate_freq <= pii[j].center_freq)
                {
                        printf("\n Error in function input_data!");
                        printf("\n Penetration frequency must be greater than the");
                        printf(" center frequency!");
                        printf("\n Program will terminate!");
                        printf("\n Correct the input file!");
                        exit(0);
                }

                if ((cii[0].seed < 1) || (cii[0].seed > 30268))
                {
                        printf("\n Error in function input_data!");
                        printf("\n The seed must be between 1");
                        printf("\n and 30268 inclusive!");
                        printf("\n Program will terminate!");
                        printf("\n Correct the input file!");
                        exit(0);
                }

        }

        return;

}  /* End of input_data */
```

7.3.3. Function **void out1**

Description:

This function is called by **comp_arrays** and prints the input and calculated parameters for the computing run and all input and computed parameters for each layer to the file specified by the third argument on the command line. Essentially, **out1** prints out all of the elements of the arrays *cdco* and *pdco*. **Out1** is in file LEW2.CPP.

Variables passed to **out1**:

cdco - structure of type **compute**.
pdco - structure of type **ray_path**.
daout1 - **STRING**, output file name.

Local variable:

i - **integer**, counts through the number of layers.

Functions called:

fopen - library function, opens file for printing, requires STDIO.H.
printf - library function, prints to screen, requires STDIO.H.
fprintf - library function, prints to file, requires STDIO.H.
fclose - library function, closes file, requires STDIO.H.

```
void out1(struct compute cdco[1], struct ray_path pdco[MAXLAYERS], STRING daout1)
{
        /* Variable */

        int i;

        /* Code */

        if ((datyfile = fopen(daout1,"w")) == NULL)
        {
                printf("\n Error in function out1!");
                printf("\n First output file cannot be opened!");
                printf("\n Terminating Program! \n");
                exit(0);
        }

        fprintf(datyfile,"\n Computing Parameters \n");
        fprintf(datyfile,"\nInput parameters\n");
        fprintf(datyfile,"\n slices = %d", cdco[0].slices);
        fprintf(datyfile,"\n delta_t = %f", cdco[0].delta_t);
        fprintf(datyfile,"\n afl = %f", cdco[0].afl);
        fprintf(datyfile,"\n layers = %d", cdco[0].layers);
        fprintf(datyfile,"\n seed = %d", cdco[0].seed);
        fprintf(datyfile,"\n\nComputed parameter\n");
        fprintf(datyfile,"\n delta_tau = %lf", cdco[0].delta_tau);
        fprintf(datyfile,"\n big_el = %lf", cdco[0].big_el);
        fprintf(datyfile,"\n\n Individual Path Data \n");

        for (i = 0; i < cdco[0].layers; i++)
        {
                fprintf(datyfile,"\n Layer %d \n", i + 1);
                fprintf(datyfile,"\n Input parameters \n");
                fprintf(datyfile,"\n path distance = %f", pdco[i].path_Distance);
                fprintf(datyfile,"\n center frequency = %f", pdco[i].center_freq);
                fprintf(datyfile,"\n penetration frequency = %f", pdco[i].penetrate_freq);
                fprintf(datyfile,"\n Thickness scale factor = %f", pdco[i].thick_scale);
                fprintf(datyfile,"\n Height of the maximum density = %f", pdco[i].maxD_hgt);
                fprintf(datyfile,"\n peak amplitude = %f", pdco[i].peak_amplitude);
                fprintf(datyfile,"\n sigma_tau = %f", pdco[i].sigma_tau);
                fprintf(datyfile,"\n sigma_c = %f", pdco[i].sigma_c);
                fprintf(datyfile,"\n sigma_D = %f", pdco[i].sigma_D);
                fprintf(datyfile,"\n fds = %f", pdco[i].fds);
                fprintf(datyfile,"\n fdl = %f \n", pdco[i].fdl);
                fprintf(datyfile,"\n Computed parameters \n");
                fprintf(datyfile,"\n tau_c = %lf", pdco[i].tau_c);
                fprintf(datyfile,"\n sigma_f = %lf", pdco[i].sigma_f);
                fprintf(datyfile,"\n slp = %lf", pdco[i].slp);
                fprintf(datyfile,"\n tau_L = %lf", pdco[i].tau_L);
                fprintf(datyfile,"\n tau_U = %lf", pdco[i].tau_U);
                fprintf(datyfile,"\n tau_l = %lf", pdco[i].tau_l);
                fprintf(datyfile,"\n alpha = %lf", pdco[i].alpha);
                fprintf(datyfile,"\n sigma_l = %lf", pdco[i].sigma_l);
                fprintf(datyfile,"\n lambda = %lf\n", pdco[i].lambda);

        }  /* End of  i loop. */

        if (fclose(datyfile) == EOF)
        {
                printf("\n Error in function out1!");
                printf("\n Cannot close the first output file!");
                printf("\n Terminating program! \n");
                exit(0);
        }

}  /* End of out1 */
```

7.3.4. Function **void outit**

Description:

This function prints the complex coefficients of the transfer function to the file specified by the fourth argument in the command line. The complex numbers are represented by a list of floats that are successively real and imaginary. The white space separator is the space character. **Outit** appends the list of complex coefficients for each time slice to the same file. Separate time slices are separated by an extra line. **Outit** is in file LEW2.CPP.

Parameters passed to **outit**:

dat - array of **float**, the complex number array.
daout2 - **STRING**, the large file name.

Local variables:

q - **integer**, counts through *dat*.

Functions called:

fopen - library function, opens file for printing, in this case for appending to file, requires STDIO.H.
printf - library function, prints to screen, requires STDIO.H.
fprintf - library function, prints to file, requires STDIO.H.
fclose - library function, closes file, requires STDIO.H.

```
void outit(float dat[2 * DATA], STRING daout2)
{
                /* Variable */

        int q;

                /* Code */

        if ((bigfile = fopen(daout2,"a")) == NULL)
        {
                printf("\n Error in function outit! \n");
                printf("\n Second output file cannot be opened! \n");
                printf("\n Terminating program! \n");
                exit (0);
        }

        for (q = 0; q < 2 * DATA; q++)
                fprintf(bigfile,"%lf ", dat[q]);

        fprintf(bigfile,"\n ");

        if (fclose(bigfile) == EOF)
        {
                printf("\n Error in function outit! \n");
                printf("\n Cannot close the second output file! \n");
                printf("\n Terminating program! \n");
                exit(0);
        }

} /* End of outit */
```