WVU CS 350 – Assignment 2

Documentation and Description of Algorithms

Paul Prince – Spring Semester, 2010

Table of Contents

Index of Partial Code Listings

Index of Illustrations

Question 1: Mini-Database in C

For this question, we are asked to write a simple database program that will help you keep track of your tools. For each tool we track a unique ID number, the name of the tool, a quantity, and a cost. The program is required to present a menu interface to the user that supports adding new records, deleting records, reading and writing database files, searching for tools by ID or name, etc.

It was not clear to me whether it was intended for us to keep the entire database in memory, only writing it to a file at the user's request, or to keep the database on disk, and use random-access file operations on it. I have elected to implement an on-disk database, where only a single record is kept in memory at a time.

Database File Format

The tool_t structure represents a single tool, and has fields for each of the required properties. In addition, a special tool instance represents an empty record:

```
/* The tool data structure. */
typedef struct {
        int id;
        char name[MAX_LEN_TOOLNAME+1];
        int qty;
        int cost; /* in cents. */
} tool_t;
tool_t EMPTY_TOOL = { -1, "", 0, 0 };
```

Listing 1: The tool_t struct and EMPTY_TOOL

Note that we store the cost as an integer number of cents, rather than a floating-point number of dollars.[1]

On disk, the database file format is simply this structure naïvely written out `MAX_TOOLS` times (the value of `MAX_TOOLS` is 100 in my submitted code, in keeping with the assignment.) The length of the database file is always the same, `(MAX_TOOLS * sizeof(tool_t))`.

With this format, we can randomly access records in the database file, or we can read/write them sequentially.

For an example of randomly accessing the database, let's view the `write_tool()` function, which in an Object-Oriented environment would probably be called `Tool.save()` :

```
void write_tool(tool_t tool) {
        if(fseek(active_database, tool.id*sizeof(tool_t), SEEK_SET) != 0){
                printf("FATAL: Error writing tool to active database.\n");
                exit(99);
        }
        if(fwrite(&tool, sizeof(tool_t), 1, active_database) != 1){
                printf("FATAL: Error writing tool to active database.\n");
                exit(98);
        }
}
```

Listing 2: The write_tool() function

1 This is best practice for representing money, but unfortunately I did not find time to hide this implementation detail from the user, and thus when you run the program, you will be prompted for prices in cents, and prices will be printed to the screen in cents, rather than in dollars.

And for an example of sequential access, let's take a look at `print_db()` :

```
void print_db(){
        rewind(active_database);
        int i;
        tool_t tool;
        printf("\n");
        printf("  PRINTING TOOL DATABASE:\n");
        printf("  ======================:\n");
        printf("\n");
        printf("+------+--------+--------+--------------\n");
        printf("|  ID  |  Qty   |  Cost  | Name   ...\n");
        printf("+======+========+========+==============\n");
        for(i=0; i<MAX_TOOLS; i++){
                if(fread(&tool, sizeof(tool_t), 1, active_database) != 1){
                        printf("FATAL: Error reading from database.\n");
                        exit(89);
                }
                if(tool.id >= 0 && tool.id < MAX_TOOLS){
                        printf("|%6d|%8d|%8d|%s\n", tool.id, tool.qty, tool.cost,
tool.name);
                }
        }
        printf("+------+--------+--------+--------------\n");
        printf("\n");
}
```

Listing 3: The print_db() function

It is worth noting that reading and writing C structures like this is entirely non-portable. Indeed, a database file I created at home very likely may not be compatible with the version of the executable built for grading.

Loading and Saving

Since I decided to implement my database on-disk, and not in-memory, I do not provide a "Save Database" function in my main menu. Instead, as you use the program interface to modify the database, your changes are written to disk immediately.

When first starting the program, you will need to initialize an on-disk database (or load a previously-initialized one) before you can begin adding records.

I use a couple of global variables and the `change_active()` function to keep track of which, if any, data file is currently open:

```
/* Global variables */
FILE *active_database;
char active_path[MAX_LEN_PATH] = "";

void change_active(FILE *new_active, const char* new_path){
        if (active_database) {
                if(fclose(active_database) != 0) {
                        perror("FATAL: Error while closing previously-active
database.\n");
                        exit(201);
                }
        }
        active_database = new_active;

        if (active_database) rewind(active_database);

        strncpy(active_path, new_path, MAX_LEN_PATH-1);
        active_path[MAX_LEN_PATH-1] = '\0';
}
```

Listing 4: The change_active() function and its global variables

Searching

One of the requirements of the assignment is to implement searching the database, by both tool ID and tool name. My program implements both functions, accessible from the "Search" option on the main menu.

Searching by Tool ID

For searching by ID, we can use seek directly to the proper location in the database file (random access), and determine if a record is stored there (instead of the `EMPTY_TOOL` placeholder.):

```
void print_search_results_by_id(int id) {
        tool_t tool;
        int total_results = 0;
    ...
        if(fseek(active_database, id*sizeof(tool_t), SEEK_SET) != 0){
                printf("FATAL: Error writing tool to active database.\n");
                exit(29);
        }

        if(fread(&tool, sizeof(tool_t), 1, active_database) != 1){
                printf("FATAL: Error reading from database.\n");
                exit(28);
        }

        if(tool.id == id){
                printf("|%6d|%8d|%8d|%s\n", tool.id, tool.qty, tool.cost, tool.name);
                total_results++;
        }
    ...
}
```

Listing 5: The print_search_results_by_id() function

Searching by Tool Name

For searching by tool name, I do a brute-force linear search, checking each record to see if the tool name matches the query. I decided to implement a "starts-with" search, so that all tools with tool names starting with the user's query will be returned as results. For example, if you search for "Wire", you may get results for "Wire Cutters", "Wire Tie Tool", and "Wired Receptacle". The `strncmp()` standard library function makes this trivially easy to implement:

```
void print_search_results_by_name(const char *starts_with) {
        tool_t tool;
        int i;
        int total_results = 0;
    ...
        rewind(active_database);
        for(i=0; i<MAX_TOOLS; i++){
                if(fread(&tool, sizeof(tool_t), 1, active_database) != 1){
                        printf("FATAL: Error reading from database.\n");
                        exit(89);
                }

                if(tool.id >= 0 && tool.id < MAX_TOOLS){
                        if(strncmp(tool.name, starts_with, strlen(starts_with)) == 0){
                                printf("|%6d|%8d|%8d|%s\n", tool.id, tool.qty,
                                                                tool.cost, tool.name);
                                total_results++;
                        }
                }
        }
    ...
}
```

Listing 6: The print_search_results_by_name() function

Question 2: Adaptive Image Enhancement

For this question, we were asked to write a program that reads Portable Bitmap image files, applies certain transformations to produce a number of output images, and that also prints out certain statistics concerning the input and output images.

This one was a lot of fun to write. Although it seemed very difficult at first, I quickly realized that all of the information required to complete the assignment was provided in the problem description.

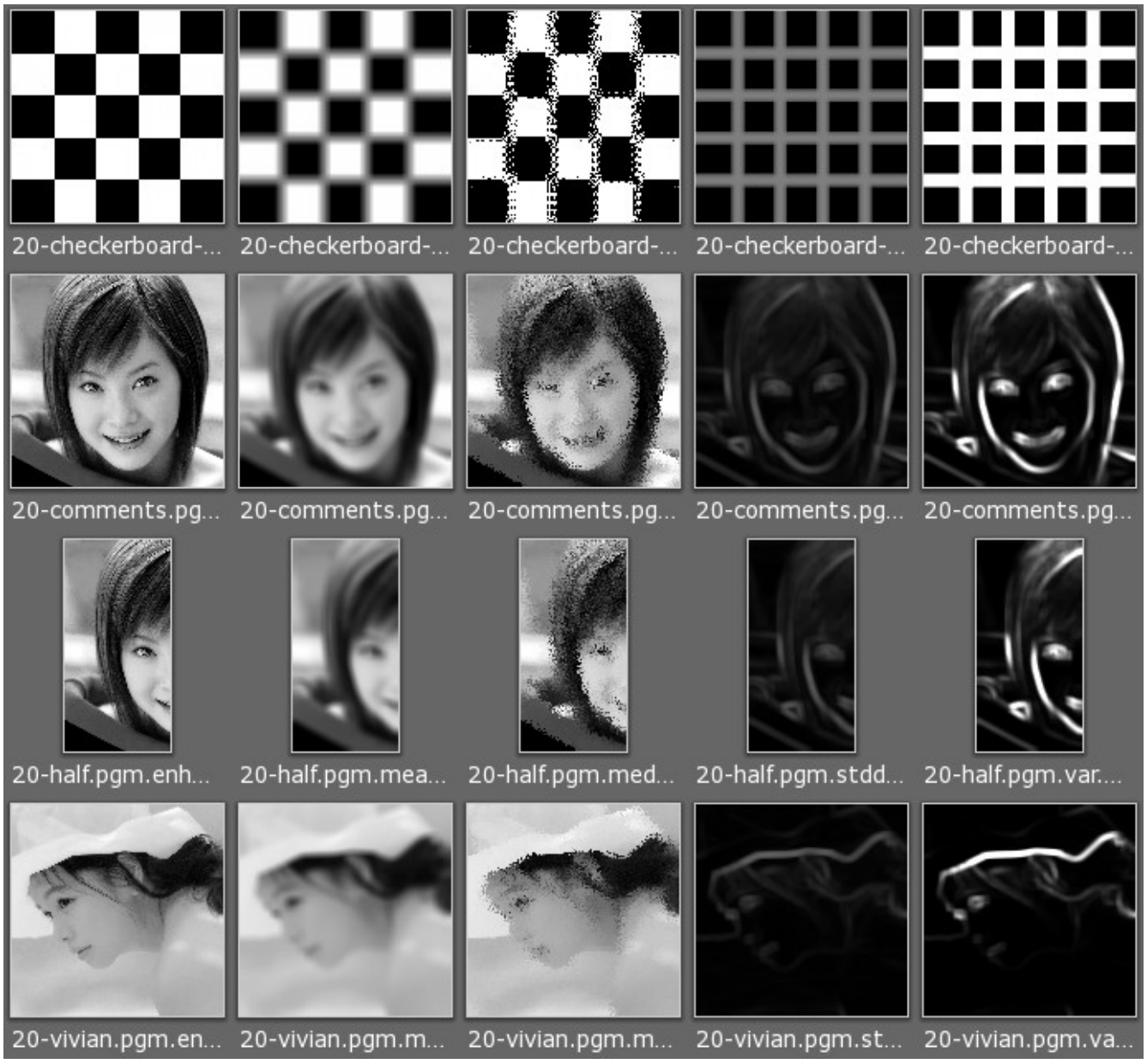

Illustration 1: The output from an early version of my program

Window Selection Algorithm

As soon as the instructor began discussing this question in class, and began to talk about partitioning an input image into overlapping NxN blocks, I realized that the image edges would present a problem. As the discussion continued, several methods for handling the edges were proposed, including cropping the output image to remove the edges, or ignoring the edge pixels entirely.

I was determined, however, to handle the image edges correctly. The solution was quite simple, and actually made the code simpler (since I only have to deal with a single image size in any given run of the program).

My idea is to iterate over the pixels in the input image, and for each one, select an NxN square block of pixels around it. For edge pixels, this window will overlap the image edge; in that case, I truncate off the parts of the window that fall outside the window boundary.

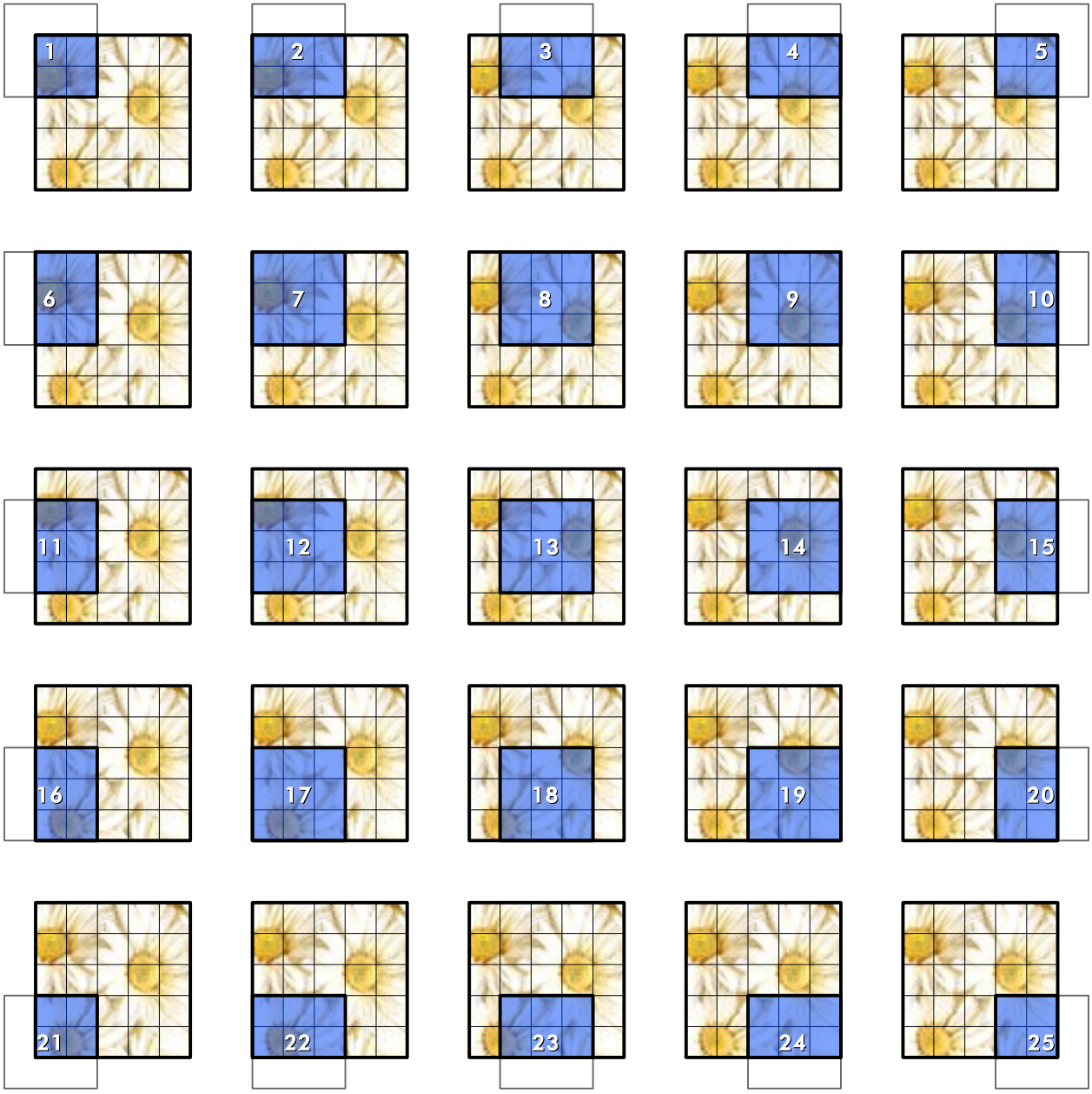

Illustration 2: The 25 3x3-pixel windows selected from a 5x5-pixel image

The implication is that although a 3x3 square window may be requested, smaller windows may be selected instead, and thus the functions that operate on the windows must be able to handle arbitrary rectangular window sizes.

This is the code that selects a valid window, given a center point in a larger rectangular region, and the ideal length of the window's sides:

```
typedef struct {
        int upper_left_row;
        int upper_left_col;
        int lower_right_row;
        int lower_right_col;
} window_t;

window_t select_window(int size, int row, int col, int max_rows, int max_cols) {
       window_t window;
       window.upper_left_row  = row - size/2;
       window.upper_left_col  = col - size/2;
       window.lower_right_row = row + size/2;
       window.lower_right_col = col + size/2;
       if ( window.upper_left_row < 0 ) window.upper_left_row = 0;
       if ( window.upper_left_col < 0 ) window.upper_left_col = 0;
       if ( window.lower_right_row > max_rows-1 ) window.lower_right_row = max_rows-1;
       if ( window.lower_right_col > max_cols-1 ) window.lower_right_col = max_cols-1;
       return window;
}
```

Listing 7: The window_t type and the select_window() function

Not very difficult at all, given a little thought!

Calculating Window Means and Standard Deviations

This is my function for calculating the mean and variance of a rectangular window (the standard deviation is the square-root of the variance):

```
/* Calculate the mean and variance of a rectangular region. */
void windowcalc_mean_and_variance(image_ptr image, int image_width, window_t window,
                                  double* result_mean, double* result_variance) {

        unsigned long int n = 0;       /* Pixel count */
        double mean  = 0.0;
        double delta = 0.0;
        double M2    = 0.0;

        int i, j;
        for ( i=window.upper_left_row; i<=window.lower_right_row; i++ ) {
                for ( j=window.upper_left_col; j<=window.lower_right_col; j++ ) {
                        n++;
                        unsigned int pixel_value = image[i*image_width + j];
                        delta = pixel_value - mean;
                        mean = mean + delta / n;
                        M2 = M2 + delta * (pixel_value - mean);
                }
        }
        double variance = M2/(n - 1);

        /* Return results via output arguments. */
        if (result_mean != NULL) *result_mean = mean;
        if (result_variance != NULL) *result_variance = variance;
}
```

Listing 8: The windowcalc_mean_and_variance() function

This is based on an algorithm for calculating variance by Donald Knuth (*The Art of Computer Programming*, volume 2: *Seminumerical Algorithms*, 3rd edn., p. 232. Boston: Addison-Wesley.), who cites work by B.P. Welford. An interesting and handy aspect of this particularly algorithm is that it requires only a single pass (loop) over its input dataset.

I adapted my C version from the following reference implementation, which I believe is written in Python:

```
def online_variance(data):
    n = 0
    mean = 0
    M2 = 0

    for x in data:
        n = n + 1
        delta = x - mean
        mean = mean + delta/n
        M2 = M2 + delta*(x - mean)  # This expression uses the new value of mean

    variance_n = M2/n
    variance = M2/(n - 1)
    return variance
```

Listing 9: Knuth's algorithm for calculating variance

Since Knuth's algorithm calculates the mean as a side-effect of calculating the variance, I decided to let this one function calculate both values, and return them via output arguments.[2]

2 The result of this is that later, when I needed to merely find the mean (and not variance or standard deviation), I ended up reusing this rather expensive function for that purpose, rather than a much simpler one for finding only the mean.

Calculating Window Medians

I did some searching and found a number of efficient algorithms for calculating medians. However, I decided that simply using the standard library's implementation of Quicksort (and then picking the middle value) would be sufficient:

```
/* Calculate the median of a rectangular region. */
void windowcalc_median(image_ptr image, int image_width, window_t window, double*
result_median) {
        int rows = window.lower_right_row - window.upper_left_row + 1;
        int cols = window.lower_right_col - window.upper_left_col + 1;

        /* Copy the window into our own image object,
           because our median algo modifies the array it operates on. */
        image_ptr image_sorted = (image_ptr) malloc(rows*cols*sizeof(unsigned char));
        if (image_sorted == NULL) {
                fprintf(stderr, "Unable to allocate memory.\n");
                exit(253);
        }

        int i, j;
        int n = 0;
        for ( i=window.upper_left_row; i<=window.lower_right_row; i++ ) {
                for ( j=window.upper_left_col; j<=window.lower_right_col; j++ ) {
                        image_sorted[n] = image[i*image_width+j];
                        n++;
                }
        }

        /* Use the standard-library Quicksort to find the median. */
        qsort(image_sorted, rows*cols, sizeof(unsigned char), int_compare);
        double median = image_sorted[ rows*cols/2 ];
        if ( (rows*cols) % 2 == 0 ) {
                median = (median + image_sorted[ rows*cols/2-1 ])/2;
        }
        free(image_sorted);

        /* Return results via output arguments. */
        if (result_median != NULL) *result_median = median;
        return; /* void */
}
```

Listing 10: The windowcalc_median() function

To keep the API consistent among all of the "windowcalc" functions, I return the result here via output arguments, even though it is only a single value.

Even using this quasi-brute-force approach, I find the execution time of the program to be very acceptable, however after profiling with the `gprof` tool, it is apparent that there would be much to gain from optimizing the median-finding code, as well as the code for computing the enhanced value of the image. (The variance-finding code does not contribute significantly to the run time.)

Calculating Statistics for Whole Images

In addition to finding means, medians, and variances for small regions of the image, we are required to do some calculations on entire images.

After listing the various images and statistics my program would be required to produce, I decided that it would be advisable to reuse the same code for calculating mean, median, and variance on both images *and* subregions.

This is accomplished firstly by defining a special `window_t` object, representing a window that overlays the image exactly. Using this special window, it is very easy to perform calculations on the entire image:

```
        /* Calculate overall statistics for the original image. */
         /* Note: This special-valued window represents the entire image. Use with care. */
         window_t ENTIRE_IMAGE = { 0, 0, rows-1, cols-1 };
         double mean, variance, median, stddev;
         windowcalc_mean_and_variance(Image, cols, ENTIRE_IMAGE, &mean, &variance);
         windowcalc_median(Image, cols, ENTIRE_IMAGE, &median);
         stddev = sqrt(variance);
```

Listing 11: Definition of ENTIRE_WINDOW and example of its use

Calculating the Enhanced Value

This function follows directly from the assignment, and required very little in the way of design:

```
/* Calculate the enhanced value for a pixel given the other required terms. */
int calc_enhanced(unsigned char input_pixel, double overall_mean, double overall_stddev,
                  double window_mean, double window_stddev ){

        /* Constants for the assignment-provided transformation. */
        double A       = 2.00;
        double C1      = 0.40;
        double C2      = 0.02;
        double C3      = 0.40;

        /* The actual transformation:  Selective brightening. */
        if(
                (window_mean <= overall_mean * C1) &&
                (overall_stddev * C2 <= window_stddev) &&
                (overall_stddev * C3 <= window_stddev)
        ){
                return A * input_pixel;
        } else {
                return input_pixel;
        }
}
```

When I first wrote the code, I was doing this calculation directly in `main()`, which worked fine. However, I had to break the enhancement out into its own function in order to measure its performance with `gprof`:

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name

 46.03      2.27     2.27                             int_compare
 36.29      4.07     1.79   262148     0.01     0.01  windowcalc_mean_and_variance
  8.72      4.50     0.43        4   107.67   107.67  write_pnm
  8.41      4.91     0.42   262146     0.00     0.00  windowcalc_median
  0.41      4.93     0.02   262144     0.00     0.00  calc_enhanced
  0.30      4.95     0.02   262144     0.00     0.00  select_window
  0.00      4.95     0.00        3     0.00     0.00  getnum
  0.00      4.95     0.00        1     0.00     0.00  read_pnm
```

Complete Source Code Listings

(next pages…)

minidb.c

```c
/* Includes */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

/* Constants */
#define MAX_TOOLS        100
#define MAX_LEN_PATH     512
#define MAX_LEN_TOOLNAME 255

/* Global variables */
FILE *active_database;
char active_path[MAX_LEN_PATH] = "";

/* The tool data structure. */
typedef struct {
        int id;
        char name[MAX_LEN_TOOLNAME+1];
        int qty;
        int cost; /* in cents. */
} tool_t;
tool_t EMPTY_TOOL = { -1, "", 0, 0 };

/* Prototypes: Functions called from the main menu loop: */
void select_db();
void initialize_db();
void add_update_db();
void print_db();
void delete_from_db();
void search_db();
void print_search_results_by_id(int id);
void print_search_results_by_name(const char *starts_with);

/* Prototypes: other functions. */
void change_active(FILE *new_active, const char *new_path);
void write_tool(tool_t tool);
void prompt_str(const char *prompt_string, char *output_string, int max_len);
void chomp(char *str);


/* main() */
int main(void){

        for(;;){
```

```
        printf("\n");
        printf(" ================================== MENU ==================================\n");
        printf("\n");

        if(active_database == NULL) {
                printf("  ******************************************\n");
                printf("  * No database is currently active.       *\n");
                printf("  * You will need to initialize or load one. *\n");
                printf("  ******************************************\n");
        } else {
                printf("  +++++\n");
                printf("  + Database OPEN and ACTIVE!\n");
                printf("  +   DB Path: \"%s\"\n", active_path);
        }

        printf("\n");
        printf("    I - Initialize a database\n");
        printf("    N - Add/Update records\n");
        printf("    S - Search for a record\n");
        printf("    D - Delete a record\n");
        printf("    P - List all records\n");
        printf("    C - Close current database\n");
        printf("    L - Load an existing database\n");
        printf("    Q - Exit\n");
        printf("\n");

        char choice[255];
        prompt_str("Choice?", choice, 255);

        switch(*choice){

                /* These menu options do NOT require an active database. */

                case 'L': case 'l':
                        select_db();
                        continue;

                case 'I': case 'i':
                        initialize_db();
                        continue;

                case 'Q': case 'q':
                case 'E': case 'e':
                case 'X': case 'x':
                        printf("Bye!\n");
                        exit(0); /* with much success! */

                case 'N': case 'n':
```

```
                case 'A': case 'a':
                case 'U': case 'u':
                case 'P': case 'p':
                case 'C': case 'c':
                case 'D': case 'd':
                case 'S': case 's':
                        /* Do nothing yet.  These are handled by the next switch,
                         * after checking to make sure a database file is active. */
                        break;

                default:
                        printf("Invalid selection, try again..\n");
                        continue;
        }

        if (!active_database) {
                printf(" !! No active database.\n");
                printf(" !! Select or initialize one first.\n");
                continue;
        }

        switch(*choice){

                /* These menu options DO require an active database. */

                case 'N': case 'n':
                case 'A': case 'a':
                case 'U': case 'u':
                        add_update_db();
                        continue;

                case 'P': case 'p':
                        print_db();
                        continue;

                case 'C': case 'c':
                        change_active(NULL, "");
                        continue;

                case 'D': case 'd':
                        delete_from_db();
                        continue;

                case 'S': case 's':
                        search_db();
                        continue;

                default:
```

```
                                printf("Invalid selection, try again..\n");
                                continue;
                        }
                }
}


/*******************/
/* OTHER FUNCTIONS */
/*******************/

void print_db(){
        rewind(active_database);
        int i;
        tool_t tool;
        printf("\n");
        printf("  PRINTING TOOL DATABASE:\n");
        printf("  =======================:\n");
        printf("\n");
        printf("+------+--------+--------+---------------\n");
        printf("|  ID  |  Qty   |  Cost  | Name   ...\n");
        printf("+======+========+========+===============\n");
        for(i=0; i<MAX_TOOLS; i++){
                if(fread(&tool, sizeof(tool_t), 1, active_database) != 1){
                        printf("FATAL: Error reading from database.\n");
                        exit(89);
                }
                if(tool.id >= 0 && tool.id < MAX_TOOLS){
                        printf("|%6d|%8d|%8d|%s\n", tool.id, tool.qty, tool.cost, tool.name);
                }
        }
        printf("+------+--------+--------+---------------\n");
        printf("\n");
}


void print_search_results_by_id(int id) {
        tool_t tool;
        int total_results = 0;

        printf("\n");
        printf("  SEARCH RESULTS:\n");
        printf("  =======================:\n");
        printf("  Query was:\n");
        printf("    ID = %d\n", id);
        printf("\n");
        printf("+------+--------+--------+---------------\n");
        printf("|  ID  |  Qty   |  Cost  | Name   ...\n");
```

```
        printf("+======+========+========+===============\n");

        if(fseek(active_database, id*sizeof(tool_t), SEEK_SET) != 0){
                printf("FATAL: Error writing tool to active database.\n");
                exit(29);
        }

        if(fread(&tool, sizeof(tool_t), 1, active_database) != 1){
                printf("FATAL: Error reading from database.\n");
                exit(28);
        }

        if(tool.id == id){
                printf("|%6d|%8d|%8d|%s\n", tool.id, tool.qty, tool.cost, tool.name);
                total_results++;
        }

        printf("+------+--------+--------+---------------\n");
        printf("   result count = %d\n", total_results);
        printf("\n");
}


void print_search_results_by_name(const char *starts_with) {
        tool_t tool;
        int i;
        int total_results = 0;

        printf("\n");
        printf("  SEARCH RESULTS:\n");
        printf("  =======================:\n");
        printf("  Query was:\n");
        printf("    name = \"%s\"\n", starts_with);
        printf("\n");
        printf("+------+--------+--------+---------------\n");
        printf("|  ID  |  Qty   |  Cost  | Name   ...\n");
        printf("+======+========+========+===============\n");

        rewind(active_database);
        for(i=0; i<MAX_TOOLS; i++){
                if(fread(&tool, sizeof(tool_t), 1, active_database) != 1){
                        printf("FATAL: Error reading from database.\n");
                        exit(89);
                }

                if(tool.id >= 0 && tool.id < MAX_TOOLS){
                        if(strncmp(tool.name, starts_with, strlen(starts_with)) == 0){
                                printf("|%6d|%8d|%8d|%s\n", tool.id, tool.qty, tool.cost, tool.name);
```

```
                                total_results++;
                        }
                }
        }

        printf("+------+--------+--------+---------------\n");
        printf("   result count = %d\n", total_results);
        printf("\n");
}


void delete_from_db() {
        tool_t tool;
        char buf_tool_id[MAX_LEN_TOOLNAME];

        printf("\nEnter tool record to DELETE (Or enter Q for an ID to Quit):\n");
        prompt_str("  ID", buf_tool_id,   MAX_LEN_TOOLNAME);

        if ( buf_tool_id[0] == 'Q' || buf_tool_id[0] == 'q' ) return;

        tool.id = atoi(buf_tool_id);

        if (tool.id < 0 || tool.id > MAX_TOOLS-1) {
                printf("  !! Tool id is out of range (must be 0-%d inclusive).\n", MAX_TOOLS-1);
                return;
        }

        if (fseek(active_database, tool.id*sizeof(tool_t), SEEK_SET) != 0){
                printf("FATAL: Seek failed.\n");
                exit(79);
        }

        if (fread(&tool, sizeof(tool_t), 1, active_database) != 1){
                printf("FATAL: Read failed.\n");
                exit(78);
        }

        if (tool.id == -1) {
                printf("  !! Tool id DOES NOT EXIST in database.\n");
                rewind(active_database);
                return;
        }

        if (fseek(active_database, tool.id*sizeof(tool_t), SEEK_SET) != 0){
                printf("FATAL: Seek failed.\n");
                exit(77);
        }
```

```
        if (fwrite(&EMPTY_TOOL, sizeof(tool_t), 1, active_database) != 1){
                printf("FATAL: Write failed.\n");
                exit(76);
        }
}


void search_db(){
        char buf[MAX_LEN_TOOLNAME];
        int id;
        for(;;){
                printf("\n");
                printf("Enter I to search by ID, or N to search by name.  Q to cancel.\n");
                prompt_str("Choice?", buf, MAX_LEN_TOOLNAME);
                switch(buf[0]){

                        case 'I': case 'i':
                                prompt_str("ID?", buf, MAX_LEN_TOOLNAME);
                                id = atoi(buf);
                                if (id<0 || id>MAX_TOOLS-1) {
                                        printf(" !! Invalid Tool ID, try again.\n");
                                } else {
                                        print_search_results_by_id(id);
                                }
                                continue;

                        case 'N': case 'n':
                                prompt_str("Name starts with?", buf, MAX_LEN_TOOLNAME);
                                        print_search_results_by_name(buf);
                                continue;

                        case 'Q': case 'q':
                                return;

                        default:
                                continue;
                }
        }
}


void add_update_db() {

        char buf_tool_id[MAX_LEN_TOOLNAME];
        char buf_tool_qty[MAX_LEN_TOOLNAME];
        char buf_tool_cost[MAX_LEN_TOOLNAME];

        printf("\n");
```

```
        printf("\n");
        printf("INPUT NEW RECORDS..\n");

        for(;;){
                tool_t tool;

                printf("\nEnter tool record to add/overwrite (Or enter Q for an ID to Quit):\n");
                prompt_str("  ID           ", buf_tool_id,   MAX_LEN_TOOLNAME);
                if ( buf_tool_id[0] == 'Q' || buf_tool_id[0] == 'q' ) break;

                tool.id = atoi(buf_tool_id);
                if (tool.id < 0 || tool.id > MAX_TOOLS-1) {
                        printf("  !! Tool id is out of range (must be 0-%d inclusive).\n", MAX_TOOLS-1);
                        continue;
                }

                prompt_str("  Name         ", tool.name, MAX_LEN_TOOLNAME);

                prompt_str("  Quantity     ", buf_tool_qty,  MAX_LEN_TOOLNAME);
                tool.qty = atoi(buf_tool_qty);
                if (tool.qty < 0) {
                        printf("  !! Tool quantity must be > 0.\n");
                        continue;
                }

                prompt_str("  Cost (cents) ", buf_tool_cost, MAX_LEN_TOOLNAME);
                tool.cost = atoi(buf_tool_cost);
                if (tool.cost < 0) {
                        printf("  !! Tool cost must be > 0.\n");
                        continue;
                }

                printf("Adding Tool ID=%d, \"%s\", Qty=%d, at a cost of %d cents each.\n\n",
                        tool.id, tool.name, tool.qty, tool.cost);

                write_tool(tool);
        }

        rewind(active_database);
}


void write_tool(tool_t tool) {
        if(fseek(active_database, tool.id*sizeof(tool_t), SEEK_SET) != 0){
                printf("FATAL: Error writing tool to active database.\n");
                exit(99);
        }
        if(fwrite(&tool, sizeof(tool_t), 1, active_database) != 1){
```

```
                printf("FATAL: Error writing tool to active database.\n");
                exit(98);
        }
}


void select_db() {
        char path[MAX_LEN_PATH];
        prompt_str("Path to DB to make active", path, MAX_LEN_PATH);

        FILE *fp = fopen(path, "r+");

        if (!fp) {
                printf("FATAL: Could not open open file \"%s\".\n", path);
                exit(200);
        }

        change_active(fp, path);
}


void initialize_db(){
        char path[MAX_LEN_PATH];
        printf("** Warning! ** This will overwrite the data in the file you specify.\n");
        prompt_str("Path to DB to initialize", path, MAX_LEN_PATH);

        FILE *fp = fopen(path, "w+");
        if (!fp) {
                printf("FATAL: Could not open open file \"%s\" for writing.\n", path);
                exit(19);
        }

        int i;
        for(i=0; i<MAX_TOOLS; i++){
                if( fwrite(&EMPTY_TOOL, sizeof(tool_t), 1, fp) != 1 ) {
                        printf("FATAL: fwrite() failed on \"%s\".\n", path);
                        exit(18);
                }
        }

        change_active(fp, path);
}


void change_active(FILE *new_active, const char* new_path){
        if (active_database) {
                if(fclose(active_database) != 0) {
                        perror("FATAL: Error while closing previously-active database.\n");
```

```
                        exit(201);
                }
        }
        active_database = new_active;

        if (active_database) rewind(active_database);

        strncpy(active_path, new_path, MAX_LEN_PATH-1);
        active_path[MAX_LEN_PATH-1] = '\0';
}


void prompt_str(const char *prompt_string, char* output_string, int max_len){
        char *fgets_ret;
        for(;;){
                printf("    %s: ", prompt_string);
                fflush(stdout);

                fgets_ret = fgets(output_string, max_len, stdin);
                if (fgets_ret){
                        chomp(output_string);
                        if(strlen(output_string) > 0){
                                return;
                        }
                }
        }
        chomp(output_string);
}


/* If the last character of a string is a newline, remove it from the string. */
void chomp(char *str){
        if (str[strlen(str)-1] == '\n') {
                str[strlen(str)-1] = '\0';
        }
}
```

imEnhance.c

```c
/* Includes */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

/* The provided library for reading/writing PBM images. */
#include "iplib2New-modified.c"

/* Represents a rectangular (not necessarily sqaure) sub-region of a larger (or equally-sized) rectangular region. */
typedef struct {
        int upper_left_row;
        int upper_left_col;
        int lower_right_row;
        int lower_right_col;
} window_t;

/* Prototypes */
window_t select_window(int size, int row, int col, int max_rows, int max_cols);
void windowcalc_mean_and_variance(image_ptr image, int image_width, window_t window, double* result_mean, double* result_variance);
void windowcalc_median(image_ptr image, int image_width, window_t window, double* result_median);
int calc_enhanced(unsigned char input_pixel, double overall_mean, double overall_stddev, double window_mean, double window_stddev );

/* Integer comparison function for use with the stdlib's Qsort. */
int int_compare(const void *p1, const void *p2) {
        return ( *(int*)p1 - *(int*)p2 );
}


/* main() */
int main(int argc, char **argv) {

        /* Check arguments. */
        if (argc != 7) {
                fprintf(stderr, "Usage...\n\n");
                fprintf(stderr, " Exactly as given in the assignment:\n");
                fprintf(stderr, " $ ./imEnhance in_file.pgm out_file.avg.pgm out_file.var.pgm out_file.med.pgm outfile.enh.pgm 3\n");
                exit(252);
        }
        int window_size = atoi(argv[6]);
        if (window_size < 3) {
                fprintf(stderr, "Window size must be integer >= 3.\n");
        }
```

```
	/* Attempt to read the input image. */
	image_ptr Image = NULL;
	int rows, cols, type;
	Image = read_pnm(argv[1], &rows, &cols, &type);
	if (Image == NULL) {
		fprintf(stderr, "Failed to open \"%s\" as input image.\n", argv[1]);
		exit(254);
	}

	if(type != 5){
		fprintf(stderr, "Sorry, but I can only handle Type 5 PBM's, i.e., greyscale only!\n");
		exit(10);
	}

	printf("Processing a Type %d image, %dx%d pixels, window size is %dx%d pixels ...\n",
		type, rows, cols, window_size, window_size);
	printf("    File: \"%s\"\n", argv[1]);


	/* Calculate overall statistics for the original image. */
	/* Note: This special-valued window represents the entire image. Use with care. */
	window_t ENTIRE_IMAGE = { 0, 0, rows-1, cols-1 };
	double mean, variance, median, stddev;
	windowcalc_mean_and_variance(Image, cols, ENTIRE_IMAGE, &mean, &variance);
	windowcalc_median(Image, cols, ENTIRE_IMAGE, &median);
	stddev = sqrt(variance);


	/* Allocate additional memory to hold our output images. */
	image_ptr Mean_Image            = (image_ptr) malloc(rows*cols);
	image_ptr Median_Image          = (image_ptr) malloc(rows*cols);
	image_ptr Enhanced_Image        = (image_ptr) malloc(rows*cols);
	image_ptr Variance_Image        = (image_ptr) malloc(rows*cols);
	if ( !(Mean_Image && Variance_Image && Median_Image && Enhanced_Image))
	{
		fprintf(stderr, "Could not allocate memory.\n");
		exit(251);
	}


	/* Iterate over the input image, populating the others along the way. */
	double window_mean, window_variance, window_median, window_stddev;
	int i, j;
	for (i=0; i<rows; i++){
		for (j=0; j<cols; j++) {

			/* Choose the subregion of interest (window) around this pixel. */
			window_t window = select_window(window_size, i, j, rows, cols);
```

```
                /* Calculate the median of just the window. */
                windowcalc_median(Image, cols, window, &window_median);

                /* Calculate the mean and variance of just the window. */
                windowcalc_mean_and_variance(Image, cols, window, &window_mean, &window_variance);

                /* Calculate standard deviation of just the window. */
                window_stddev = sqrt(window_variance);

                /* Calculate the enhanced value of this pixel. */
                int window_enhanced = calc_enhanced( Image[i*cols+j], mean, stddev, window_mean, window_stddev );

                /* Assignments to the output images. */
                Mean_Image[i*cols+j]     = window_mean;
                Median_Image[i*cols+j]   = window_median;
                Enhanced_Image[i*cols+j] = window_enhanced;
                Variance_Image[i*cols+j] = window_stddev;
        }
}


/* Calculate some other statistics, as required by the assignment. */
/* NOTE!!  variance_image_* is really stddev_image_*  !! */
double median_image_median,
       mean_image_mean,     mean_image_variance,
       variance_image_mean, variance_image_variance,
       enhanced_image_mean, enhanced_image_variance;

windowcalc_median(Median_Image, cols, ENTIRE_IMAGE, &median_image_median);
windowcalc_mean_and_variance(Mean_Image, cols, ENTIRE_IMAGE, &mean_image_mean, &mean_image_variance);
windowcalc_mean_and_variance(Variance_Image, cols, ENTIRE_IMAGE, &variance_image_mean, &variance_image_variance);
windowcalc_mean_and_variance(Enhanced_Image, cols, ENTIRE_IMAGE, &enhanced_image_mean, &enhanced_image_variance);

/* And then print them out.. */
printf("    STATISTICS:\n");
printf("        (Numbering corresponds to the requirements stated in the assignment.)\n");
printf("        vi. Original Image\n");
printf("              1. Mean      (M):     %8.2f\n", mean);
printf("              2. Std. Dev. (S):     %8.2f\n", stddev);
printf("              3. Median    (Q):     %8.2f\n", median);
printf("       vii. From returned results\n");
printf("              1. Mean of Block Means:       %8.2f\n", mean_image_mean);
printf("              2. Median of Block Medians:   %8.2f\n", median_image_median);
printf("              3. Std. Dev. of Block Means:  %8.2f\n", sqrt(mean_image_variance));
printf("              4. Mean of Block Std. Dev.'s: %8.2f\n", variance_image_mean);
printf("              5. Enhanced Image Mean:       %8.2f\n", enhanced_image_mean);
printf("... done.\n\n");
```

```
        /* Variance image is too dark, so we multiply it by 2 for human eyes. */
        for (i=0; i<rows; i++) for (j=0; j<cols; j++) Variance_Image[i*cols+j] *= 2;


        /* Write the in-memory output images to their files. */
        write_pnm( Mean_Image,     argv[2], rows, cols, type);
        write_pnm( Variance_Image, argv[3], rows, cols, type);
        write_pnm( Median_Image,   argv[4], rows, cols, type);
        write_pnm( Enhanced_Image, argv[5], rows, cols, type);

        exit(0); /* with much success! */
}


/*******************/
/* OTHER FUNCTIONS */
/*******************/


/* Selects a square window with sides of length max_cols centered at row,col from the larger region of size max_rows x max_cols. */
/* Note, if the square window would exceed the bounds of the larger region, the window will be cropped to fit inside it instead. */
/* This means that this function regularly returns non-square rectangular windows. */
window_t select_window(int size, int row, int col, int max_rows, int max_cols) {
        /* Even arguments for size do not produce even-length window sides; size = 6 is equivalent to size = 7. */
        window_t window;
        window.upper_left_row  = row - size/2;
        window.upper_left_col  = col - size/2;
        window.lower_right_row = row + size/2;
        window.lower_right_col = col + size/2;
        if ( window.upper_left_row < 0 ) window.upper_left_row = 0;
        if ( window.upper_left_col < 0 ) window.upper_left_col = 0;
        if ( window.lower_right_row > max_rows-1 ) window.lower_right_row = max_rows-1;
        if ( window.lower_right_col > max_cols-1 ) window.lower_right_col = max_cols-1;
        return window;
}


/* Calculate the median of a rectangular region. */
void windowcalc_median(image_ptr image, int image_width, window_t window, double* result_median) {
        int rows = window.lower_right_row - window.upper_left_row + 1;
        int cols = window.lower_right_col - window.upper_left_col + 1;


        /* Copy the window into our own image object, because our median algo modifies the array it operates on. */
        image_ptr image_sorted = (image_ptr) malloc(rows*cols*sizeof(unsigned char));
        if (image_sorted == NULL) {
```

```
                fprintf(stderr, "Unable to allocate memory.\n");
                exit(253);
        }

        int i, j;
        int n = 0;
        for ( i=window.upper_left_row; i<=window.lower_right_row; i++ ) {
                for ( j=window.upper_left_col; j<=window.lower_right_col; j++ ) {
                        image_sorted[n] = image[i*image_width+j];
                        n++;
                }
        }


        /* Use the standard-library Quicksort to find the median. */
        qsort(image_sorted, rows*cols, sizeof(unsigned char), int_compare);
        double median = image_sorted[ rows*cols/2 ];
        if ( (rows*cols) % 2 == 0 ) {
                median = (median + image_sorted[ rows*cols/2-1 ])/2;
        }
        free(image_sorted);


        /* Return results via output arguments. */
        if (result_median != NULL) *result_median = median;
        return; /* void */
}


/* Calculate the mean and variance of a rectangular region. */
void windowcalc_mean_and_variance(image_ptr image, int image_width, window_t window, double* result_mean, double* result_variance) {
        unsigned long int n = 0;       /* Pixel count */
        double mean  = 0.0;
        double delta = 0.0;
        double M2    = 0.0;

        int i, j;
        for ( i=window.upper_left_row; i<=window.lower_right_row; i++ ) {
                for ( j=window.upper_left_col; j<=window.lower_right_col; j++ ) {
                        n++;
                        unsigned int pixel_value = image[i*image_width + j];
                        delta = pixel_value - mean;
                        mean = mean + delta / n;
                        M2 = M2 + delta * (pixel_value - mean);
                }
        }
        double variance = M2/(n - 1);
```

```
        /* Return results via output arguments. */
        if (result_mean != NULL) *result_mean = mean;
        if (result_variance != NULL) *result_variance = variance;
}


/* Calculate the enhanced value for a pixel given the other required terms. */
int calc_enhanced(unsigned char input_pixel, double overall_mean, double overall_stddev, double window_mean, double window_stddev ){

        /* Constants for the assignment-provided transformation. */
        double A       = 2.00;
        double C1      = 0.40;
        double C2      = 0.02;
        double C3      = 0.40;

        /* The actual transformation:  Selective brightening. */
        if(
                (window_mean <= overall_mean * C1) &&
                (overall_stddev * C2 <= window_stddev) &&
                (overall_stddev * C3 >= window_stddev)
        ){
                return A * input_pixel;
        } else {
                return input_pixel;
        }
}
```

run_tests.sh

```
#!/bin/sh
#
# First, put your input images in ./test_input, for example ./test_input/Duck_on_Pond.pgm , ./test_input/Snowy_Forest.pgm
#       (name them ending in ".pgm")
#
# Then, run this script.  It will:
#       1) Loop over each of your test images,
#       2) Run the imEnhance program on each one,
#       3)   at several window sizes (3x3, 5x5, 7x7, 9x9, etc.),
#       3) Place the output under ./test_output according to window size, thus:
#
#               ./test_output/3x3/Duck_on_Pond.enhanced.pgm
#               ./test_output/3x3/Duck_on_Pond.mean.pgm
#               ./test_output/3x3/Duck_on_Pond.median.pgm
#               ./test_output/3x3/Duck_on_Pond.var.pgm
#
#               ./test_output/3x3/Snowy_Forest.enhanced.pgm
#               ./test_output/3x3/Snowy_Forest.mean.pgm
#               ./test_output/3x3/Snowy_Forest.median.pgm
#               ./test_output/3x3/Snowy_Forest.var.pgm
#
#               ./test_output/5x5/Duck_on_Pond.enhanced.pgm
#               ./test_output/5x5/Duck_on_Pond.mean.pgm
#               ./test_output/5x5/Duck_on_Pond.median.pgm
#               ./test_output/5x5/Duck_on_Pond.var.pgm
#
#               ./test_output/5x5/Snowy_Forest.enhanced.pgm
#               ./test_output/5x5/Snowy_Forest.mean.pgm
#               ./test_output/5x5/Snowy_Forest.median.pgm
#               ./test_output/5x5/Snowy_Forest.var.pgm
#
#               etc. etc.
#
# If you wish to re-run the script, you will need to rm or mv ./test_output.

INPUT_DIR="./test_input"
OUTPUT_DIR="./test_output"

#imEnhance="./imEnhance.gprof.o"
imEnhance="./imEnhance"

mkdir $OUTPUT_DIR
if [ $? -ne 0 ]; then
        exit
fi;
```

```
for n in 3 5 7 9 15; do

        mkdir "${OUTPUT_DIR}/${n}x${n}"

        for i in $INPUT_DIR/*.pgm; do
                i=`basename $i`
                $imEnhance                                      \
                        $INPUT_DIR/$i                                   \
                        $OUTPUT_DIR/${n}x${n}/$i.mean.pgm    \
                        $OUTPUT_DIR/${n}x${n}/$i.var.pgm     \
                        $OUTPUT_DIR/${n}x${n}/$i.median.pgm  \
                        $OUTPUT_DIR/${n}x${n}/$i.enhanced.pgm        \
                        ${n}
        done
done
```

Makefile

```
# This is a Makefile.

CC=gcc
CFLAGS=-Wall

all: imEnhance minidb

imEnhance: imEnhance.c iplib2New-modified.c
	$(CC) $(CFLAGS) -O3 -lm -o imEnhance         imEnhance.c
#	$(CC) $(CFLAGS) -pg -lm -o imEnhance.gprof.o imEnhance.c
#	^^ this can be used for profiling prior to hand-optimization
#	$(CC) $(CFLAGS) -g  -lm -o imEnhance.gprof.o imEnhance.c
#	^^ or you might need a debug build (ok, more likely, I'll need it.)

minidb: minidb.c
	$(CC) $(CFLAGS) -o minidb minidb.c

clean:
	rm -rf imEnhance imEnhance.gprof.o gmon.out test_output
	rm -rf minidb

test:
	rm -rf test_output
	./run_tests
```