**CS 350:COMPUTER SYSTEM CONCEPTS Kharon Harrell**

**PROGRAMMING ASSIGNMENT 2**

**Q1. MINI DATABASE IN C**

**Problem Statement**

As the owner of a hardware store, there is a need for a database to store inventory items, prices, quantity, and more. A database management system is necessary to manage data efficiently and enables users to perform multiple tasks at once.

**Problem Analysis**

This program is expected to initialize a data file, allow input data per item, lists all items, delete records of items, and updates any information in the file.

Input(s): item data: name, quantity, record number, cost

Output(s): inventory database: name, quantity, record number, cost

Constraint(s): items < 100

**Algorithm Design**

Initial Algorithm

1. Initialize a struct data type with the following attributes; String name, int quantity, int record number, double cost
2. Create functions to manipulate database
3. Display database information

**Test Plan**

Due to the nature of this program, there will be a lot more testing than there was previously. There will be a test case for an empty database and full database, this will serve as our edge cases. A test case for each manipulation of the database, such as adding and removing from the database. Finally, there will be a test case for a normal usage of the database.

**Implementation**

**// -------------------------------------------------------------**

**// minidb.c**

**// Program with all the functionality of the Mini Database in C**

**// Author: Kharon Harrell**

**// CS 350 Section 001**

**// ------------------------------------------------------------**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "minidb.h"

inventory\_t database[100];

void printRecord(inventory\_t item);

void printMenu(){

printf("1. Initalize the database\n");

printf("2. Input new records\n");

printf("3. Search for a record\n");

printf("4. Delete a record\n");

printf("5. List all records\n");

printf("6. List all records (sorted)\n");

printf("7. Save the database\n");

printf("8. Load an existing database\n");

printf("9. Exit\n");

}

void initializeDB() {

printf("\nInitializing...\n");

for(int i=0; i < 100; i++){

database[i].name[0] = '\0';

database[i].record = i+1;

database[i].quantity = 0;

database[i].cost = 0;

}

}

void listDB() {

for(int i = 0; i < 100; i++){

printRecord(database[i]);

}

}

void printRecord(inventory\_t item) {

printf("%d ", item.record);

printf("%s ", item.name);

printf("%d ", item.quantity);

printf("%6.2f\n", item.cost);

}

void inputRecord() {

inventory\_t record;

printf("Enter name:\n");

scanf("%s", record.name);

printf("Enter Record ID:\n");

scanf("%d", &record.record);

printf("Enter Quantity:\n");

scanf("%d", &record.quantity);

printf("Enter Cost:\n");

scanf("%f", &record.cost);

database[record.record-1] = record;

}

void searchDB() {

int id = 0;

char name[50];

printf("Enter Tool ID:\n");

scanf("%d", &id);

printf("Enter Tool Name:\n");

scanf("%s", name);

for(int i =0; i < 100; i++){

if((database[i].record == id) && (strcmp(database[i].name, name) == 0)){

printRecord(database[i]);

return;

}

}

printf("Record not found\n");

}

void deleteDB() {

int id = 0;

printf("Enter Tool ID:\n");

scanf("%d", &id);

database[id-1].name[0] = '\0';

database[id-1].cost = 0;

database[id-1].quantity = 0;

}

void saveDB() {

char filename[50];

FILE \*filePointer = NULL;

printf("Enter filename:\n");

scanf("%s", filename);

filePointer = fopen(filename, "wb");

fwrite(database, sizeof(inventory\_t), 100, filePointer);

fclose(filePointer);

}

void loadDB() {

char filename[50];

FILE \*filePointer = NULL;

printf("Enter filename:\n");

scanf("%s", filename);

filePointer = fopen(filename, "rb");

fread(database, sizeof(inventory\_t), 100, filePointer);

fclose(filePointer);

}

void listSortedDB() {

char temp[50];

for(int i = 0; i < 100; i++){

for(int j=1; j < 100; j++) {

if(strcmp(database[i].name, database[j].name) > 0){

strcpy(temp,database[i].name);

strcpy(database[i].name,database[j].name);

strcpy(database[j].name,temp);

}

}

printRecord(database[i]);

}

}

**Testing**

After successfully compiling and linking the program, I realized that my test cases had to be a bit different than I expected. The final testing consisted of manual testing for each function, such as adding, removing, listing, etc. I decided not to include screenshots of the testing due to the sheer volume of functionality.

**Q2. IMAGE SUBSAMPLING**

**Problem Statement**

For this program, we will reduce an image *A* with a size to a new image *B* with a size of . In simple terms, we will be reducing the size of image to the size of the user’s choosing, producing a new, smaller image. The program is simple in theory, but rigorous in practice.

**Problem Analysis**

This program will reduce the size of an image to a smaller size, while preserving the display of the image.

Input(s): initial image

Output(s): reduced image file

Constraint(s): max file size for an image

Formulas: &

**Algorithm Design**

Initial Algorithm

1. Read the input image
2. Partition the original image into smaller subimages
3. Push each block into a function to analyze the image
4. Arrange results into the corresponding image
5. Write reduced image to an output file
6. Calculate the mean, standard deviation, and the average standard deviation for the for the original image

**Test Plan**

Testing this program should only require two test cases; a simple image containing very little complexity, perhaps an image with basic geometry and a regular image.

**Implementation**

**// -------------------------------------------------------------**

**// minidb.c**

**// Program with all the functionality of the Mini Database in C**

**// Author: Kharon Harrell**

**// CS 350 Section 001**

**// ------------------------------------------------------------**

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include "iplib2New.h"

void image\_analysis(image\_ptr imagePtr, int row, int col, int width, int height, int rows, int cols, float \*mean, float \*sd);

int main(int argc, char \*\*argv)

{

image\_ptr imagePtr, imagePtr2;

unsigned char \*\* output\_image = NULL; /\* space for output image \*/

char \* output\_filename = NULL;

int factor = 0;

float mean = 0;

float sd = 0;

int rows, cols, type;

int i=0, j=0, value=0, count=0;

int tp;

int n;

/\* check inputs \*/

if (argc != 4)

{

printf("wrong inputs: use %s infile out1 out2 \n", argv[0]);

return 0;

}

/\* first read-in the image \*/

printf("reading input image ... \n");

imagePtr = read\_pnm(argv[1], &rows, &cols, &type);

printf("image read successfully \n");

printf("rows=%d, cols=%d, type=%d \n", rows, cols, type);

/\* printf("rows=%d, cols=%d, type=%d \n", ROWS, COLS, TYPE); \*/

output\_filename = argv[2]; //output file = third arg

factor = atoi(argv[3]); //factor = string to integer of arguemnt 4

n = rows/factor; //n = dimension of reduced image

//imageptr2 = allocates memory for reduced image

imagePtr2 = malloc(n\*n\*sizeof(unsigned char));

//iterates over the entire image and inputs into image analysis

for(int i=0; i<rows; i = i+factor){

for(int j=0; j<cols; j = j+factor){

image\_analysis(imagePtr, i, j, factor, factor, rows, cols, &mean, &sd);

imagePtr2[(i/factor) \* n + j/factor] = mean;

}

}

write\_pnm(imagePtr2, output\_filename, rows/factor, cols/factor, PGM); //writes to a file

image\_analysis(imagePtr, 0, 0, rows, cols, rows, cols, &mean, &sd);

printf("overall mean = %6.2f\n", mean);

printf("overall sd = %6.2f\n", sd);

return 0;

}

//image analsis = takes image, row colum of sub block, height and width of the subblock, rows and cols are dimensions of the original image

void image\_analysis(image\_ptr imagePtr, int row, int col, int width, int height, int rows, int cols, float \*mean, float \*sd) {

float sum = 0;

//calculate sum

for(int i = 0; i < height; i++){

for(int j = 0; j < width; j++){

sum += imagePtr[(row + i)\*rows + col + j];

}

}

\*mean = sum/(width\*height);

sum = 0;

//calculate sd

for(int i= 0; i< height; i++){

for(int j=0; j<width;j++){

sum += pow(imagePtr[(row + i)\*rows + col + j]-\*mean, 2);

}

}

sum = 1.0/(width\*height-1.0)\*sum;

\*sd = sqrt(sum);

}

**Testing**

Similar to the previous program, my test cases changed. Instead of using the initial test cases, I decided to use the images given, and simply compared it to the original images with reduced ones that my program generated.



**Original Image**

****

**Reduced Image**