Computer Science 327
Project 1
C Programming Project
Rainfall Flood Simulation

Notes

Some parts of this project contain mathematical formulas, but nothing more than you have seen in Math 165 or before. There may be parts of this project description that may be ambiguous (not on purpose), but it is your job to also clarify program specifications given in a natural language. Please read the assignments early and asks questions in class. Questions that lead to further specification or changes in the project specification will be posted in Piazza or in the announcements section of the Canvas course.

Introduction

Understanding how water accumulates due to runoff is an important component in the analysis of flooding. In this project, we will use LIDAR data for land height at 1 meter resolution to create a simple simulation of water flow over the terrain when it rains. Note that this is not a scientific method, and will not necessarily give accurate, or even useful results. However, it should indicate where water is most likely to pool in a geographic area.

Our model is similar to that of a two-dimensional cellular automata which have long been used to model a variety of real and imaginary systems. A Google search on "cellular automata" will yield a plethora web sites dedicated to different forms of cellular automata. Our model is similar in that each cell will represent one square meter of terrain. https://mathworld.wolfram.com/ElementaryCellularAutomaton.html At each time step of the simulation, water flows between cells based on a formula that is dependent on the height of the cell and the height of water in the cell.

Data Format: Point Cloud Data

The point cloud data is a stream of 3-tuples that represent grid locations and height. Each tuple consists of three double floating point numbers in ASCII text format that represent the x and y grid coordinate followed by the terrain height data. The x and y coordinates give the center of the cell. The stream of tuples is terminated by the end of the stream (end of file). An example stream is shown below.

```
445000.5 4650999.5 304.95001220703125
445001.5 4650999.5 304.95001220703125
445002.5 4650999.5 304.959991455078125
445003.5 4650999.5 304.94000244140625
445004.5 4650999.5 304.94000244140625
.
.
.
445995.5 4650000.5 284.209991455078125
445996.5 4650000.5 284.220001220703125
445997.5 4650000.5 284.260009765625
445998.5 4650000.5 284.269989013671875
445999.5 4650000.5 284.269989013671875
```

Project 1, Part a

The purpose of this part of the project is to create the infrastructure for the assignment and to do a simple C program that will get you familiar with the process. The project is divided into several parts.

1. Remove or rename any previous Git repositories that you have created with the following name, and create a new repository named "**coms3270P1**" Also remember that part of your grade will depend on README and DEVELOPER files. Please see the syllabus.

2. Create a function named "stat1" that reads a point cloud stream until the end and prints out the height and grid location of the minimum height and maximum height. It should also print out the average height of all data points. Place this function in the file "pointcloud.c"

3. Create a "main" function that calls the "stat1" function and place this function in the file "pointcloud.c"

4. Create a header file named "pointcloud.h" that creates correct prototypes for the functions defined in 2 above. Be sure to include correct guards for the header file.

5. Create a makefile for this project so that when a user types make in the top directory for the repository, the executable program named "stat" that if run, the main program in the "pointcloud.c" file is executed.

6. Create at least one datafile in the point cloud format with the properties that the minimum is height is 0, the maximum height is 10, and the average height is 1.

7. Create and store your project in the gitlab repository git.las.iastate.edu. When completed and your program is correct and documented, create a tag with the name “partacomplete”

8. At least 10% of points will come from documentation and following the syllabus requirements for the project.

Project 1, Part b

In this part of the assignment, you will be creating several functions to manipulate 2D arrays and output bitmap pictures of them.

1. Write a function to allocate the space for a 2-dimensional array and initialize all elements to zero. The allocation must use the stdlib malloc or calloc function (stdlib.h). The prototype for this function must be void `*allocateArray(int rows, int columns)`, where rows is the number of rows in the array and columns is the number of columns in the array. Each cell in the array is of type `double`. Place this function in the file `util.c`.

2. Write an access macro using `#define` to “return” the index in a one-dimensional array of an element in a 2-dimensional array. Your macro should take the parameters of row and column in the 2-dimensional array. This macro along with the prototype for the allocateArray function above should be placed in the file `util.h`.

3. Create a structure to store points from the point cloud. The structure should hold the data for the x, y, and z (height) coordinates. In addition, the structure should contain a fourth parameter that will represent the amount of water at that location. All of these variables are of type double. In addition to these double types, we will also utilize 4 pointers that point to this structure. Call these pointers north, south, east, and west. In total, this structure contains 8 variables. Use a typedef to create the type `pcd_t` to be the structure of this type and place it in the pointcloud.h header file.

4. Because input may come from a stream that cannot be rewound, stdin, for example, we must store the data as we read it. However, it is unknown when the end of stream will occur. Thus, there is a need for an expanding data structure. There are two possibilities: linked list and expanding array. For this project you will use an expanding array that doubles in size when there is no room left to add an element. (This is similar to how ArrayList works in Java.) To begin, place the following prototype functions and structures in util.h. These will be the common interfaces for other programs to use the C array list you develop.

```
typedef struct
{
	int max_size;
	int max_element_size;
	void* data;
	int size;
} List;
```

Where `max_size` is the current maximum possible size of the list; `max_element_size` is the maximum size of any element in the array; `data` is a pointer to the current array containing data; and `size` is the current number of elements in the list. The following prototypes for functions are also in util.h.

```
int listInit(List* l, int max_elmt_size);
void listAddEnd(List* l, void* elmt);
void *listGet(List* l, int index);
```

The listInit function initializes a List structure with data to create an empty list. The default initial max_size of the list is 10.

The listAddEnd adds an element to the end of the list. The data is copied from the elmt pointer. Note that max_elmt_size data is copied, even if not all the data pointed to be elmt is not used. Thus, elmt pointer must point to a block of memory that is at least elmt_size large.

The listGet function returns a pointer to the data in the list. Note that it is not copied and that changing the data the pointer references changes the data in the list.

5. Implement the prototype functions in 4.

6. Use the expanding list and structures you developed above to read point cloud data from stdin. You will use a new format for the point cloud data format that includes a single integer as the first data item that is the number **columns** in the data set.

   For example:

```
1000
445000.5 4650999.5 304.95001220703125
445001.5 4650999.5 304.95001220703125
445002.5 4650999.5 304.959991455078125
445003.5 4650999.5 304.94000244140625
445004.5 4650999.5 304.94000244140625
```

```
.
.
.
445995.5 4650000.5 284.209991455078125
445996.5 4650000.5 284.220001220703125
445997.5 4650000.5 284.260009765625
445998.5 4650000.5 284.269989013671875
445999.5 4650000.5 284.269989013671875
```

To read the data, add a function to file pointcloudio.c and include the function with the prototype `readPointCloudData( FILE *stream )`. (More about how to use the FILE * type below) This function should read the point stream data and store each point in an expanding list. The list elements should be of type pcd_t.

You can use the fscanf function in stdio.h to read data from an arbitrary stream. In this case, you can use it to read from files or stdin. In order to use this read from standard input, you put the file stream identifier as the first parameter. The rest of the parameters are the same. For example, `fscanf( stdin, “%d”, &v)` is the same as `scanf(“%d”, &v)`.

7. Add the bmp.c and bmp.h files from this github site to your project: https://github.com/wernsey/bitmap

8. Assume for now that rows are given from top to bottom. Write a function in pointcloud.c with prototype `void imagePointCloud( List *l, char *filename )`, where l is a pointer to a list structure that contains the point-cloud data, and filename is a pointer to a cstring that is the name of the file (and path if desired) to save the data. The function then creates a gif image of the point-cloud data as follows:

   Determine the minimum and maximum z (height) over all points in the point-cloud data.
   Divide the range of heights into 256 equal lengths.
   Assign heights in the lowest (smallest) height-range to black (0, 0, 0) pixel value.
   Assign heights in the highest (largest) height-range to white (255, 255, 255) pixel value.
   Heights in other ranges are distributed proportionally. For example, a height in the middle between the highest and lowest would have a pixel value of (128, 128, 128). Note that this is not a hard computation and is a very simple expression. Don’t over think it.

   Use the bmp.c functions to output the point-cloud data as a gif file that can be viewed.

9. Create a file named display.c that contains a main function that calls functions above to read a point-cloud data file and then output to a file named out.gif.

10. Update your makefile and documentation appropriately.

11. Create at least two small point-cloud test files to help debug your code. An example of such a file might be a file that contains 4 by 4 (16 points) of which they are all the same, or a file that contains 4 by 8 (32 points) of which half are height 100 and half are height 200, etc.

Project 1, Part c

In this part of the project, you will create a simple watershed simulation to show where and how water drains over the terrain. This simulation is not based on any scientific paper, and is not a correct simulation or model of how water actually moves over terrain. However, it will give you a good idea on how these types of simulations can be written and of course, help with C programming and design.

A significant component of writing software often involves refactoring code. This can happen for many reasons, including changes in requirements, design mistakes, programming errors, etc. In this part, you will refactor some existing methods to make the program more readable and easier to change.

1. At this point, you should have part a and part b complete. Since we will be changing files that will cause part a and part b to potentially fail to compile, make a branch named "**partab**" You can then continue fix any errors on parts a or b if they come up without affecting changes you will be making to the main branch. (This would be similar to a version release for a software product.) When you are sure that parts a and b are working correctly, create a tag in this branch named "partabcomplete" The TAs will use this tag to pull from your repository to grade parts a and b. If this tag is not present or incorrectly named, we cannot grade your project, and you will lose many points. Please double-check your names.

2. For this part you will create a main function in the file named "**watershed.c**" and when compiled with the rest of the objects files, will create an executable file named "**watershed**" (see makefile requirements below). This executable has several command line parameters defined as follows:

```
./watershed <ifile> <iter> <iwater> <wcoef> <ecoef> <ofilebase> <seq>
```

   a. **ifile** is the input file name of the pointcloud file.
   b. **iter** is the integer number of computation steps to perform.
   c. **iwater** is the initial floating point amount of water covering the pointcloud.
   d. **wcoef** is a floating point number between 0.0 and 0.2, which is the water flow coefficient.

e. **ecoef** is a floating point number between 0.9 and 1.0, which is the evaporation coefficient.
f. **ofilebase** is the base filename of the output file. If no seq parameter is given the output file is simply <ofilebase>.gif, and is the final output of the simulation.
g. **seq** is a number indicating that every seq number of steps a new output file is generated with the name <ofilebase><iter step>.gif. A seq number of 1 would output a new file every step.

Note that the functions sscanf and sprintf may be useful.

3. In the previous part you created a data structure named pcd_t. Update this structure so that the names of the four pointers are **north**, **south**, **east**, and **west**. Also update the name in the field that represents the amount of water with the name **wd**. Note that failure to make this change may cause your program to be improperly graded. Also, you may add additional fields to this structure as needed, which can make parts of this program easier to write.

4. Create a new structure in pointcloud.c that represents a pointcloud and all its data. Alias this structure to the **pointcloud_t** type. This structure must contain the following items.

   a. List points; // the points in the pointcloud
   b. int rows; // the number of rows in the pointcloud
   c. int cols; // the number of columns in the pointcloud

   However, it will be very useful to store other data in this structure. For example, I created a statistics structure and then added it to this structure so that every pointcloud contains a set of statistics associated with it. This structure is updated when the pointcloud is read.

5. Refactor the function readPointCloudData so that it returns a pointer to a pointcloud_t structure that contains all the data read from the pointcloud file.

```
pointcloud_t *readPointCloudData(FILE* stream );
```

   Refactor imagePointCloud so that it takes a pointcloud_t * as a parameter.

```
void imagePointCloud(pointcloud_t *pc, char* filename);
```

   Note: Do not do this refactoring until after you finish initializeWatershed below. This function should now output pointclouds with the correct orientation. (North is up and east is right)

6. Write an **initializeWatershed** function in pointcloud.c that performs the following functions:

   a. Sets the amount of water in each location to zero.
   b. Set the north, south, east, and west pointers to point to the cells (points) that are directly north, south, east, and west of each point. For this, you will have to use the x and y coordinates of the pointcloud. The point with the minimum x coordinate and the minimum y coordinate is the very most southwest point. The point with the maximum x coordinate and maximum y coordinate is the very most northeast point. Pointers that point outside of the grid should be set to NULL. E.g. the north pointer of the farthest northeast point.
   c. The prototype for this function must be:

   ```
   int initializeWatershed(pointcloud_t *);
   ```

   d. You may use the return value of 0 to indicate no error, and other integers to communicate an error code.

7. Write a function named **watershedAddUniformWater** in pointcloud.c that takes in a pointcloud_t pointer and a double amount parameter. The functions adds the amount to each point in the pointcloud. (I may interchangeably use the word point and cell to mean the same thing.) The prototype is:

   ```
   void watershedAddUniformWater(pointcloud_t* pc, double amount);
   ```

8. Write a function named **watershedStep** that computes one time step (iteration) of the analysis. The prototype for this function is:

   ```
   void watershedStep(pointcloud_t* pc);
   ```

   One step the analysis consists of updating all the points. The following computation is used to update a single cell (point).

   Let C be any cell in the pointcloud. Let W be the cell west of C, E by the cell east of C, N be the cell north of C, and S be the cell south of C. Let C_w, W_w, E_w, N_w, and S_w be the current water at each of these cells. Also let C_e, W_e, E_e, N_e, and S_e be the elevation at these points. Let the function f be defined as follows:

$f: \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, defined by

$$f(t1, w1, t2, w2) = (t2 + w2) - (t1 + w1) * wcoef.$$

Then the amount of water in cell C on the next step is given by:

$f(C_e, C_w, W_e, W_w) + f(C_e, C_w, E_e, E_w) + f(C_e, C_w, N_e, N_w) + f(C_e, C_w, S_e, S_w) - C_w * ecoef$

Note that you should not update any of the cell for the next step until all cell values for the next step have been computed. For edge cells with neighbors that are null, no value is added to the computation.

It behooves you to make this computation easy to change. There may be updates to this function before the assignment is due. (up to two days before).

9. Write a function named imagePointCloudWater in pointcloud.c that creates a gif image of the water content for each point. The prototype of this function is

```
void imagePointCloudWater(pointcloud_t* pc, double maxwd, char* filename);
```

The maxwd parameter is the value where point data water depth on the map with this value or greater is completely blue (with no transparency). Whereas a water depth value of 0 depicts the original map with no shading. You may research how to shade a map. I am also available in lecture to talk about ways to go about this. The image must be displayed in the correct orientation with north at the top of the image.

10. Update the makefile so that if you type make with no parameters, the executable watershed is created.

11. In your repository, tag the final version of part c as **p1complete**.