CSC221 Data Structures & Algorithms I Fall 2016

Programming Assignment #1 – 3-D Arrays

In this lab our goals are to:

- review your C++ programming skills
- work with binary files, 3-D dynamically allocated arrays
- work with header files, object files, linking, and Makefiles

In this assignment we will compute and visualize a height map of a 3-D LiDAR data set. The main program is provided for you in pre-compiled form (i.e., main.o). Visualization is provided by a simple local library (`libppm_graphic.a`). Your task is to implement a class named **array3d** to store the LiDAR data set, and to provide the necessary class methods. To get the parts to work together, we need to all agree on the public interface for class **array3d**:

```
class array3d {
   private:

       //  Private part of your implementation goes here.

   public:
      array3d() ;                    // Constructor
      ~array3d() ;                   // Destructor
      bool read( char * fname ) ;
      void get_sizes( int & m, int & n, int & p ) ;
      int get_zmap_value( int x, int y  ) ;
} ;
```

Public Member Functions:

bool read(char * fname) : This public member function accepts a C-style character string representing a file name. This file is a binary file containing pre-processed LiDAR data. Function `read()` should:

- open the file
- read and store the sizes m, n, and p
- allocate an $m \times n \times p$ 3-D array of characters
- read the data into the 3-D array
- close the file

Function `read()` returns **true** if successful, and **false** if an error occurs.[1]

[1] An error may occur if the function is unable to open the file, or if memory allocation fails when creating the 3-D array.

void get_sizes(int & m, int & n, int & p) : This public member function gives the caller the values of m, n, and p using pass-by-reference semantics.

void get_zmap_value(int x, int y) : This function returns the highest occupied cell (largest z-index containing the value 1) for position `x, y` in the ground plane,

Task Overview:

1. Design and implement the private details of class **array3d**
2. Implement the public methods details of class **array3d**
3. Copy the main program from `/usr/local/compiled_code/main.o` to your Lab1 directory.
4. Copy the data file `/usr/local/data/ig339_crp.dat` to your Lab1 directory.
5. Write a Makefile. Your Makefile must:
   - compile your class **array3d.cc** to an object code file
   - link the array3d.o, main.o, and the ppm_graphic library to create an executable named **zview**.

Helpful Hints: Our PPM library is in /usr/local/lib. You will need to use two compiler options during the linking step so that the `g++` compiler:

1. knows the location of the PPM library, and
2. links the PPM library into your executable.

The correct compiler options are:

```
-L/usr/local/lib  -lppm_graphic
```

LiDAR Data:

LiDAR instruments produce a set of points where a reflection was detected; these data are often called a "point cloud". For purposes of this assignment, the point cloud has been pre-processed into an "implicit geometry" form. In the implicit geometry, the observed space is divided into a uniform mesh of "cells". In this data set, each cell represents a cube approximately 0.5 meters (19.7 inches) in diameter. Any cell which is contains by at least one point from in raw LiDAR point cloud is marked "occupied" (array entry is 1). All other cells are marked "empty" (array entry is 0).

Data Format:

The first 12 bytes of the file represent three 4-byte integers: `m`, `n`, and `p`, representing the number of rows, columns, and pages in the data that follows.

The remaining bytes (C++ type char) of the file indicate "occupied" by the value 1, and "empty" with the value 0. The data is most easily understood by thinking in terms of an `i-j-k` coordinate system, where:

- the positive `i` direction is South.
- the positive `j` direction is East.
- the positive `k` direction is Up.

The first p consecutive bytes contain the values in the Z-direction for the upper-left position, i.e. coordinate (0, 0) of the ground plane. The next p bytes contain the the values in the Z-direction for the next position in the ground plane one position to the South. You can think of the data as a matrix in column major order, where each "entry" consists of the p vertical values. I.e., the dimension ordering in file is: **columns**, **rows pages**.

It is easy for some folks to become disoriented when thinking about dimension ordering. To minimize confusion, let us agree to a naming convention for the variables involved. First consider the sizes:

m is the number of rows
n is the number of columns
p is the number of pages

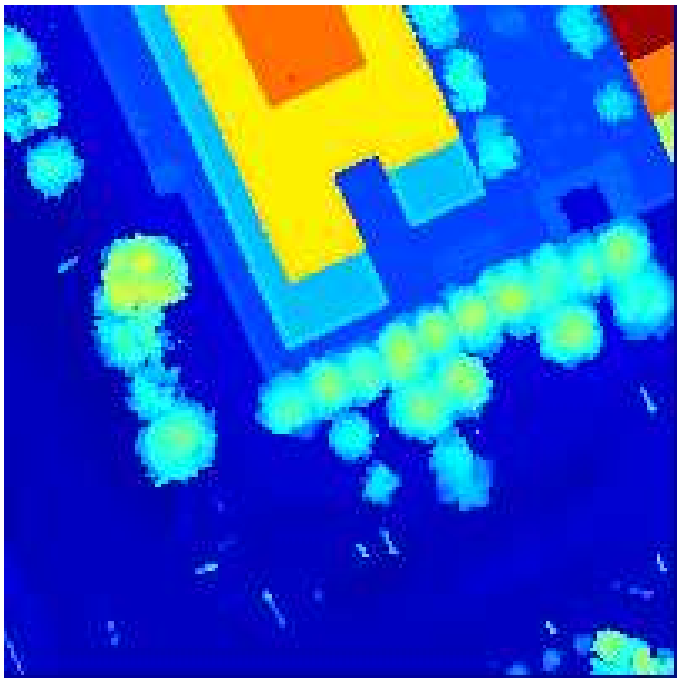
Also, let us agree on the indexing variables:

i refers to the i^{th} row
j refers to the j^{th} column
k refers to the k^{th} page

Output: If all has gone well, the command:

```
%  zview ig339_crp.dat
```

will produce the following aerial view of the LiDAR data set (a.k.a. "LiDAR Elevation"):



Notice you can easily identify, trees, buildings, and light poles.

Turn In:

Keep all your work in a sub-directory named `Lab1`. Change to the parent directory of `Lab1` and create a tar archive of your work using the command:

```
% tar cf Lab1.tar Lab1
```

Upload the file `Lab1.tar` to your account on telesto.