CS246—Assignment 3 (Fall 2016)

J. Avery, C. Kierstead, and B. Lushman

Due Date 1: Monday, October 24, 5pm
Due Date 2: Monday, October 31, 5pm

Questions 1a, 2a and 3a are due on Due Date 1; the remainder of the assignment is due on Due Date 2.

Note: You must use the C++ I/O streaming and memory management facilities on this assignment. Moreover, the only standard headers you may `#include` are `<iostream>`, `<fstream>`, `<sstream>`, `<iomanip>`, `<string>`, and `<utility>`. Marmoset will be programmed to **reject** submissions that violate these restrictions.
Note: Each question on this assignment asks you to write a C++ program, and the programs you write on this assignment each span multiple files. Moreover, each question asks you to submit a `Makefile` for building your program. For these reasons, we **strongly** recommend that you develop your solution for each question in a separate directory. Just remember that, for each question, you should be *in* that directory when you create your zip file, so that your zip file does not contain any extra directory structure.
Note: Questions on this assignment will be hand-marked to ensure that you are writing high-quality code, and to ensure that your solutions employ the programming techniques mandated by each question.
Note: You are not permitted to ask any public questions on Piazza about what the programs that make up the assignment are supposed to do. A major part of this assignment involves designing test cases, and questions that ask what the programs should do in one case or another will give away potential test cases to the rest of the class. Instead, we will provide compiled executables, suitable for running on `linux.student.cs`, that you can use to check intended behaviour. Questions found in violation of this rule will be marked private or deleted; repeat offences could be subject to discipline.

1. In this exercise, you will write a C++ class (implemented as a `struct`) to control a simple robotic drone exploring some terrain. Your drone starts at coordinates (0,0), facing north. Use the following structure definition for coordinates:

   ```
   struct Position {
     int ew, ns;
     Position( int ew = 0, int ns = 0 );
   };
   ```

   The east-west direction is the first component of a position, and the north-south direction is the second. Your `Drone` class must be properly initialized via a constructor, and must provide the following methods:

Method	*Description*
`void forward()`	Move the drone one unit forward.
`void backward()`	Moves the drone one unit backward.
`void left()`	Turns the drone 90 degrees to the left, while remaining in the same location.
`void right()`	Turns the drone 90 degrees to the right, while remaining in the same location.
`Position current()`	Returns the current position of the drone.
`int totalDistance()`	Returns the total units of distance travelled by the drone.
`int manhattanDistance()`	Returns the "Manhattan distance" between the current position and the origin where the Manhattan distance defined as the absolute north-south displacement plus the absolute east-west displacement.
`bool repeated()`	Returns true if the current position is one that the drone has previously visited.

For simplicity, you may assume that the drone will never visit more than 50 positions before running out of fuel or otherwise breaking down.

Implement the specified operations for the `Drone`. (Some starter code has been provided for you in the file `drone.h`, along with a sample executable.) **You may not change the contents of `drone.h` other than by adding your instance variables and comments i.e. the interface must stay exactly the same.**

The test harness `a3q1.cc` is provided with which you may interact with your drone for testing purposes. **The test harness is not robust and you are not to devise tests for it, just for the `Drone` class. Do not change this file.**

(a) **Due on Due Date 1:** Design the test suite `suiteq1.txt` for this program and zip the suite into `a3q1a.zip`.

(b) **Due on Due Date 2:** Implement this in C++ and place the files `Makefile`, `a3q1.cc`, `drone.h` and `drone.cc` in the zip file, `a3q1b.zip`. Your Makefile must create an executable named `drone`. Note that the executable name is case-sensitive.

2. Consider the following class definition for a two-dimensional integer `Matrix` class:

```
class Matrix {
  public:
    // Requires numRows >= 0 && numCols >= 0.
    // If both dimensions are 0, then array pointer is set to nullptr.
    // If only one dimension is 0, also treat as 0x0 matrix; otherwise,
    // allocate space and set values in 2-D array to 0--fill later using
    // either operator>> or set().
    Matrix( int numRows = 0, int numCols = 0 );
    Matrix( const Matrix & );        // copy constructor
    Matrix( Matrix && );             // move constructor
    ~Matrix();
    Matrix & operator=( const Matrix & );  // copy assignment operator
    Matrix & operator=( Matrix && ); // move assignment operator
    Matrix operator+( const Matrix & ) const;  // add two matrices
    Matrix operator*( const Matrix & ) const;  // multiply two matrices
    int rows() const; // returns the number of rows in the matrix
    int cols() const; // returns the number of columns in the matrix
```

```
    // Requires 0 <= row < this->rows() && 0 <= col < this->cols()
    // Sets this's [row][col] == value
    void set( int row, int col, int value );

    // Requires 0 <= row < this->rows() && 0 <= col < this->cols()
    // Returns this's [row][col]
    int get( int row, int col ) const;

  private:
     // add your helper and instance variables here
};
```

Implement the specified constructors, destructor and assignment operators for the `Matrix`. (Some starter code has been provided for you in the file `Matrix.h`, along with a sample executable.) Further, you are to overload the input, output, addition, and multiplication operators as follows:

```
// Creates an empty matrix whose dimensions are 0x0 and the 2-D pointer is
// set to nullptr
Matrix m0;
cout << "m0 = " << m0 << endl; // Outputs "[]"
Matrix m1(2,3); // Creates a 2x3 matrix filled with 0s
m1.set(0,0,1);  // Sets m1[0][0] = 1
Matrix m2(m1);  // Calls the copy constructor to make a deep copy of m1 in m2
Matrix m3;

// Reads in the number of rows, the number of columns, and then the values from
// standard input.
cin >> m3;

// Outputs the matrix that is the sum of m1 and m2 (dimensions must be equal).
cout << "m1 + m2 =\n" << m1 + m2 << endl;

// Outputs the matrix that is the multiplication of m2 by m3, assuming
// their dimensions are compatible.
cout << "m2 * m3 =\n" << m2 * m3 << endl;
```

which produces the following output.

```
$ ./matrix < in.txt
m0 = []
m1 + m2 =    2    0    0
   0    0    0

m2 * m3 =    1    0
   0    0

$ cat in.txt
3 2
1 1 1
2 2 2
```

Some implementation notes follow:

- In order to work more easily with the move operations (move constructor, move assignment operator), we have defined an *empty* matrix as one where the row and column dimensions are both set to 0 and the two-dimensional array pointer is set to `nullptr`. Thus, a matrix whose information is "stolen" becomes an empty matrix. The matrix "stealing" the information must be able to grow or shrink as necessary and not throw an error.
- Use the language features to simplify how you handle reading in the matrix dimensions and values. It should be possible for any white space to be used to separate the numbers (spaces, tabs, newlines, etc.) and have the input operator work properly. You shouldn't need to do anything complex, so don't over-think it.
- When outputting the values of the matrix, an empty matrix produces the string "[]". A non-empty matrix outputs each row's values on a separate line, and sets the width of the value to 4 (see the `setw` operator in the `iomanip` library). You are not required to handle values whose width (including the sign) exceed this size.
- The declaration of the `Matrix` type can be found in the provided `Matrix.h` file. For your submission you must add all requisite declarations to `Matrix.h` and all routine and member definitions to `Matrix.cc`. The public interface to `Matrix` may *not* be changed.
- The provided test harness, `a3q2.cc`, can be compiled with your solution to test (and then debug) your code. **The test harness is not robust and you are not to devise tests for it, just for the `Matrix` class. Do not change this file.** The test harness allows you to have up to 10 matrices defined at one time, identified as `m0` to `m9`. If a matrix has not been initialized, it consists of a `nullptr`. Most of the test harness commands cannot be performed upon an uninitialized matrix, and the harness enforces this. Additionally, the user prompts are printed to standard error so that they will not interfere with the output produced, and thus make it easier to write your test files. The test harness also provides some simple error checking, such as ensuring that get/set are within the bounds of the matrix,and that matrix dimensions are compatible for addition and multiplication. If the commands do not meets its criteria, they are silently ignored. **Thus, you are not required to test these cases and only need to test valid input.**
- Your Makefile must create an executable named `matrix`. Note that the executable name is case-sensitive.

The test harness commands consist of:

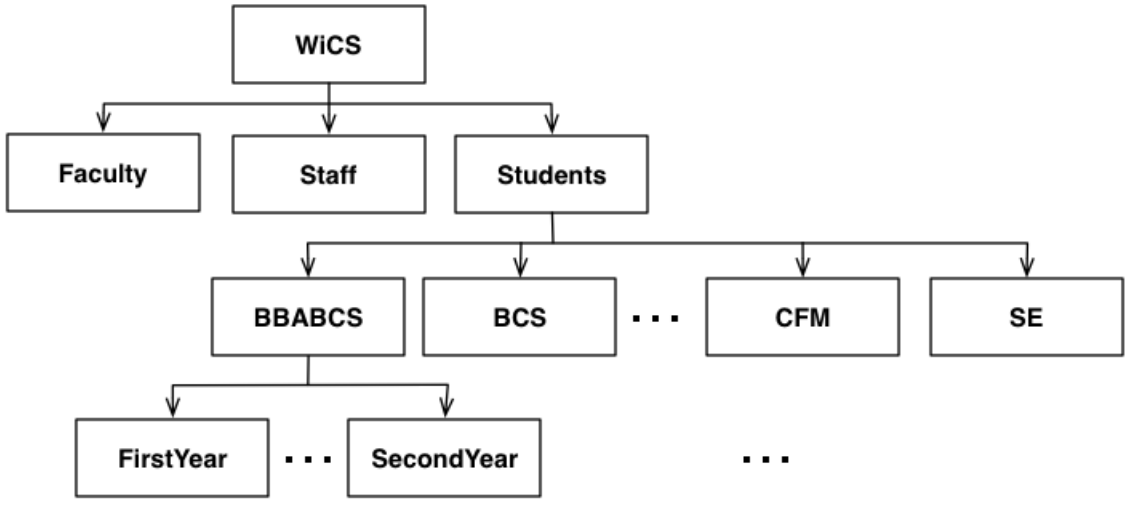
Command	*Description*
c m_i	Initializes matrix m_i by calling its constructor and passing in the dimensions, and then setting its values, where the information is read from standard input. Invokes the matrix destructor if the object previously existed. ~~Outputs m_i using `operator>>` to standard output.~~
C m_i	Invokes the copy constructor to create a temporary matrix, passing in m_i as the parameter. Outputs the temporary matrix and m_i using `operator>>` to standard output. m_i must not be a `nullptr`.
d m_i	Invokes the destructor of matrix m_i. m_i must not be a `nullptr`.
a m_i m_j	Assign copies m_j to m_i ($m_i = m_j$) by performing a deep copy. Outputs m_i and m_j using `operator>>` to standard output. Neither m_i nor m_j must be a `nullptr`.

A m_i m_j	Assign moves m_j to m_i (m_i `= std::move(` m_j `)`) so m_i steals the information from m_j, leaving m_j empty. **Note that this is an alternate form of move assignment to what was presented in class, but is another common version.** `std::move` is used to explicitly mark m_j as an *rvalue*. Neither m_i nor m_j must be a `nullptr`.
r m_i	Uses `operator>>` to initialize matrix m_i by reading the dimensions (row, then column) and value from standard input. Since the operator requires an object, an initially empty matrix is created using the default constructor. Outputs m_i using `operator>>` to standard output. m_i must initially be a `nullptr`.
p m_i	Uses `operator<<` to output matrix m_i to standard output. m_i must not be a `nullptr`.
m m_i	Uses the move constructor to move the contents of m_i to a temporary matrix. Outputs the temporary matrix and m_i using `operator>>` to standard output. m_i must not be a `nullptr`.
s m_i x y z	Set $m_i[x][y] = z$ and output to standard output. m_i must not be a `nullptr`.
g m_i x y	Get $m_i[x][y]$ and output to standard output. m_i must not be a `nullptr`.
+ m_i m_j	Create a temporary matrix that is equal to m_i+m_j if and only if the dimensions of m_i and m_j are the same. Outputs ~~m_i, m_j and~~ the temporary matrix using `operator>>` to standard output. Neither m_i nor m_j must be a `nullptr`.
* m_i m_j	Create a temporary matrix that is equal to $m_i \times m_j$ if and only if the number of columns in m_i equals the number of rows in m_j. Outputs ~~m_i, m_j and~~ the temporary matrix using `operator>>` to standard output. Neither m_i nor m_j must be a `nullptr`.

(a) **Due on Due Date 1**: Design the test suite `suiteq2.txt` for this program and zip the suite into `a3q2a.zip`.

(b) **Due on Due Date 2**: Implement this in C++ and place the files `Makefile`, `a3q2.cc`, `matrix.h` and `matrix.cc` in the zip file, `a3q2b.zip`.

3. As part of an email system, you've been asked to implement a tree-like data structure to represent email groups. In its simplest form, the tree consists of a single node and a single email address[1]. It could also consist of nested groups. For example, the Women In CS (WiCS) mailing list could have a group for the faculty representatives, one for the staff representatives, and then a group for the students, where the students are subdivided by plan, and then by year so that the entire group or particular subsets can be targeted in mass mailings.



[1]Email addresses are just simple strings, and there is no requirement that they constitute a valid email address or follow the format of an email address. For our purposes, any arbitrary string that does not contain whitespace could be a valid email address. You are not required to test for invalid email addresses.

Some implementation notes follow:

- The declaration of the `Group` type can be found in the provided `Group.h` file. For your submission you must add all requisite declarations to `Group.h` and all routine and member definitions to `Group.cc`.
- In order to complete the `Group` implementation, you will also need to implement the nested inner classes: `GroupNode`[2] and `EmailNode`[3] classes. For your submission you must add all requisite declarations to `Group.h` and all routine and member definitions to `Group.cc`. **Note:** `Group` has been declared as a `friend` of both `GroupNode` and `EmailNode` so that its code can access their private information if necessary.
- A `Group` may have 0 or more email addresses, and 0 or more subgroups.
- Searching for an email address first starts in the list of email addresses for the root `Group` node. The search stops as soon as the first occurrence is found. If the address cannot be found there, then each subgroup in turn is searched. Since each subgroup is a `Group`, the search action follows the previously specified order.
- A `Group` can only be deleted if it is a subgroup of the `Group` node currently being indexed; otherwise, the command fails by doing nothing. For example, if the group g_0 is the pointer to the WiCS group as in the diagram, then the WiCS group will *not* be removed from g_0.
- Note that the information in each list is stored in the standard `string` lexicographic[4] order using the standard `string` comparison operators. Thus, the output will be in lexicographic order.
- **It is *strongly* suggested that you first implement and test your linked list code before you work on the rest to ensure that it is correct.**
- The provided test harness, `a3q3.cc`, can be compiled with your solution to test (and then debug) your code. **The test harness is not robust and you are not to devise tests for it, just for the `Group` class. Do not change this file.** The test harness allows you to have up to 10 groups defined at one time, identified as `g0` to `g9`. If a group has not been initialized, it consists of a `nullptr`. Most of the test harness commands cannot be performed upon an uninitialized group. Additionally, the user prompts are printed to standard error so that they will not interfere with the output produced, and thus make it easier to write your test files.
- Your Makefile must create an executable named `emailgroups`. Note that the executable name is case-sensitive.

The test harness commands consist of:

[2] Linked list of `Group` pointers, used to contain subgroups to the current group.

[3] Linked list of email addresses, implemented as `string` objects embedded in the nodes.

[4] In lexicographic order, the string "2" comes after the string "11" since they are not treated as numbers when the comparison is performed. As per usual, the string "cat" comes before "dog" in lexicographic ordering.

Command	*Description*
b g_i *name*	Initializes group g_i by calling its constructor and passing in the group name, *name*. g_i must initially be a `nullptr`.
aa g_i *email*	Uses `Group::addAddress` to add *email* to g_i. g_i must not be a `nullptr`.
ag g_i g_j	Uses `Group::addGroup` to add g_j to g_i and sets g_j to `nullptr`. Neither g_i nor g_j must be a `nullptr`.
ra g_i *email*	Uses `Group::removeAddress` to remove the first occurrence of *email* from g_i. g_i must not be a `nullptr`.
rg g_i *name*	Uses `Group::removeGroup` to remove the first subgroup of g_i that has a name that matches *name*. g_i must not be a `nullptr`.
sa g_i *email*	Uses `Group::findAddress` to return an `Group::EmailNode*` set to the node that contains the first occurrence of *email* in g_i or `nullptr` if no such address can be found. g_i must not be a `nullptr`.
sg g_i *name*	Uses `Group::findGroup` to return a `Group::GroupNode*` set to the node that contains the first occurrence of *name* in g_i as a *subgroup* or `nullptr` if no such subgroup can be found. g_i must not be a `nullptr`.
p g_i	Uses `operator<<` to output group g_i to standard output. g_i must not be a `nullptr`.

(a) **Due on Due Date 1**: Design the test suite `suiteq3.txt` for this program and zip the suite into `a3q3a.zip`.

(b) **Due on Due Date 2**: Implement this in C++ and place your `Makefile`, `a3q3.cc` and all `.h` and `.cc` files that make up your program in the zip file, `a3q3b.zip`.