CS106B
Sprint 2017 April 27th, 2017

CS106B Practice Midterm (KEY)

This is an open-note, open-book exam. You can refer to any course handouts, textbooks, handwritten lecture notes, and printouts of any code relevant to any CS106B assignment. You may not use any laptops, cell phones, or internet devices of any sort. You will be graded on functionality—but good style helps graders understand what you were attempting. You do not need to #include any libraries and you do not need to forward declare any functions. You have 2 hours. We hope this exam is an exciting journey.

Last Name: _____

First Name: _____

Sunet ID (eg jdoe): _____

Section Leader: _____

I accept the letter and spirit of the honor code. I've neither given nor received aid on this exam. I pledge to write more neatly than I ever have in my entire life.

(signed) _____

		Score	Grader
1. Tracing Functions	[15]	_____	_____
2. Blammo	[15]	_____	_____
3. Grade Histogram	[15]	_____	_____
4. Autonomous Art	[15]	_____	_____
Total	**[60]**	**_____**	_____

Question 1: Tracing and Big O (15 Points)

Assume that the following **mystery** and **enigma** functions have be defined as follows:

```
int mystery(int n) {
    if (n == 0) {
        return 0;
    } else {
        return mystery(n-1) + enigma(n) + enigma(n);
    }
}

int enigma(int n) {
    int index = 0;
    int sum = 0;
    while (index < 2*n) {
        sum++;
        index++;
    }
    return sum;
}
```

a. **[2 Points]** What is the value of enigma(2)?

   Your answer does not need to be simplified (e.g. 1 + 12 + 23 is an acceptable form for an answer).

   4

b. **[4 Points]** What is the value of mystery(3)?

   Again, your answer does not need to be simplified.

   2*(2*3 + 2*2 + 2*1) + 0 = 24

c. **[4 Points]** What is the worst case runtime of enigma expressed in big-O notation, where N is the value of the argument n?

   You may assume that n is a nonnegative integer.

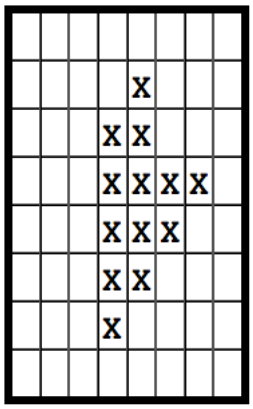
   O(n)

d. **[5 Points]** What is the worst case runtime of mystery expressed in big-O notation, where N is the value of the argument n?

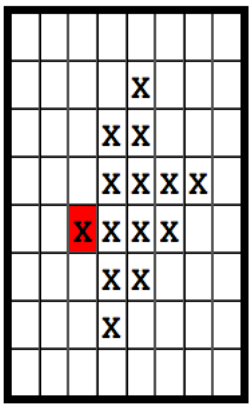
   You may assume that n is a nonnegative integer. **$O(n^2)$**

Question 2: Blammo (15 Points)

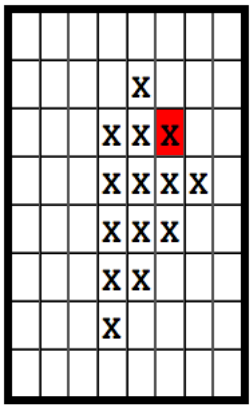
This problem is about a board game called Blamo, a game in which players take turns placing tiles on a grid and getting points based on the length of horizontal and vertical chains formed by each move. For instance, the following is a possible state of the Blamo board (where X indicates the location of a tile):



A move involves placing a single tile on the board. The score of that move is equal to the sum of the lengths of the horizontal and the vertical chains of tiles that are adjacent to the tile placed. For example, consider a move consisting of placing a tile in the shaded cell of the board below:



This move would be worth 5 points: 4 from the horizontal chain of 4 tiles, and 1 from the vertical chain of 1 tile. A better move to make would be this:



This move is worth 6 points: 3 for the vertical chain of 3 tiles, and 3 for the horizontal chain of 3 tiles.

Your job is to write the following function, which calculates the score associated with a valid move:

```
int computeScore(Grid<bool>&board, int row, int col);
```

which takes as input **board** which represents the state of a Blamo board, and **row** and **col** which corresponds to the location of the move on board. The function computeScore() calculates the score of a Blamo move consisting of placing a single tile on the board at the position (row,col) **board** represents the state of the Blamo board such that:

- board[y][x] (where x and y are integers) is true if there is a tile at row y and column x of board and false if if there is a not a tile there.

row and col represent the position on board that a tile was placed. You may assume row and col are within the bounds of board.

Please put your answer to question 2 here:

```
int computeScore(Grid<bool>&board, int row, int col) {
    for (int dRow = -1; dRow <= 1; dRow++) {
        for (int dCol = -1; dCol <= 1; dCol++) {
            if (abs(dRow) + abs(dCol) == 2 ||
                (dRow == 0 && dCol == 0)) continue;
            int currRow = row + dRow;
            int currCol = col + dCol;
            while (board.inBounds(currRow,currCol) &&
                   board[currRow][currCol]) {
                currRow += dRow;
                currCol += dCol;
                score++;

            }
        }
    }
    return score;
}
```

Notes: It isn't clear from the problem whether or not the **board[row][col]** is **true** or **false** (i.e. whether or not the move has already been made) – students are welcome to make either assumption as long as they are consistent.

Question 3: Grade Distribution Histogram (15 points)

A common way to visualize how a class of students perform on exams is by using a histogram, which provides an estimate of the probability distribution of the grades for the exam. For example, given the following scores on an exam, we can draw the histogram (shown to the right), which represents how many students received grades in the 60s, 70s, 80s, and 90s.

Student	Grade
StudentA	97
studentB	89
studentC	93
studentD	75
studentE	94
studentF	85
studentG	88
studentH	68
studentI	79
studentJ	84

```
Histogram:
60s: *
70s: **
80s: ****
90s: ***
```

A histogram can also be used to determine the distribution of grades for an entire quarter, based on an average of each student's grades.

Consider the following map which associates student names with a vector of their grades for the quarter. We would like to produce a **histogram of student averages**. In other words, **average each student's grades, then produce a histogram of the averages.** The histogram for the averages is shown to the right:

Student	Grade	Average
studentA	97, 92, 88	92.3
studentB	89, 93, 77	86.3
studentC	93, 95, 105	97.7
studentD	75, 25, 50	50
studentE	94, 94, 94	94
studentF	85, 82, 73	80
studentG	88, 91, 99	92.7
studentH	68, 78, 88	78
studentI	79, 85, 77	80.3
studentJ	84, 85, 86	85

```
Histogram of Averages:
50s: *
70s: *
80s: ****
90s: ****
```

Given a map of student names (strings) as keys, and a vector (ints) to each student's scores, your job is to write the following three functions:

[4 points]

```
// Returns the average value of a vector of integers.
// Assumes there is at least one grade.
double average(Vector<int> & gradeVec) {

    // assumption: there is at least one grade
    double sum = 0;
    for (int grade : gradeVec) {
        sum += grade;
    }
    return sum / gradeVec.size();

}
```

[7 points]

```
// Produce a map of average grade distributions, grouped by
// tens (e.g., if 8 people scored an average in the 90s, there
// would be a key in the map for 90, and its value would be 8)
void histogram(Map<string,Vector<int>> & grades,
               Map<int,int> & hist) {

    for (string key : grades) {
        int avg = (int)(average(grades[key]))/10 * 10;
        hist[avg]++;
    }

}
```

Question 3: Grade Distribution Histogram (continued)

[4 points]

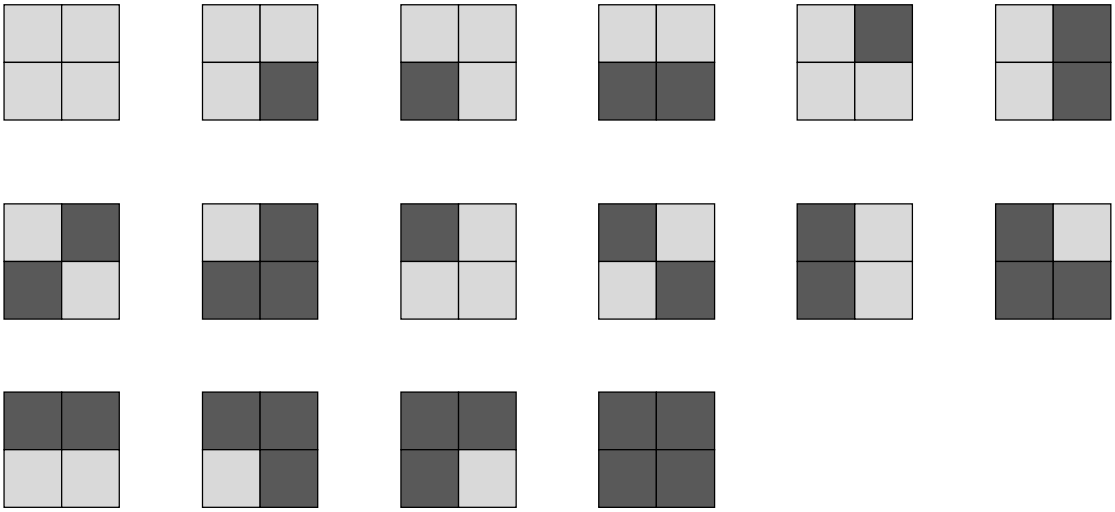
```
// Print a histogram in the following form:
// 50s:***
// 60s:*****
// 70s:**
// 80s:***
// 90s:******
//
// For the example above, the map holds the
// following key/value pairs: {50:3, 60:5, 70:2, 80:3, 90:6}
// Assume that keys and values are positive and that keys are
// multiples of ten.
void printHistogram(Map<int,int> & hist) {



    // print the grade then a line of asterisks for the total
    for (int key : hist) {
            cout << key << "s:";
            for (int i=0; i < hist[key]; i++) {
                cout << "*";
            }
            cout << endl;
    }









}
```

Question 4: Autonomous Art (15 points)

If we could generate all images presumably we would recreate the Mona Lisa and generate novel artwork!

Write a function **`genImages`** that generates and saves all possible images of a given width and height using a fixed palette of colors.

As an example, for a picture that is 2 pixels by 2 pixels made up of colors light-grey and dark-grey there are 16 possible images. Here are all 16 possibilities:



Each picture is represented as a `Grid<int>`.

The input to your **`genImages`** function is the target size of the image (the number of rows and columns in the underlying grid) and a vector which contains all colors that can be used in the picture.

Your function should work for **any** positive grid dimensions (`rows` and `cols`) and **any** vector of colors with size greater than 0.

Your function should save each image using a call to a helper method **`saveImage`** that we provide and that you ***don't*** have to write:

```
void saveImage(Grid<int> & image);
```

Hint: The color of each pixel is a decision. Come up with a recursive helper function that can explore all possible combinations of decisions.

```
// note: there are many different ways to solve this problem.
void genImages(Vector<int>& colors, int rows, int cols){
    Grid<int> image(rows,cols);
    genHelper(image, colors, 0);
}

void genHelper(Grid<int>& image,
               Vector<int>& colors, int next) {

    // base case, all colored
    if (next == image.numRows() * image.numCols()) {
       saveImage(image);
       return;
    }
    // recursive case, if not finished
    int row = next / image.numCols();
    int col = next % image.numCols();

    for (int color : colors) {
        image[row][col] = color;
        genHelper(image, colors, next + 1);
    }
}

// another possible solution:
void genImages(Vector<int>& colors, int rows, int cols){
    Grid<int> image(rows,cols);
    genHelper(image, colors, 0, 0);
}

void genImagesHelper(Grid<int> &image,
                     Vector<int>& colors,
                     int currRow, int currCol) {

    // base case: reached end of rows = done
    if (currRow = grid.numRows()) {
        saveImage(image);
    } else if (currCol == grid.numCols()) {
        // recursive case: go to next row
        genImagesHelper(image, colors, currRow+1, 0);
    } else {
       for (int c : colors) {
        image[currRow][currCol] = c;
        genImagesHelper(image, colors, currRow, currCol + 1);
       }
}
```