CS106B
Fall 2016

Chris and Chris
Nov 3rd, 2016

CS106B 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. Destiny's Trace	[12]	______	______
2. Grade Histogram	[15]	______	______
3. Rotate Image	[18]	______	______
4. Autonomous Art	[15]	______	______
Total [60]		______	______

Question 1: Destiny's Trace (12 points)

```
int beyonce(int n) {
   int sum = 0;
   while(n > 0) {
      sum += kelley(n);
      n /= 2;
   }
   return sum;
}

int kelley(int m) {
   int sum = 0;
   for(int i = 0; i < m; i++) {
      for(int j = 0; j <= i; j++) {
         michelle(i, sum);
      }
   }
   return sum;
}

void michelle(int n, int & sum) {
   for(int j = 0; j < 3; j++) {
      sum += n;
   }
   sum++;
}
```

(a) [3 points] What is the result of **`kelley(3)`**?

`kelley(3) returns the value 30`

(b) [3 points] What is the result of **`beyonce(3)`**?

`beyonce(3) returns the value 31`

(c) [2 points] What is the computational complexity of the **`michelle`** function expressed in terms of big-O notation, where N is the value of the parameter `n`?

`O(1)`

(d) [2 points] What is the computational complexity of the **`kelley`** function expressed in terms of big-O notation, where N is the value of the parameter `m`?

$O(n^2)$

(e) [2 points] What is the computational complexity of the **`beyonce`** function expressed in terms of big-O notation, where N is the value of the parameter `n`?

$O(\log n\ n^2)$ is correct (full credit)

$O(n^2)$ is even better and we gave extra credit if you showed the geometric series sum explanation.

Question 2: 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 2: 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 3: Rotate Image (18 points)

In the Fauxtoshop assignment, you were asked to write a number of filters for images stored in a `Grid<int>`. Almost all photo-manipulation programs also have a function that rotates images, and the user can type in an angle to rotate the image. One way to rotate an image clockwise around its center point is as follows:

For each pixel in the rotated image at coordinate (x, y), copy the color of the pixel from the original image closest to (xOld, yOld) such that:

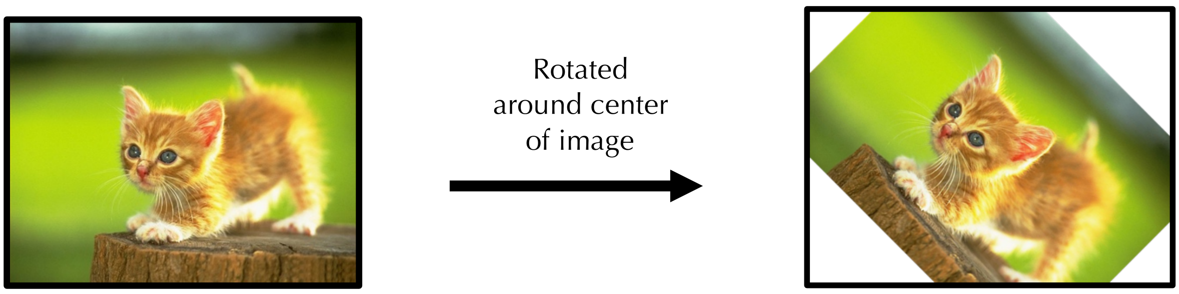
$$\text{xOld} = +\, x \cdot \cos(\theta) + y \cdot \sin(\theta)$$
$$\text{yOld} = -\, x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

Where:

- θ is the angle of rotation.
- x coordinates represent number of pixels to the **right** of the image center (or, if negative, the number of pixels to the left of the center).
- y coordinates represent number of pixels **below** the image center (or, if negative, the number of pixels above the image center).

You will need to translate between rows and columns and x and y coordinates. As an example, the grid cell (row = height/2, col = width/2) translates to coordinate (x = 0, y = 0).

One thing to consider when rotating a non-circular image is that some of the rotated pixels will not fit in the grid. Think about rotating the following rectangle by 45º:



Additionally, part of the new image will not receive any rotated pixels. In this case, we will make those pixels white (suggestion: make the new grid completely white to start).

Using the rotation equations above, complete the two functions: (1) **`makeWhite`** which populates a `Grid<int>` with white, and (2) **`rotateImage`** which rotates an image by `angle` number of degrees

You can use the following helper functions:

Function	**Return value**
`double` **`cos`**`(x)`, where `x` is in degrees.	The cosine of `x`
`double` **`sin`**`(x)`, where `x` is in degrees.	The sine of `x`
`int` **`round`**`(x)`	Mathematical rounding.

[3 points]

```
const int WHITE = 0xFFFFFF;

// makes every pixel in the given grid white.
void makeWhite(Grid<int> & image) {


    for (int r=0; r < grid.numRows(); r++) {
        for (int c=0; c < grid.numCols(); c++) {
            grid[r][c] = WHITE;
        }
    }


}
```

[15 points]

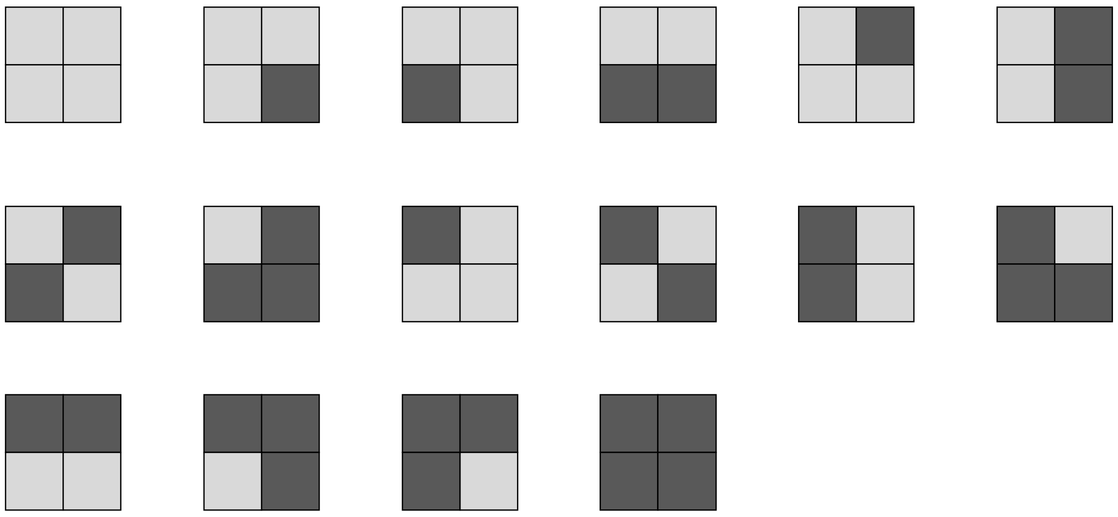
```
// modify the grid so that it represents an image
// rotated by theta number of degrees.
void rotateImage(Grid<int>& image, double theta) {

   Grid<int> newGrid(grid.numRows(),grid.numCols());

   double midC = (grid.numCols()-1) / 2.0;
   double midR = (grid.numRows()-1) / 2.0;

   makeWhite(newGrid);

   for (int r=0; r < grid.numRows(); r++) {
      for (int c=0; c < grid.numCols(); c++) {
         // calculate offset locations to rotate around middle
         double x = c - midC;
         double y = r - midR;

         double xOld = x * cos(degrees)
               + y * sin(degrees);
         double yOld = -x * sin(degrees)
               + y * cos(degrees);

         // undo the column offset
         int cOld = (int)(round(xOld + midC));

         // undo the row offset
         int rOld = (int)(round(yOld + midR));

         if (grid.inBounds(rOld,cOld)) {
            newGrid[r][c] = grid[rOld][cOld];
         }
      }
   }
   grid = newGrid;

}
```

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.

```
void genImages(Vector<int>& colors, int rows, int cols){
  Grid<int> grid(rows, cols);
  helper(colors, grid, 0, 0);
}

void helper(Vector<int>& colors, Grid<int>& grid,
     int r, int c) {

  // base case
  if(!grid.inBounds(r, c)) {
    saveImage(grid);
  } else {

    // calculate next pixel position
    int nextR = r;
    int nextC = c + 1;
    if(nextC == grid.nCols) {
      nextR = r + 1;
      nextC = 0;
    }

    // try each color choice and recurse from next pos.
    for(int color : colors) {
      grid[r][c] = color;
      helper(colors, grid, nextR, nextC);
    }
  }
}
```

}