Homework 3

UML State Chart Diagrams

Here is a specification for a ScoreAverager class. Use this specification for the problems that follow.

Operations (all public):

ScoreAverager(); // constructor

// postcondition: getNumScores()’ == 0 and getAverage()’ == 0

void addScore(double score);

// precondition: score >= 0

// postcondition:

// getNumScores()’ = getNumScores() + 1

// getAverage()’ = (getAverage() \* getNumScores() + score) / (getNumScores()’)

double getAverage();

// precondition: getNumScores() >= 0

// postcondition: No state change.

int getNumScores();

// postcondition: No state change.

Part1:

1. For the postcondition of addScore(), is it possible to raise a divide-by-zero error? Make sure to look at the pre- and post-conditions for ALL methods. Explain why/why not.

No. Assuming the precondition for add score refused to accept negative numbers. numScores is 0 after creation. If addScore is called then numScores increases by 1. therefore no divide by – error will occur.

1. Can an implementation of getNumScores() always return 0? Again, look at all the pre- and post-conditions. Explain why/why not.

Yes. It can return 0 if no scores are ever added to ScoreAverager. If any scores are added numScores will be > 0. in any noramly implementation Numscores will not always be 0

1. Can an implementation of getAverage() always return 0? Look at all the pre- and post-conditions. Explain why/why not.

Yes. GetAverage will return 0 so long as all scores added are 0. assuming no scores have been added, scores =0 and getnumSCores+1=1. 0/1 =0. It will also return 0 if no scores are ever added. In a normal implementation avgScore will not always be 0

1. When a caller calls addScore(double score) with a score<0, the implementation causes getAverage() and getNumScores() to both return 0 after addScore() finishes executing. Does this implementation obey the pre- and post-conditions given above? Explain why/why not.

No. the precondition for addScore requires the score to be >=0. to add a negative score is a direct violation of this precondition. This implementation will cause ScoreAverager to reset whenever an invalid score is given.

Part 2:

Which of the following possible implementations obey the specification for ScoreAverager? Answer YES or NO for each, and briefly explain why. If there are multiple violations of the specification, briefly describe each violation.

Don’t forget that where the spec is silent (doesn’t specify what must happen), the implementer can choose whatever behavior they think is reasonable.

1. Implement ScoreAverager in C++, using private instance variables: double \_avgScore, int \_numScores.
   * The constructor sets both of these to 0.
   * Method getNumScores() returns \_numScores.
   * If \_numScores > 0, method getAverage() returns \_avgScore; otherwise it returns 0.
   * Method addScore() does nothing if score is < 0. If score >= 0, it increments \_numScores then sets \_avgScore = (\_avgScore\*(\_numScores-1) + score) / \_numScores.

Yes. The initializing conditions are met. AddScore handles values so that no post conditions are met if the precondition isn’t met. GetNumScores follows implementation. Get average follows implementation. Add score correctly changes numScores and avgScore so getAverage will return the correct value.

1. Implement ScoreAverager in C++, using private instance variables: double \_avgScore, int \_numScores.
   * The constructor sets \_numScores to 0 and sets \_avgScore to 100.
   * Method getNumScores() returns \_numScores.
   * If \_numScores > 0, method getAverage() returns \_avgScore; otherwise it returns 0.
   * Method addScore() does nothing if score is < 0. If score >= 0, it increments \_numScores then sets \_avgScore = (\_avgScore\*(\_numScores-1) + score) / \_numScores.

Yes. GetNumScores() follows implmentation. GetAverage() follows implementation and has error checking for when no scores have beena dded yet. AddScore() works properly, when the first score is added, whatever was set to \_avgScore is erased and the new avgScore is based off of the first input. The accepting of negative scores is only allowed for error handling.

1. Implement ScoreAverager in C++, using private instance variables: double \_avgScore, int \_numScores.
   * The constructor sets both of these to 0.
   * Method getNumScores() returns \_numScores.
   * If \_numScores > 0, method getAverage() returns \_avgScore; otherwise it returns 0.
   * Method addScore() does the following. If score >= 0, it increments \_numScores then sets \_avgScore = (\_avgScore\*(\_numScores-1) + score) / \_numScores. If score < 0, it decrements \_numScores then sets \_avgScore = (\_avgScore\*(\_numScores+1) + score) / \_numScores.

No. getAverage and getNumScores works as intended. AddScore() works when score>=0 and when <0 essentially undoes added a score of that value. This doesn’t work because if a score of 50 is added and then a score of -50 is added then a divide by 0 error occurs.

1. Implement ScoreAverager in C++, using private instance variables: double \_sumScores, int \_numScores.
   * The constructor sets \_numScores to 0 and sets \_sumScores to 100.
   * Method getNumScores() returns \_numScores.
   * If \_numScores > 0, method getAverage() returns \_sumScores/\_numScores; otherwise it returns 0.
   * Method addScore() does nothing if score is < 0. If score >= 0, it increments \_numScores then sets \_sumScores = \_sumScores + score.

No. get average will not return an accurate average if any scores are added. It will only be accurate if no scores are added.

1. Implement ScoreAverager in C++, using private instance variables: double \_sumScores, int \_numScores.
   * The constructor sets both of these to 0.
   * Method getNumScores() returns \_numScores.
   * If \_numScores > 0, method getAverage() returns \_sumScores/\_numScores; otherwise it returns 0.
   * Method addScore() increments \_numScores then sets \_sumScores = \_sumScores + score.

Yes. GetAverage and getNumScores works as intended when scoreAverager is empty. After 1 score is added, \_sumScores is that score, \_numScores is 1, so average is that score. However many scores are added directly correlates to the \_numScores. The average of any set is just the sum of each element divided by the length of that set. AddScore adds to its sum cumulatively and then divides by the number of scores that were added.

To be honest this might be a no only because the specifications say that the math for getting averages should be done in addScore() and this implementation does the math in getAverage() instead.

Part 3:

Draw a State Chart Diagram describing the interaction protocol of a ScoreAverager.

