# Part 1: Requirements Grouping

## ui.h

The file ui.h will be responsible for the following requirements:

* (120) The program will prompt the user with a problem like: “How much is 6 times 7?” 6.57
* (130) The program will allow the user to input the answer to the problem. 6.57
* (150) If the user inputs the correct answer, the program will output a message: “Very good!” 6.57
* (160) The program will randomly select the message for a correct answer from: “Very good!”, “Excellent!”, “Nice work!”, and “Keep up the good work!” 6.58
* (170) If the user inputs an incorrect answer, the program will output a message: “No. Please try again!” 6.57
* (180) The program will randomly select the message for an incorrect answer from: “No. Please try again!”, “Wrong. Try once more.”, “Don’t give up!”, and “No. Keep trying. 6.58
* (220) After the student has attempted 10 problems, the program will display the percentage of correct answers given 6.59
* (230) If the percentage of correct answers is <75%, the program will also display: “Please see your teacher for extra help.” 6.59
* (240) If the percentage of correct answers is >=75%, the program will also display: “Congratulations, you are ready to go on to the next level!” 6.59
* (320) The User Interface object for this program is to be encapsulated, and designed to be independent of whether the program is being implemented as a console application or a GUI Windows application. The intent is that the main program and other processing functions will not directly access the screen and keyboard (no **printf**, **cin**, or **cout**), but will display problems or obtain input using calls to a UI object. If this is done correctly, it will be possible to create the program including either the Console or the GUI version of the UI object, with no changes to the other objects or main program code.
* (330) The program will maintain multiple student records during its run, saving each student’s name, number of problems attempted, and number of problems answered correctly. Up to 100 student records may be stored during the run of the program. This means the student’s name must be entered and saved when the studentstarts the problem-solving process.
* (340) The data for all students will be displayed with name, number attempted, and percent correct when the ‘Display scores’ option is selected. For this assignment, no security will be imposed to prevent access to the scores.

## question.h

The file question.h will be responsible for the following requirements:

* (110) The program will use rand to produce two 1-digit numbers. 6.57
* (140) The program will check the student’s answer. 6.57
* (190) If the user inputs an incorrect answer, the program will allow 1 additional attempt to answer correctly; if they answer on the second try, it will count as 0.5 correct for the scoring purposes. 6.57, with modifications.
* (200) Problem generation is to be through a separate function call; this function is called each time a new problem is needed (such as, after a correct answer is entered). 6.57
* (260) The program will allow the student to select a difficulty level from 1 to 3 before the problems are presented. 6.60
* (270) The difficulty level selected determines the maximum number of digits used in each operand of the problem presented. Thus, a level of ‘3' will generate problems where the operands may be 1, 2, or 3 digits, randomly selected. 6.60
* (280) The program will allow the user to select the math operation to be used in the problem generation, with a selection of ‘1' being ‘addition’ problems, ‘2' for subtraction, ‘3' for multiplication, ‘4' for division, and ‘5' being a random mix of any of the 4 operations. 6.60
* (290) For division problems, only whole number quotients will be generated by randomly generating the quotient and divisor, then multiplying them to get the dividend (thing to be divided). The user is presented with the dividend and divisor, and is asked to find the quotient.
* (300) For division problems, the definition of ‘difficulty levels’ will be:Level 1 will have 1-digit quotients and divisors; Level 2 will allow Level 1problems, plus problems with 2-digit quotients, and 1-digit divisors; Level 3 will allow Level 2 problems, plus 2-digit divisor and quotient problems.

## progress.h

The file progress.h will be responsible for the following requirements:

* (190) If the user inputs an incorrect answer, the program will allow 1 additional attempt to answer correctly; if they answer on the second try, it will count as 0.5 correct for the scoring purposes. 6.57, with modifications.
* (210) The program will count the number of correct and incorrect responses given by the student. 6.59
* (220) After the student has attempted 10 problems, the program will display the percentage of correct answers given 6.59
* (250) After the student has attempted 10 problems, and displayed the percentage and message, it will reset the counts to allow this student or another student to start over. 6.59
* (330) The program will maintain multiple student records during its run, saving each student’s name, number of problems attempted, and number of problems answered correctly. Up to 100 student records may be stored during the run of the program. This means the student’s name must be entered and saved when the student starts the problem-solving process.
* (340) The data for all students will be displayed with name, number attempted, and percent correct when the ‘Display scores’ option is selected. For this assignment, no security will be imposed to prevent access to the scores.

## Main Program

The main function will be responsible for the following requirements:

* (100) The purpose of the program is to help elementary students learn multiplication (math) 6.57
* (190) If the user inputs an incorrect answer, the program will allow 1 additional attempt to answer correctly; if they answer on the second try, it will count as 0.5 correct for the scoring purposes. 6.57, with modifications.
* (250) After the student has attempted 10 problems, and displayed the percentage and message, it will reset the counts to allow this student or another student to start over. 6.59
* (260) The program will allow the student to select a difficulty level from 1 to 3 before the problems are presented. 6.60
* (310) The program is to be designed using an OOP approach, resulting in a **minimum** of 3 different classes, to fulfill all of the requirements stated.

# Part 2: Classes

The following classes are part of the object-oriented design of this project:

* “ui” – This class is responsible for all of the user interaction, including outputting information to the console and pulling input from the user. Everything the user sees has to go through this class.
* “question” – The question class is responsible for the generation of new questions, the validation of answers, and adapting the questions based on the level of the questions.
* “progress” – This class is responsible for tracking the user’s scores and progress. It will also set the user’s name and store a copy of the level. There can be up to 100 progress objects.

# Part 3: Requirement-to-Function Mapping

The following is the function to requirement mapping of the program()

|  |  |
| --- | --- |
| **Function Name** | **Requirements Met** |
| ui::genHeader() | n/a |
| ui::clearConsole() | n/a |
| ui::askStudentName() | 330 |
| ui::askQuesiton() | 120 |
| ui::getAnswer() | 130 |
| ui::answerEchoResult() | 150, 160, 170, 180 |
| ui::echoStudentScores() | 220, 230, 240, 330, 340 |
| private question::rand() | 110, 260, 270, 290, 300 |
| question::generateQuestion() | 120, 200, 260, 280 |
| question::checkAnswer() | 140, 190 |
| question::setLevel() | 260, 270, 300 |
| question::getNumOne() | 110 |
| question::getNumTwo() | 110 |
| progress::progress() (constructor) | n/a |
| progress::answerTrue() | 210 |
| progress::answerFalse() | 190, 210 |
| progress::resetStats() | 250, 330 |
| progress::getNumTrue() | 220, 340 |
| progress::getNumFalse() | 220, 340 |

# Part 4: C++ Class Definitions

The following are the C++ class definitions.

// File: UI.h

class ui{

void genHeader();

// Generates a welcome and instructions for the

// user

// Requirements: N/A

void clearConsole();

// Clears the console using a while loop and endl

// Requirements: N/A

string askStudentName();

// Asks for the student’s name and returns it in the

// form “<first name> <last name>”

// Requirements: 330

void askQuestion(int, int, bool);

// Asks the student a question

// Inputs: first number, second number, second attempt

// Requirements: 120

int getAnswer();

// Gets the answer from a student using cin

// Returns the answer given

// Requirements: 130

void echoStudentScores(&progress[]);

// Outputs the student scores from each item of the

// array to the screen

// Requirements: 220, 230, 240, 330, 340

void answerEchoResult(bool);

// Outputs the text indicating the result to the user

// Input: Whether the answer is true or false

// Requirements: 150, 160, 170, 180

};

// File: question.h

class question {

private:

int rand();

// Wrapper for the random function, allows the

// creation of requirement-fitting random numbers

// Requirements: 110, 260, 270, 290, 300

public:

void generateQuestion();

// Generates a new question for the student to work on

// Requrements: 120, 200, 260, 280

bool checkAnswer(int);

// Checks the answer against the currently stored

// question

// Requrements: 140, 190

bool setLevel(int);

// Changes the level of the questions generated

// Requrements: 260, 270, 300

int getNumOne();

// Returns the first number of the question

// Requirements: 110

int getNumTwo();

// Returns the second number of the question

// Requirements: 110

};

// File: progress.h

class progress {

progress();

// Constructor: Sets the variables

// Requrements: N/A

void answerTrue();

// Increments the internal true counter

// Requrements: 210

void answerFalse();

// Increments the internal false counter

// Requrements: 190, 210

void resetStats();

// Reset the stats for the new user

// Requrements: 250, 330

int getNumTrue();

// Returns the number of answers that the user has

// answered right

// Requirements: 220, 340

int getNumFalse();

// Returns the number of answers that the user has

// answered wrong

// Requirements: 220, 340

};

# Part 5: Main program pseudo-code

Here is the psudo-code for the main program:

int main() {

Generate Main Menu

while(“not the menu selection for exit”) {

Ask the student for their name

Store their name in a new progress object

Generate a new question

while(number of questions < 10) {

Output the question to the screen

Get the user’s answer

Check their answer against the stored answer

if (they’re right) {

Increment the true counter

Generate a new question

} else {

Increment the false counter

If (this is their first try) {

Set a flag for another try

} else {

Generate a new question

}

}

}

Exit gracefully

}