Program #3

Due Oct 21 by 11:59pm **Points** 100

Making Algebra Fun

Introduction

For this homework project you will have two weeks to complete the programming portion of the assignment. **However, you will be required to submit a design document to Canvas (due on Sunday, October 14th, 11:59pm).** You can use the same documentation template and format that was introduced during week 2. Please review the **recitation syllabus (https://web.engr.oregonstate.edu/~goinsj/resources/CS161/CS161_Recitation_Syllabus.pdf)** (available from the Recitations tab) for additional details on the design document expectations.

Example design document (introduced in week 2): **Polya_template.pdf (https://web.engr.oregonstate.edu/~goinsj/resources/CS161/Polya_template.pdf)**

As noted last week, design is very important when developing programs and there are a variety of ways to approach it. You may draw pictures, write it out in prose or structured text, use pseudo code, and more! The point of design is to give you a blueprint to follow while you are coding. This saves time debugging your program as you can catch mistakes early. It is better to spend one hour of designing than it is to spend five hours debugging.

For this assignment you must design a solution to the following problem statement and implement your solution.

(85 points) Problem Statement

We have been tasked to implement a program/application that makes learning algebra fun! This program will challenge the student to solve math problems. In our game, the student is in an elevator and trying to get all the way to the top floor of a building. They can reach the top (and hopefully receive the gold prize) by solving each equation correctly and moving up or down floors based on their answer!

You will ask the student how many floors there are in the building to start, and whether the student wants to play on easy or difficult mode. The student will then be given an equation to solve. If the student is playing in easy mode then they get to choose which type of equation to solve (addition, multiplication, subtraction or division). If the user is playing in difficult mode then your program will randomly select one of the possible equation types (addition, multiplication, subtraction or division).

When displaying an equation for the user to solve, you will generate one random number for the expression and one number for the solution/right side of the equation (see the example program output for clarification). If the user is playing in easy mode, the generated numbers will be between 0-100 (inclusive). If the student is playing in difficult mode, the numbers will instead be between -100 to +100 (inclusive). When the student solves the equation correctly, they get to go up one floor closer to the gold. If the user solves the equation incorrectly, then they go down one floor.

After the student answers the equation, you must take action. First, you will display the current position of the user (in the building). Assuming the user has not won the game, you will begin the process of generating another equation to solve. If the user is playing in easy mode, prompt the student to choose a type of equation for the next problem (addition, multiplication, subtraction, or division). If the user is playing in difficult mode, they don't get to pick the type of problem and your problem will randomly assign one. This process repeats until the student eventually reaches the top floor. After the student reaches the gold, you must give the student an opportunity to play again and repeat the program.

You must always inform the user which level of the building they are on with a message and ASCII art (search the internet for inspirational examples of ASCII art). You must insert a blank line between re-prompting the student with a new equation to solve. If the student wins and wants to play again, you must ask how many floors are in the new building (and whether the user wants to play in easy or difficult mode).

Example Operation

Your code does not need to run exactly like this but it needs to meet the requirements as outlined in this document.

```
What difficulty would you like to play? Easy (1), Hard (2): 1
How many floors are in your building? 3
Welcome! There's a treasure chest on the roof of the building. Hurry to reach the top!
|       |
----------
|       |
----------
|    X  |
----------

Do you want to solve a multiplication (1), addition (2), subtraction (3), or division (4) equation? 1
Please solve the following equation:
x * 11 = 88, what is x? 8
Good job, you are on level 2!
|       |
----------
|    X  |
----------
|       |
----------

Do you want to solve a multiplication (1), addition (2), subtraction (3), or division (4) equation? 4
Please solve the following equation:
```

```
x / 50 = 90, what is x? 4500
Good job, you are on level 3!
|    X  |
----------
|       |
----------
|       |
----------

You win the game!!!

Do you want to play again (0-no, 1-yes)? 1

What difficulty would you like to play? Easy (1), Hard (2): 2
How many floors are in your building? 3
Welcome! There's a treasure chest on the roof of the building. Hurry to reach the top!
|       |
----------
|       |
----------
|    X  |
----------

Please solve the following equation:
x * -10 = 50, what is x? -5
Good job, you are on level 2!
|       |
----------
|    X  |
----------
|       |
----------

Please solve the following equation:
x / -50 = 80, what is x? 40
Oh no, that was wrong. You are now at level 1.
|       |
----------
|       |
----------
|    X  |
----------

Please solve the following equation:
x + -99 = -24, what is x? 75
Good job, you are on level 2!
|       |
----------
|    X  |
----------
|       |
----------

Please solve the following equation:
x * 4 = -88, what is x? -22
Good job, you are on level 3!
```

```
|    X  |
----------
|       |
----------
|       |
----------

You win the game!!!

Do you want to play again (0-no, 1-yes)? 0
```

Additional Requirements

- Your final submission must be implemented using at least 3 functions
  - You must write (and use) your own function to convert a C++ string into a signed integer (you may not use any of the built-in C++ conversions for strings)
    - Clarification (10/17): Your function needs to handle the conversion using only C++ math operations (e.g. no usage of the atoi function)
    - this function will accept a string
    - this function will return an int
    - if the string did not contain only numeric data, the function will return a value of INT_MAX (from the <climits> library)
  - You must have a function that displays an image of the elevator. It should accept two parameters: 1) the number of floors in the building & 2) the current elevator floor.
  - You must write a function that displays a math problem on the screen. This function must:
    - be named: display_equation
    - accept three input parameters: 1) the type of math problem (1 for addition, 2 for subtraction, 3 for multiplication, 4 for division) 2) the lowest acceptable random number (e.g. 0 for easy mode) & 3) the highest acceptable random number (e.g. 100)
    - print a randomly generated math problem on the screen
    - return the answer to the displayed math problem as a signed int
    - use the declaration: **int display_equation(int eq_type, int range_low, int range_high);**
- Catch all bad input (bad cases, non-matching types, etc)
- The answer to your math equations should always be an integer. So **x * 12 = 100** would not be a valid question for this game since the answer (8.33) is not an integer.
- The elevator cannot go below the first floor.
- The user must be able to choose to repeat the game (after winning it).

Program Tips

- **Added 10/16/18**: Some students have asked for tips on techniques to generate the random problems. This might be more difficult that it seems at first glance. As stated above, the answer to each problem must always be an integer (i.e. the answer (X) can never be something like 2/3 or 0.72)

- One option is to write your code to keep generating random questions until it finds one that meets the criteria of X being an integer. This is allowed but it isn't very efficient.
- The alternative is to be creative in the way you generate the random questions
  - For addition and subtraction, this is easy, since you are to use the format X + A = B (where A and B are random numbers). Since A and B are integers, it is implied that X will also be an integer.
  - Division should also be a simple case. In the equation X/A = B, X will solve to be an integer (assuming A and B are integers).
  - Multiplication is a trickier case (e.g. X * A = B). As example, consider that A and B will fall into the range -100 to 100 (if the user is playing on hard difficulty).
    A question such as X * -77 = 3 will not work (since the answer would be the fraction -3/77). To help address this problem, you could take two steps. First, write your code so that the absolute value of B is always greater than the absolute value of A (effectively you just switch the two random variables if necessary). In this example, we would then have X * 3 = -77. Unfortunately, in this case X will still be a fraction (e.g. -77/3 = -25.66...). One option is to use these random numbers and then adjust the numbers slightly to generate a valid problem. So your program could perform the **integer** division B/A (in this case that is -77/3 which equals -25). Your code would then work "backwards" to generate the problem. Take the quotient of the integer division and multiply that by the value of A (in this example that would be -25 * 3 = -75). Now you can replace B with the newly generated number. So in this example, A is now 3 and B has been adjusted to -75. The student's problem that will be displayed is X * 3 = -75. Now the math will work correctly so that the answer X is an integer.
  - For the multiplication workaround discussed above, it's okay that you are adjusting the original random number (and possibly switching them) so that the answer will be an integer.
- During week 3 you do not need to know about functions to successfully design solutions to each problem
- Treat each function as a mini program
- Design the solution to each problem using loops, conditionals and what you know about strings
- It could be helpful to put each solution on a notecard to simulate a function

The rest of the implementation is up to you, but try to make your game as clean and attractive to play as possible.

Creativity Contest!

Who has the most entertaining program with ASCII art? All students will get the opportunity to enter a contest (you can signup after the submission deadline on Sunday, October 21st). The submissions will be collected and CS161 students will vote on the results. The author of the entry with the most votes will receive a $10 gift certificate to Interzone (a coffee shop located on Monroe St. near OSU campus). Please note that this is not an endorsement of the Interzone company, rather this gift card was mailed to me by a textbook publisher and I feel that it's not ethical for me to keep it.

(15 pts) Program Style/Comments

In your implementation, make sure that you include a program header in your program, in addition to proper indentation/spacing and other comments! Below is an example header to include. Make sure you review the **style guidelines for this class (https://web.engr.oregonstate.edu/~goinsj/resources/general/cpp_style_guideline.pdf)** , and try to follow them, i.e. don't align everything on the left or put everything on one line!

```
/*************************************************
** Program: elevatormath.cpp
** Author: Your Name
** Date: 10/09/2018
** Description:
** Input:
** Output:
*************************************************/
```

Assignment Submission

Note that there are two deadlines for this assignment.

Your design document must be scanned and electronically submitted to Canvas (in the assignment tab) by **Sunday, October 14th, 11:59pm**. You may use one of the document scanners (available in KEC1130 and some other College of Engineering labs) to scan your paper design into a PDF document that can be submitted to Canvas.

Electronically submit your C++ source code to **TEACH (https://engineering.oregonstate.edu/teach)** by the assignment due date on **Sunday, October 21st, 11:59pm**. Your TEACH submission should contain only the .cpp file with your source code.

Reminder

Every assignment in this course is graded by demoing your work for 10 minutes with a TA. You are required to **meet with a TA within two weeks of the due date** to demo. You can schedule a demo with a TA from the **TA Office Hours** tab in Canvas. The available times can be viewed in the far right column of the table (labeled "Grading Hours"). Click on one of the links to access the poll and insert your name in a specific time slot.

- **Demo Outside 2 Weeks**: Assignments that are not demo'd within the acceptable time period will be subject to a 50 point deduction.
- **Demo Late Assignments**: Late assignments must still be demoed within the two week demo period beginning from the assignment's original due date.
- **Missing a Demo**: If you miss your demo with a TA, you will receive a 10 point (one letter grade) deduction to that assignment for each demo missed.

Each program needs to be written according to the **style guidelines (https://web.engr.oregonstate.edu/~goinsj/resources/general/cpp_style_guideline.pdf)** for this class. Remember that an important part of computer programming is making sure that your work is easily understandable by other programmers.

Program #3 Rubric

Criteria	Ratings	Pts
Program Header At a minimum, header should contain author's name and a description of the program. (2 pts each) 1 point for submitting code that compiles on flip.		5.0 pts
Good indentation / Use of whitespace Is code easy for the TA to read? Conditional blocks of code should always be indented.		5.0 pts
Each function is documented Every function contains it's own initial block comment (or multiple lines of comments prior to the function definition) that provides the reader with an explanation of the function's purpose. -1 pt for each function that is missing a header (up to 5 pts)		5.0 pts
Implements & uses function to convert C++ string into a signed int (see details) Function accepts a C++ string (2 pts) Function returns an int (2 pts) Function returns INT_MAX if string contained any non-integer input (besides a negative sign) (4 pts) Function uses only C++ math operations to convert the string into a corresponding int (no atoi, no stoi, etc). Student losses all of these points if function uses non-math operations. (5 pts) User's "string to int" function is used at least once in the code (5 pts)		18.0 pts
Code implements (& uses) display_equation function Uses exact declaration/prototype: int display_equation(int eq_type, int range_low, int range_high); (4 pts) Function prints an equation to the screen of the requested type (addition, multiplication, etc) (2 pts) Function returns the correct answer to the generated problem (10 pts) Student's code calls the display_equation with appropriate parameters for easy (0 to 100) and difficult (-100 to 100) mode (2 pts)		18.0 pts
Code implements function to draw elevator on screen Function accepts two parameters: # of floors in the building, current elevator floor (2 pts) Function displays the elevator in ASCII art on the specified floor (8 pts)		10.0 pts
Input validation (see details) Edge cases might vary depending on implementation (e.g. "-1", "0", "4.3", "2a", ""). Code should reprompt if input was not valid. Code handles bad input when prompting for number of floors (2 pts) Code handles bad input when prompting for level of difficulty (2 pts) Code handles bad input when operating in easy mode and asking user for the type of math problem (2 pts) Code handles bad input when prompting the user for the correct answer to the displayed problem (2 pts) Code handles bad input when asking the user if they want to play again (2 pts)		10.0 pts
Random number usage / Equation generation Code uses rand() to generate 2 random numbers for each algebra question (2 pts) Code randomly picks a type of equation to display if user is in difficult mode (2 pts) All randomly generated questions can be solved using only integer responses. In particular, pay special attention to / and * questions. No floating point answers are required. (2 pts for each type, +, -, *, /)		12.0 pts

Criteria	Ratings	Pts
Proper game operation (see details) User gets to choose how many floors are in the building (1 pts) User gets to choose whether to play in easy or difficult mode (1 pts) If playing in easy mode, user gets to choose the type of problem displayed for each round (+, -, *, /) (2 pts) Program calls the elevator display function after each question and shows the user their current position (with a blank line added into the display) (3 pts) User goes up one floor if problem is answered correctly (2 pts) User goes down one floor if problem is answered correctly (2 pts) Elevator cannot go below the first floor (2 pts) User wins after reaching the top level (2 pts) Code allows user to choose to repeat the game after winning it (2 pts)		17.0 pts
		Total Points: 100.0