Assignment 2: Math Millionaire

Due: 23:59, Thu 6 Oct 2022 Full marks: 100

Introduction

The objective of this assignment is to let you practice the use of control flow constructs (conditional and looping statements). You will write a C++ program to simulate the international TV game show *"Who Wants to Be a Millionaire?"* in the math flavor and a simplified manner.

In case you are not familiar with this game show, you may look at this Wikipedia page. There was a Hong Kong variant of the game show franchise, which was once a popular TV program during 2001-2005. The main goal of the game is to win HK$1 million by correctly answering 15 multiple-choice questions with 4 answer choices and increasing difficulty. Answering each question correctly can win a certain amount of money. The player can choose to walk away with the money earned thus far or to continue tackling more difficult questions to increase their prize fund at the risk of losing what they earned so far. The player has a set of *lifelines* that they may use only once to help them with a question. There are also two "safety nets" at certain levels, passing which will guarantee a certain prize fund to take away if the player gets the current question wrong.

Table 1: The game's payout structure

Question number	Question value (HK$)
15	1000000
14	500000
13	250000
12	150000
11	80000
10	60000#
9	40000
8	30000
7	20000
6	10000
5	8000#
4	4000
3	3000
2	2000
1	1000

In this assignment, we will adopt the payout structure used in the Hong Kong version of the game (Table 1), in which level 5 and level 10 (highlighted in pink and marked with #) are taken as the safety nets. That means if the player answers Question 5 right, they will earn at least HK$8,000 for sure. Answering Question 10 correctly will guarantee a prize of HK$60,000.

To simplify the implementation, you are required to implement only one lifeline in this game, which is adopted from the US version – *Jump the Question*. This lifeline allows the player to skip the current

question and move on to the next one, but not to earn the money of the question they skipped. The lifeline can be used once only and cannot be used in the final question (Question 15). Note that when used in a "safety net" question, say skipping Question 5, the player moves on to Question 6 but this hasn't yet guaranteed the take-away of HK$8,000. If the player answers Question 6 wrong, they will leave the game empty-handed (HK$ 0). On the contrary, skipping Question 5 and getting Question 6 correct, the safety net of getting HK$8,000 will still be effective for subsequent levels.

Program Specification

In the program output, each question begins with a header line whose format is:

`Question n ($m):`

where `n` is the question number and `m` is the question value.

Refer to Table 1 for the corresponding prize value for each question. Examples:

- `Question 1 ($1000):`
- `Question 15 ($1000000):`

For safety net questions, there is an extra symbol # shown to remind the player.

- `Question 5 ($8000#):`
- `Question 10 ($60000#):`

All the questions in this game are simple arithmetic questions of the following structure:

`x op y,`

where op is one of the binary operators below:

- `+` : addition
- `-` : subtraction
- `*` : multiplication
- `/` : integer division

The two operands x and y in each question are random integers in the range from 1 to 100, inclusive. The choice of operator for a question is also picked randomly.

Each question will be displayed in the following format:

`x op y = ?`

For example,

`20 + 15 = ?`

Each question is followed by its answer list whose format is illustrated in the example below:

`A. 14  B. 27  C. 35  D. 27  E. Jump  F. Withdraw`

The line shows 4 numeric answer choices plus two more options for using the "Jump the Question" lifeline and withdrawing from the game. Note: one of the 4 numeric choices is the correct answer. The other three answers are again random integers in the range from 1 to 100, inclusive. They could be duplicated values (e.g., both choices B and D are 27), but they must be different from the correct

answer (35 in this case). The correct answer is randomly positioned in the 4-choice answer list. A single space follows each dot symbol; two spaces exist between choices. For clarity, we revise the example with the space characters shown as ␣ as follows:

```
A.␣14␣␣B.␣27␣␣C.␣35␣␣D.␣27␣␣E.␣Jump␣␣F.␣Withdraw
```

You do *not* have to validate the user input. We assume that *all inputs are always valid options*. We will not test your program against invalid inputs, and the behavior of your program in such cases could deviate from that of our sample program.

The following suggests a rough rundown of your program. Your actual program design and variable naming may differ from what we provide here as long as they are correct.

- Prompt the user to enter an integer to seed the random number generator.
- Initialize the prize value for the 1st question, amount of money earned, and any other variables.
- Write the main loop for asking the user 15 questions, which carries out the following steps:

1. For each question, generate two random integers in [1, 100] for the operands `x` and `y`. Generate another random integer `z` in [0, 3]~~, or [1, 4],~~ to pick one of the operators (+, -, *, /) to form the question `x op y`.

2. Compute the solution `soln` (int) to the formed question `x op y`.

3. Print the question header. Remember to print # for safety net questions 5 and 10.

4. Print a line showing the question, e.g.,
   ```
   20 + 15 = ?
   ```

5. Print the answer choice line, e.g.,
   ```
   A. 14   B. 27   C. 35   D. 27   E. Jump   F. Withdraw
   ```

   In this step, generate 3 random integers in [1, 100] to be the wrong answers that mix with the correct answer. Use extra looping to ensure their values are different from the correct value `soln`. To randomly place the correct answer in the output, you may generate another random integer `c` in [0, 3]~~, or [1, 4],~~ to decide which of A, B, C and D should be associated with `soln`. Save the character of the correct choice into a character variable, namely `correct`. Refer to the pseudocode in Figure 1 for more details of how to implement this part. The option "E. Jump" should be shown only if the lifeline was not yet used.

6. Prompt the user to enter an option (character 'A', 'B', 'C', 'D', 'E' or 'F') for the question. Save it into a character variable. Note that your program should accept both uppercase and lowercase, so 'a' or 'A' means the same choice. Entering 'E' while the option "E. Jump" has disappeared (i.e., the lifeline has been used) is taken as a wrong answer leading to the end of the game.

7. Set the prize value for the next question according to Table 1.

8. If the input equals 'F' (i.e., withdraw), set money earned to be the withdrawal amount saved in step 12; break the main loop to end the game.

9. If (1) the lifeline was not used, and (2) the question level is not 15 (final question), and (3) the input equals 'E', then mark the lifeline as used and continue the next iteration, thus bypassing the current question and its reward.

10. If user input does not equal `correct`, print "Wrong!" and break the main loop.

11. *(Reaching here means "correctly answered".)* Print "Right!" and save the current question value into a variable denoting the withdrawal amount.

12. Handle safety net cases (Question 5 and Question 10). Set money earned to the guaranteed value accordingly.

When the game ends, print two lines to indicate game-over and the amount of money earned, e.g.,

```
Game over!
You got $10000!
```

If the user wins $1000000, then print an extra line of congratulation:

```
Congrats! You are a millionaire!
```

Some Remarks on Implementation

This assignment aims mainly at practicing control flow (if, else, switch, ?:, while, do-while, for, break, continue, etc.). You can use any combination of these constructs. The use of arrays is basically not necessary but not forbidden in this assignment. You may use any function in the C++ standard library if you see fit. But non-standard C++ or third-party library functions are not allowed.

To ease program testing and our grading, your program is required to prompt the user to enter the seed (e.g., 1) for the random number generator before the game begins. Setting the seed value makes your program output deterministic because the same random number sequence is generated for the same seed. You are also required to follow our stated method (using the mt19937 random number generator) to generate the random integers. We have provided an example C++ source file (example_rng.cpp) for how to generate a random integer within a specified range.

As you can see above, there are several random integers to be generated in this program. In order to produce exactly the same output (same questions and answer lists) as our sample program does, the random integers must be generated in the same order used in our sample program. Please follow the pseudocode in Figure 1 for how to generate the questions and answer lists. It is important to follow our order of generating x, y, z, c, r's and the order of listing +, -, *, / and A, B, C, D in your program. If you reorder some of them, say generating z before y, then your program output will differ from our sample, and your program will fail to pass most test cases in our grading.

Also note the difference in random number sequences generated on Windows and on macOS due to different library implementations. Even for the same seed, the two platforms may generate different sets of random integers for the questions. The sample runs given below were conducted on macOS. For Windows users, please use the sample program for Windows to perform your correctness checks.

```
uniform_int_distribution<> distr1(1, 100);  // random int in [1, 100]
uniform_int_distribution<> distr2(0, 3);    // random int in [0, 3]

for / while { // loop through 15 questions

    int x = distr1(gen);  // 1st operand [1, 100]
    int y = distr1(gen);  // 2nd operand [1, 100]
    int z = distr2(gen);  // choice of operator (+, -, *, /)
    int c = distr2(gen);  // position of correct answer (A, B, C, D)

    op = operator at z
    soln = x op y         // e.g. realized by a switch-case statement
                                z = 0 -> soln = x + y
                                z = 1 -> soln = x - y
                                z = 2 -> soln = x * y
                                z = 3 -> soln = x / y

    for ( k in [0, 3] ) { // choice A. B. C. D.
                          // k = 0 -> A, k = 1 -> B, k = 2 -> C, k = 3 -> D
        char choice = ASCII of 'A' + k
        // print choice.
        if (k == c) {
            // print soln, save choice char as the correct choice
        }
        else {
            int r;  // a wrong answer
            do {
                  r = distr1(gen);  // [1, 100]
            }
            while (r == soln);  // avoid duplicate with correct answer
            // print r
        }
        ...
    }
}
```

Figure 1. Pseudocode for generating the questions and answer lists

A hint on testing: Since we restrict all the wrong answers to be within [1, 100], whenever you see an answer choice of negative sign or more than 2 digits (except 100), that must be the correct answer. This can save some calculation effort and speed up your testing.

Sample Runs

In the following sample runs, the blue text is user input and the other text is the program printout. You can try the provided sample program for other input. *Your program output should be exactly the same as what the sample program produces* (same text, symbols, letter case, spacings, etc.). Note that *there is a trailing space after the ':' symbol in the user prompt text "Final answer: "*.

Sample Run 1

The player gets first 2 questions correct, skips Question 3, and withdraws at level 4, earning $2000.

```
Enter seed: 1↵
Question 1 ($1000):
38 + 13 = ?
A. 10  B. 76  C. 6  D. 51  E. Jump  F. Withdraw
Final answer: D↵
Right!
Question 2 ($2000):
80 + 65 = ?
A. 77  B. 145  C. 72  D. 7  E. Jump  F. Withdraw
Final answer: B↵
Right!
Question 3 ($3000):
26 + 51 = ?
A. 19  B. 85  C. 77  D. 12  E. Jump  F. Withdraw
Final answer: E↵
Question 4 ($4000):
29 * 30 = ?
A. 69  B. 88  C. 870  D. 88  F. Withdraw
Final answer: F↵
Game over!
You got $2000!
```

Sample Run 2

The player gets first 4 questions correct, skips the safety-net Question 5, but answers the next question wrong. In this case, safety net does not apply, and the player earns $0.

```
Enter seed: 1↵
Question 1 ($1000):
38 + 13 = ?
A. 10  B. 76  C. 6  D. 51  E. Jump  F. Withdraw
Final answer: d↵
Right!
Question 2 ($2000):
80 + 65 = ?
A. 77  B. 145  C. 72  D. 7  E. Jump  F. Withdraw
Final answer: b↵
Right!
Question 3 ($3000):
26 + 51 = ?
A. 19  B. 85  C. 77  D. 12  E. Jump  F. Withdraw
Final answer: c↵
Right!
Question 4 ($4000):
29 * 30 = ?
A. 69  B. 88  C. 870  D. 88  E. Jump  F. Withdraw
Final answer: c↵
Right!
Question 5 ($8000#):
95 * 97 = ?
```

```
A. 10  B. 9215  C. 8  D. 64  E. Jump  F. Withdraw
Final answer: e↵
Question 6 ($10000):
62 - 23 = ?
A. 1  B. 39  C. 61  D. 82  F. Withdraw
Final answer: a↵
Wrong!
Game over!
You got $0!
```

Sample Run 3

The player gets first 4 questions correct, skips the safety-net Question 5, then answers Question 6 correct but Question 7 wrong. In this case, the safety net applies, and the player earns $8000.

```
Enter seed: 1↵
Question 1 ($1000):
38 + 13 = ?
A. 10  B. 76  C. 6  D. 51  E. Jump  F. Withdraw
Final answer: d↵
Right!
Question 2 ($2000):
80 + 65 = ?
A. 77  B. 145  C. 72  D. 7  E. Jump  F. Withdraw
Final answer: b↵
Right!
Question 3 ($3000):
26 + 51 = ?
A. 19  B. 85  C. 77  D. 12  E. Jump  F. Withdraw
Final answer: c↵
Right!
Question 4 ($4000):
29 * 30 = ?
A. 69  B. 88  C. 870  D. 88  E. Jump  F. Withdraw
Final answer: c↵
Right!
Question 5 ($8000#):
95 * 97 = ?
A. 10  B. 9215  C. 8  D. 64  E. Jump  F. Withdraw
Final answer: e↵
Question 6 ($10000):
62 - 23 = ?
A. 1  B. 39  C. 61  D. 82  F. Withdraw
Final answer: b↵
Right!
Question 7 ($20000):
9 - 89 = ?
A. 48  B. 73  C. 31  D. -80  F. Withdraw
Final answer: c↵
Wrong!
Game over!
You got $8000!
```

Sample Run 4

The player gets first 6 questions correct, skips Question 7 ($20000), and then withdraws from the game at Question 8 ($30000). In this case, the earning is $10000 only.

```
Enter seed: 2↵
Question 1 ($1000):
41 - 16 = ?
A. 25  B. 23  C. 44  D. 83  E. Jump  F. Withdraw
Final answer: a↵
Right!
Question 2 ($2000):
76 * 8 = ?
A. 96  B. 608  C. 76  D. 86  E. Jump  F. Withdraw
Final answer: b↵
Right!
Question 3 ($3000):
48 / 64 = ?
A. 21  B. 38  C. 0  D. 40  E. Jump  F. Withdraw
Final answer: c↵
Right!
Question 4 ($4000):
68 * 5 = ?
A. 52  B. 39  C. 34  D. 340  E. Jump  F. Withdraw
Final answer: d↵
Right!
Question 5 ($8000#):
59 - 68 = ?
A. -9  B. 69  C. 47  D. 71  E. Jump  F. Withdraw
Final answer: a↵
Right!
Question 6 ($10000):
96 - 84 = ?
A. 67  B. 81  C. 53  D. 12  E. Jump  F. Withdraw
Final answer: d↵
Right!
Question 7 ($20000):
77 + 51 = ?
A. 64  B. 80  C. 128  D. 50  E. Jump  F. Withdraw
Final answer: e↵
Question 8 ($30000):
40 + 47 = ?
A. 16  B. 9  C. 87  D. 18  F. Withdraw
Final answer: f↵
Game over!
You got $10000!
```

Sample Run 5

The player gets all 15 questions correct, earning $1M. Note the extra congratulation message at the end, and that the “E. Jump” option disappears in Question 15 (because the final question disallows it) even if it was not used in all previous questions.

```
Enter seed: 2↵
Question 1 ($1000):
41 - 16 = ?
A. 25  B. 23  C. 44  D. 83  E. Jump  F. Withdraw
Final answer: a↵
Right!
Question 2 ($2000):
76 * 8 = ?
A. 96  B. 608  C. 76  D. 86  E. Jump  F. Withdraw
Final answer: b↵
Right!
Question 3 ($3000):
48 / 64 = ?
A. 21  B. 38  C. 0  D. 40  E. Jump  F. Withdraw
Final answer: c↵
Right!
Question 4 ($4000):
68 * 5 = ?
A. 52  B. 39  C. 34  D. 340  E. Jump  F. Withdraw
Final answer: d↵
Right!
Question 5 ($8000#):
59 - 68 = ?
A. -9  B. 69  C. 47  D. 71  E. Jump  F. Withdraw
Final answer: a↵
Right!
Question 6 ($10000):
96 - 84 = ?
A. 67  B. 81  C. 53  D. 12  E. Jump  F. Withdraw
Final answer: d↵
Right!
Question 7 ($20000):
77 + 51 = ?
A. 64  B. 80  C. 128  D. 50  E. Jump  F. Withdraw
Final answer: c↵
Right!
Question 8 ($30000):
40 + 47 = ?
A. 16  B. 9  C. 87  D. 18  E. Jump  F. Withdraw
Final answer: c↵
Right!
Question 9 ($40000):
23 - 74 = ?
A. 91  B. -51  C. 63  D. 84  E. Jump  F. Withdraw
Final answer: b↵
Right!
Question 10 ($60000#):
97 + 44 = ?
A. 9  B. 77  C. 141  D. 11  E. Jump  F. Withdraw
Final answer: c↵
Right!
```

```
Question 11 ($80000):
41 + 35 = ?
A. 71  B. 76  C. 87  D. 71  E. Jump  F. Withdraw
Final answer: b↵
Right!
Question 12 ($150000):
20 * 57 = ?
A. 69  B. 1140  C. 41  D. 82  E. Jump  F. Withdraw
Final answer: b↵
Right!
Question 13 ($250000):
62 - 71 = ?
A. 98  B. -9  C. 19  D. 85  E. Jump  F. Withdraw
Final answer: b↵
Right!
Question 14 ($500000):
91 * 88 = ?
A. 53  B. 75  C. 73  D. 8008  E. Jump  F. Withdraw
Final answer: d↵
Right!
Question 15 ($1000000):
91 / 100 = ?
A. 0  B. 17  C. 56  D. 22  F. Withdraw
Final answer: a↵
Right!
Game over!
You got $1000000!
Congrats! You are a millionaire!
```

Submission and Marking

- Your program file name should be `millionaire.cpp`. Submit the file in Blackboard (https://blackboard.cuhk.edu.hk/).
- Insert *your name*, *student ID*, and *e-mail* as comments at the beginning of your source file.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- Your program should *include suitable comments as documentation*.
- ***Do NOT plagiarize.*** Sending your work to others is subject to the same penalty for copying work.