# COMP 2710 - Project 2: TRUEL OF THE FATES

Points Possible: 100
Deadline: **11:59pm, Friday, Sep 27th, 2024 (Central Time)**

***Goals:***

- Able to learn the semantics of "while" and "do-while" statements.
- Able to learn how to define functions, create functional logics, and return values.
- Able to write and use test drivers.
- Able to learn how to use assert() function.
- Able to use randomization to get random numbers.
- Able to apply git to control a big project with multiple versions.

***Description:***

In the late 1800s, somewhere in rural town Alabama, Aaron, Bob, and Charlie had an intensive armed battle over who would become the leader of Alabamian Marksmanship Council. They were leaders of opposing marksman groups in the South, and the council overlooking them recently lost its leadership. So, the position is up for grab by only one group leader to rule 'em all. That means, one must kill the other two to emerge victorious, so a truel to the death was agreed upon by all three.

Aaron was a poor shooter and only hit his target with a probability of $1/3$. Bob was a bit better and hit his target with a probability of $1/2$. In contrast, Charlie was an expert and never missed. A hit means a kill, and that person immediately drops out of the truel.

To compensate for the imbalances in their marksmanship skills, the three decided that they would fire in turns, starting with Aaron, followed by Bob, and then by Charlie. The cycle would then repeat until there was one last man standing. He would obtain the title of the President of the Alabamian Marksmanship Council. Now, all gunmen would follow common strategies that could give them fair chance. They include:

**Strategy 1: An obvious and reasonable strategy is for each man to shoot at the most accurate shooter still alive, on the basis that this shooter is the deadliest and has the best chance of hitting back. Write a program to simulate the truel using this strategy.**

Your program should use random numbers and the probabilities given in the problem to determine whether a shooter hits his target. You will likely want to create multiple functions to simulate the problem. Once you can simulate a single truel, add a loop to your program that simulates 10,000 truels. Count the number of times each marksman wins and print their probability of winning. For example, if we are talking about Aaron's chance of winning, your output might look something like this:

```
Aaron won 3612/10000 truels or 36.12%.
```

Let us look at this strategy: each gunman $i$ has a probability of $p_i$ to hit the strongest opponent and $1 - p_i$ to miss his shot. Now, you should note that, in a single truel, if Charlie remains alive for 2 rounds, he will kill the last opponent if it is his turn on the 2nd round. If Charlie is dead before that, the truel can possibly go endlessly if both Bob and Aaron did a miss consecutively. Does this information give you an idea of how your simulation results should be for a single truel? After that, consider each truel as an independent event (do not mistake this as a round of truel), how will the results be if you repeat it 10,000 times?

If you are comfortable with probabilities, you can try framing this problem based on the intuition of conditional probability, and independent of events, whatever suits you.

**Strategy 2: An alternative strategy for Aaron is to intentionally miss on his first shot, while the rest of a truel continues in the same matter as in Strategy 1. Write a function, or multiple functions of your choice, to simulate Strategy 2.**

Think about why this strategy is always advantageous for low-skilled gunman like Aaron to defeat his opponents compared to the 1st strategy, using the same analysis above, but this time, Aaron's probability for his first round of every truel is $p = 0$. Write a simple analysis on Strategy 2 based on your understanding as a comment block at the head your code file.

**One more objective: Because Aaron apparently is the weakest link in the truel, your program must determine which strategy, either 1 or 2, is better and fairer for Aaron, based on the simulation results.**

**Note 1:** You must provide the following user interface with relatively similar output structure. Probabilities might be slightly different for different runs due to randomness, but the one clear thing is strategy 2 should be always better than strategy 1. You do not need to color the texts that are highlighted in red in the example output. Also, replace the word "Li" with your name.

```
*** Welcome to Li's Truel of the Fates Simulator ***
Unit Testing 1: Function - at_least_two_alive()
      Case 1: Aaron alive, Bob alive, Charlie alive
             Case passed ...
      Case 2: Aaron dead, Bob alive, Charlie alive
             Case passed ...
      Case 3: Aaron alive, Bob dead, Charlie alive
             Case passed ...
      Case 4: Aaron alive, Bob alive, Charlie dead
             Case passed ...
      Case 5: Aaron dead, Bob dead, Charlie alive
             Case passed ...
      Case 6: Aaron dead, Bob alive, Charlie dead
             Case passed ...
```

```
      Case 7: Aaron alive, Bob dead, Charlie dead
             Case passed ...
      Case 8: Aaron dead, Bob dead, Charlie dead
             Case passed ...
Press any key to continue...

Unit Testing 2: Function Aaron_shoots1(Bob_alive, Charlie_alive)
      Case 1: Bob alive, Charlie alive
              Aaron is shooting at Charlie
      Case 2: Bob dead, Charlie alive
              Aaron is shooting at Charlie
      Case 3: Bob alive, Charlie dead
              Aaron is shooting at Bob
Press any key to continue...

Unit Testing 3: Function Bob_shoots(Aaron_alive, Charlie_alive)
      Case 1: Aaron alive, Charlie alive
              Bob is shooting at Charlie
      Case 2: Aaron dead, Charlie alive
              Bob is shooting at Charlie
      Case 3: Aaron alive, Charlie dead
              Bob is shooting at Aaron
Press any key to continue...

Unit Testing 4: Function Charlie_shoots(Aaron_alive, Bob_alive)
       Case 1: Aaron alive, Bob alive
               Charlie is shooting at Bob
       Case 2: Aaron dead, Bob alive
               Charlie is shooting at Bob
       Case 3: Aaron alive, Bob dead
               Charlie is shooting at Aaron
Press any key to continue...

Unit Testing 5: Function Aaron_shoots2(Bob_alive, Charlie_alive)
       Case 1: Bob alive, Charlie alive
               Aaron intentionally misses his first shot
               Both Bob and Charlie are alive.
       Case 2: Bob dead, Charlie alive
               Aaron is shooting at Charlie
       Case 3: Bob alive, Charlie dead
               Aaron is shooting at Bob
Press any key to continue...

Ready to test strategy 1 (run 10000 times):
Press any key to continue...

Aaron won 3593/10000 truels or 35.9%
Bob won 4169/10000 truels or 41.69%
Charlie won 2238/10000 truels or 22.38%

Ready to test strategy 2 (run 10000 times):
Press any key to continue...

Aaron won 4131/10000 truels or 41.31%
Bob won 2594/10000 truels or 25.94%
Charlie won 3275/10000 truels or 32.75%

Strategy 2 is better than strategy 1.
```

***Getting into the gitty bits:***

You must practice controlling versions of your code files using the `git` command and separate your project into two phases. The first phase will be your program without the testdriver codes, and the second phase will include these codes along with your first phase codes in order to test your program. You will need to perform a `git` update to your project after finishing your second phase.

You are asked to submit both your phase 1 and phase 2 files and name them as version 1 and version 2 accordingly. Both files must have the type of content we asked and we will check if you actually do the `git` command, so please don't be dishonest.

If you are unsure where to begin or practice, refers to Dr. Li's document in Canvas > Files, named "Tutorial_Git_2710.pdf", or simply find any good tutorial online.

***Requirements:***

1. You must follow the above user interface to implement your program.
2. You must implement 5 following functions:

```
1) bool at_least_two_alive(bool A_alive, bool B_alive, C_alive)
   /* Input:   A_alive indicates whether Aaron is alive
    *          B_alive indicates whether Bob is alive
    *          C_alive indicates whether Charlie is alive
    * Return:   true if at least two are alive
    *           otherwise return false
   */


2) void Aaron_shoots1(bool& B_alive, bool& C_alive)
   /* Strategy 1: Use call by reference
    * Input:  B_alive indicates whether Bob alive or dead
    *         C_alive indicates whether Charlie is alive or dead
    * Return:  Change B_alive into false if Bob is killed.
    *          Change C_alive into false if Charlie is killed.
    */


3) void Bob_shoots(bool& A_alive, bool& C_alive)
   /* Call by reference
    * Input:  A_alive indicates if Aaron is alive or dead
    *         C_alive indicates whether Charlie is alive or dead
    * Return:  Change A_alive into false if Aaron is killed.
    *          Change C_alive into false if Charlie is killed.
   */


4) void Charlie_shoots(bool& A_alive, bool& B_alive)
   /* Call by reference
    * Input:  A_alive indicates if Aaron is alive or dead
```

```
     *          B_alive indicates whether Bob is alive or dead
     * Return:  Change A_alive into false if Aaron is killed.
     *           Change B_alive into false if Bob is killed.
    */
```

```
5)  void Aaron_shoots2(bool& B_alive, bool& C_alive)
    /* Strategy 2: Use call by reference
     * Input:  B_alive indicates whether Bob alive or dead
     *         C_alive indicates whether Charlie is alive or dead
     * Return:  Change B_alive into false if Bob is killed.
     *           Change C_alive into false if Charlie is killed.
    */
```

3. You must implement five unit-test drivers (five functions) to test the above five functions (see the example output on pages 2 and 3).

4. You must use the `assert()` function your test driver.

5. You must define at least three constants in your implementation. For example, the total number of runs (i.e., 10,000) can be defined as a constant. Constant definition in C++ is keyworded as `const`.

6. You must create at least two versions for solutions:
   - Version 1: Code without the test drivers
   - Version 2: "Test drivers" code.

7. The "test drivers" code must be successfully merged into the version 1.

***Hints:***

1. How to implement the line which asks you to press any key to continue in the output:

   ```
   Press any key to continue...
   ```

   ```
   // Pause Command for Linux Terminal
   cout << "Press Enter to continue...";
   cin.ignore().get();
   ```

   **Note:** You can implement the above two lines as a function to be repeatedly called by other functions in your program. If the code does not work for you, you are suggested to research and create your own way of doing it.

2. You may need to use the following libraries:

   ```
   # include <iostream>
   # include <stdlib.h>
   # include <assert.h>
   # include <ctime>
   ```

3. Initialize your random number generator like this:
   ```
   srand(time(0));
   ```

4. A sample code of generating a random number is given below:

   ```
   /* Assume that the probabilty of hit a target is 25 percent */
   int shoot_target_result;
   shoot_target_result = rand() % 100;
   if (shoot_target_result < 25) {
       cout << "Hit the target\n";

       /* add more code here */
   }
   ```

5. Please follow the following sample test driver to implement the test driver:

   ```
   void test_at_least_two_alive(void) {
      cout << "Unit Testing 1: Function – at_least_two_alive()\n";

      cout << "Case 1: Aaron alive, Bob alive, Charlie alive\n";
      assert(true == at_least_two_alive(true, true, true));
      cout << "Case passed ...\n";

      cout << "Case 2: Aaron dead, Bob alive, Charlie alive\n";
      assert(true == at_least_two_alive(false, true, true));
      cout << "Case passed ...\n";

      cout << "Case 3: Aaron alive, Bob dead, Charlie alive\n";
      assert(true == at_least_two_alive(true, false, true));
      cout << "Case passed ...\n";

      /* add test cases 4-6 below */
      ...
   }
   ```

6. If your strategy 1 is better than your strategy 2, then please recheck your source code. It is likely that you have incorrectly implemented either strategy. Have you tried to do an analysis to see why 2 must be better than 1?

7. The sample framework of the source code is given below:

```
# include <iostream>
# include <stdlib.h>
# include <assert.h>
# include <ctime>
using namespace std;

//prototypes
bool at_least_two_alive(bool A_alive, bool B_alive, bool C_alive);
/* Input: A_alive indicates whether Aaron is alive */
/*          B_alive indicates whether Bob is alive */
/*          C_alive indicates whether Charlie is alive */
```

```
/* Return: true if at least two are alive */
/*           otherwise return false */

/*
 * Add other function prototypes below
 */

void test_at_least_two_alive(void);
/* This is a test driver for at_least_two_alive() */

/*
 * Add more prototypes for other test drivers below
 */

int main()
{
     /*
      * This is the main function
      * …
      */
}

/* Implementation of at_least_two_alive() */
bool at_least_two_alive(bool A_alive, bool B_alive, bool C_alive)
{
     /* add the implementation below */
}


/* Implementation of the test driver for at_least_two_alive() */
void test_at_least_two_alive(void)
{
     cout << "Unit Testing 1: Function – at_least_two_alive()\n";

     cout << "Case 1: Aaron alive, Bob alive, Charlie alive\n";
     assert(true == at_least_two_alive(true, true, true));
     cout << "Case passed …\n";

     cout << "Case 2: Aaron dead, Bob alive, Charlie alive\n";
     assert(true == at_least_two_alive(false, true, true));
     cout << "Case passed …\n";

     /* add test cases 4-6 below */
}
```

***Programming Environment:***

Write a program in C++. Compile and run it using AU server (no matter what kind of text editor, IDE or coding environment you use, please make sure your code could run on AU server, the only test bed we accept is the AU server).

***Grading:***

***100 points maximum, criteria ordered from lowest to highest in point value.***

1. (3 points) Define at least three constants.
2. (5 points) You do your simple analysis on Strategy 2.
3. (5 points) You use `assert()` in your test drivers.
4. (5 points) Your program must compare strategy 1 with strategy 2.
5. (5 points) Your source code is of good quality, easy to read and well-organized.
6. (5 points) Use comments to provide a heading at the top of your code containing your name, Auburn Banner ID, filename, and how to compile your code. Also describe any help or sources that you used.

7. (5 points) Your source code files should be named like this:

   `project2_LastName_UserID_v1.cpp`.

   `project2_LastName_UserID_v2.cpp`.

   For example, `project2_King_pzh0039_v1.cpp` and `project2_King_pzh0039_v2.cpp`.

   **Note 3:** You will not lose any point if Canvas automatically changes your file name (e.g., `project2_LastName_UserID_v1-2.cpp`) due to your resubmissions.

8. (7 points) You must follow the specified user interface/example output.
9. (10 points) You do `git` updates and have two versions of your solution; each version has the type of content we asked.
10. (10 points) The 2nd version of your code (test drivers) can be successfully merged into the 1st, using the `git` command.
11. (20 points) Implement five required functions correctly.
12. (20 points) Implement five test drivers for the functions.

It is a lot to keep track, but it is also an opportunity for you to earn more points by just doing it and not to lose a ton on one or more constrained criteria.

**Note 4**: For this project, we will be looking at your code most of the time and mark points based on how good your implementation is, so be sure to pay attention to logic and presentation. Also, do not plagiarize. There won't be testcases to verify your program execution, because this is a simulation program requiring no external inputs from user.

**Note 5:** You will automatically lose **at least 40 points** if there are compilation errors or warning messages when we compile your source code. You will **lose points** if you don't

use the specific program file name, or don't have comments, or don't have a comment block on **EVERY** program you hand in. These will be deducted from your final score, after accounting all other requirements listed above.

***Deliverables:***

- Submit your source code files named as

  `project2_LastName_UserID_v1.cpp`

  `project2_LastName_UserID_v2.cpp`

  through the Canvas system.

  This project requires you to submit files ending with .cpp extension.

  No other file types are accepted.

***Late Submission Penalty:***

- Late submissions will not be accepted and will result in a **ZERO grade** without valid excuses, in which case you should talk to Dr. Li to explain your situation.
- GTA/Instructor will not accept any late submission caused by internet latency.

***Rebuttal period:***

- You will be given a period of **2 business days** to read and respond to the comments and grades of your homework or project assignment. The TA may use this opportunity to address any concern and question you have. The TA also may ask for additional information from you regarding your homework or project.

Good luck y'all! WAR EAGLE!