Description:

In the land of Puzzlevania, Aaron, Bob, and Charlie had an argument over which one of them was the greatest puzzle-solver of all time. To end the argument once and for all, they agreed on a duel to the death (this makes sense?). Aaron was a poor shot and only hit this target with a probability of 1/3. Bob was a bit better and hit his target with a probability of 1/2. Charlie was an expert marksman and never missed. A hit means a kill and the person hit drops out of the duel.

To compensate for the inequities 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 repeat until there was one man standing. That man would be remembered for all time as the Greatest Puzzle-Solver of All Time.

An obvious and reasonable strategy is for each man to shoot at the most accurate shooter still alive, on the grounds that this shooter is the deadliest and has the best chance of hitting back.

Write a program to simulate the duel 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 complete the problem. My solution had only one function to simulate the duels and it passed in the odds and the three guys as pass-by-reference parameters. Once you can simulate a duel, add a loop to your program that simulates 10,000 duels. Count the number of times that each contestant wins and print the probability of winning for each contestant (e.g., for Aaron your might output "Aaron won 3612/10000 duels or 36.12%).

Strategy 2: An alternative strategy for Aaron is to intentionally miss on his first shot. Write a function to simulate Strategy 2. Your program will determine which strategy is better for Aaron.

```
*** Welcome to Xiao Qin’s Duel 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
                Aaron misses.
        Case 2: Bob dead, Charlie alive
                Aaron is shooting at Charlie
                Charlie is dead.
        Case 3: Bob alive, Charlie dead
                Aaron is shooting at Bob
                Aaron misses.
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
                Charlie is dead.
        Case 2: Aaron dead, Charlie alive
                Aaron is shooting at Charlie
                Bob missed
        Case 3: Aaron alive, Charlie dead
                Bob is shooting at Aaron
                Aaron is dead.
Press any key to continue...
```

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

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

...
...
...

Aaron won 3612/10000 duels or 36.12%
Bob won 4137/10000 duels or 41.37%
Charlie won 2251/10000 duels or 22.51%

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

...
...
...

Aaron won 4099/10000 duels or 40.99%
Bob won 2552/10000 duels or 25.52%
Charlie won 3349/10000 duels or 33.49%

Strategy 2 is better than strategy 1.
```

1. You must implement the 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.
      */
   ```

2. You must implement five unit test drivers (five functions) to test the above five functions. (see also the user interface on pages 2 and 3). Instead of starting each section on key press, just notify the user that you are starting that specific section and run the tests.

3. You must use assert in your test driver.

4. You must define four constants in your implementation. For example, the total number (i.e., 10,000) of runs can be defined as a constant.

Hints:

1. How to implement `“Press any key to continue...”`

   ```
   DON’T DO THIS! Just notify the user that you are running
   some test/section and do that part of the program.
   ```

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 as below:

   ```
         srand(time(0));

   I’ve also included examples of another way to produce
   random numbers in prior assignments.
   ```

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 all the test drivers.

```
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 check your source code. In this case, it is likely that you have incorrectly implemented strategy 2. Please ensure that strategy 2 is better than strategy 1.

7. The 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, 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 */
}
```

Rubric:

1) 5 unit test driver functions exist. One for each function. Each function is worth **10 pts**.
2) 5 functions described above exist and are functionally correct. Each function is worth **5 pts**.
3) (**5 pts**) Your code runs all unit tests (all 5 functions) and displays result.
4) (**5 pts**) Your code runs a simulation for strategy 1 10 times and displays the results.
5) (**5 pts**) Your code runs a simulation for strategy 2 10 times and displays the results.
6) (**5 pts**) Strategy 2 ends up being better than strategy 1.
7) (**5 pts**) All output is run and displayed on Jenkins server. This last item is a requisite for earning any other points. Failing to produce any output at all on the Jenkins server will result in a 0 for all items.