# **Lab 5:** Simulations

## Overview

In a software development company, programmers are often asked to write the implementation of a function, given its prototype and a precise description of what it is supposed to do. To make it easier for them, they are also given a ***driver program***. A driver program is one that calls their function, so that they can test it and make sure it works.

In this lab assignment you are given a few function prototypes and precise descriptions of what these functions must do. These prototypes and descriptions are in a ***header file***. A header file is simply a file containing function prototypes, possibly constant and type definitions, and sufficiently detailed comments so that a programmer can make calls to those functions in his or her own program. Your job will be to create a file called an ***implementation file***, which contains the actual definitions of the functions in the header file. The names of the header file and the implementation file are usually coordinated so that they differ only in the file extension. If the header file is named `duel.h`, then its implementation file is named `duel.cpp` in C++ (or `duel.c` in C). Header files are always given the extension "`.h`", whether in C, C++, or another programming language.

Once you have written the function definitions in the file `duel.cpp`, you will be able to compile the file into an object file using the command

```
g++ -c duel.cpp
```

This will create the file `duel.o`.

You will not be given the source code to the driver program. You don't need it. But you will have its object file, which will be named `lab05_main.o`. With your `duel.o` file and `lab05_main.o`, can create the executable program with the command

```
g++ -o lab05 lab05_main.o duel.o
```

If your code has no syntax errors, the file `lab05` will be created in your working directory. If it has errors you will have to look at the messages and correct them. It is better to use the command

```
g++ -Wall -g -o lab05 lab05_main.o duel.o
```

turning on all warning messages and compiling code that can be run under the GDB debugger. For a more detailed explanation, see my tutorial, http:///~stewart/resources/separateCompilation.pdf.

## Exercises

You will be writing functions that are used in a game. This exercise will also give you practice with random number generation. The game simulates three people with different shooting skill levels trying to duel each other. In the imaginary land of Moronia, Aaron, Bob, and Charlie had an argument over which one of them was the greatest logician of all time. To end the argument once and for all, they agreed on a duel to the death, which seemed quite logical to all of them.

The facts surrounding the duel and their skill levels are that

- Aaron is a poor shooter and only hits his target with a probability of 1/3.

- Bob is a bit better and hits his target with a probability of 1/2.
- Charlie is an expert marksman and never misses.

A ***hit*** means a kill and the person hit drops out of the duel. A ***duel*** is carried out in a specific order. To compensate for the inequities in their marksmanship skills, they decided that they would take turns firing, starting with Aaron, then Bob, and then Charlie. Since each is a smart person, they each adopt the strategy 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. The duel is carried out until only one of them is left alive. It is not possible for all to die because only one person shoots at a time and either kills or does not kill the intended target.

There is a header file named `duel.h` and an object file named `lab05_main.o` in the directory

```
/data/biocs/b/student.accounts/cs135_sw/cs136labs/lab05
```

You are to copy these files into your working directory and create a file there named `duel.cpp`. The header file contains the prototypes of four functions that you must implement in `duel.cpp`. After you have successfully written `duel.cpp` and compiled without error, create the executable program using the instructions that I wrote above. Run the executable, `lab05`. It will display a prompt to enter the number of duels to run. Enter a number and look at the output. Aaron should win about 36 to 37% of the time, Bob, about 41 to 42%, and Charlie, about 21 to 22%. If you make the number of duels large enough, this is what you should see. If not, your implementation is flawed.

***Advice***: To simplify writing the `.cpp` file, and avoid typing mistakes, copy `duel.h` into `duel.cpp` and replace the semicolons by curly braces.

***Hint***: How can you make a program simulate an event that happens with probability $p$, where $0 \leq p \leq 1.0$? Suppose you generate a random number between 0 and 1. If it is less than or equal to $p$, then the event happened. If it is greater than $p$, then it did not.

### What to Submit

Submit your implementation file, however complete it is, by the end of today's lab, i.e., before the end of the class at 2:00 P.M. The instructions are just like the previous lab's:

1. Create a directory in
   `/data/biocs/b/student.accounts/cs135_sw/cs136labs/lab05/submissions`
   whose name is your *username*.
2. Copy your file, renaming it to *`username`*`_duel.cpp`. You will lose 5% of the grade if you misname the file!
3. Change the permission on the directory that you created so that no one else can read it. You do this with the commands
   ```
   $ cd /data/biocs/b/student.accounts/cs135_sw/cs136labs/lab05/submissions
   $ chmod 700 username
   ```

***Do not submit executable files***. Remember to document your code (both preamble and comments in the code) and make it easy to read. Your work will be graded based on the rubric outlined in the Programming Rules document.

There are absolutely no extensions to the deadline. You can submit a revised version of the program by 10:00 P.M. on Thursday, Sept. 27 for partial credit. If you do, you must name it with a different name than what you first posted, e.g., *`username`*`_duel_v2.cpp`. and put it in the same directory as the original. It must be a *revision* of the original file, with only fixes to things that were not working before. I will adjust the grade based on how well the first version worked and how many changes you made.