CISC 1600—Computer Science I
Spring, 2013

Programming Project #3: Bulls and Cows
Date Due: Thursday 8 April 2015

In Exercises 5.12 and 5.13, the text mentions a two-player number-guessing game called "Bulls and Cows", to be played between a the computer and a person.[1] The computer thinks of a secret number sequence, consisting of four distinct numbers, each in the range $\{0, 1, \ldots, 9\}$. The human opponent has to determine these four numbers, in their proper sequence, by making repeated guesses. A guess of a number in its proper slot is called a "bull", whereas a guess in a number that's not in its proper slot is called a "cow". For example, if the secret number sequence is `3, 1, 4, 5` and the human opponent guesses `3, 2, 5, 4`, then the computer would output

```
Bulls: 1, Cows: 2
```

There are two little wrinkles to consider here:

1. The human opponent might want to give up, in ignominious defeat. He can do this by giving a negative number as one (or more) of the inputs. In that case, your program should magnanimously display the correct solution.

2. The human opponent might accidentally type something that doesn't fit the proper input pattern. Your program should do something reasonable when this happens; this is a good chance for you to learn something about using exceptions.

Let's discuss the overall logic of the game. First of all, it should ask the player whether she wants instructions, giving them if asked. It should now repeatedly play the game until the player tells the program to quit. To play the game, the program should randomly-generate four numbers in the range $\{0, 1, \ldots, 9\}$. It should now repeatedly ask the human player for a guess, processing it as follows:

1. If the guess is correct, then the current game is over (and a congratulatory message should be issued).

2. If the guess is incorrect, then the human player should be told the number of bulls and cows.

How can one accomplish step 2 above? First of all, counting the number of bulls is easy. Suppose that we store the solution and the guess as `vector<int>`. Then we count the bulls by doing an element-by-element comparison of these two vectors, adding 1 to the number of bulls (encountered so far) every time these two elements match up.

To count the number of cows,[2] let `solution_frequency` be a `vector<int>` defined by

$$\texttt{solution\_frequency[i]} = \text{number of times } \texttt{i} \text{ appears in the solution } (0 \leq \texttt{i} < 10)$$

and let `guess_frequency` be defined analogously. Then the total number of "hits" (either bulls or cows) is given by the sum of

$$\min(\texttt{solution\_frequency[i]}, \texttt{guess\_frequency[i]}) \qquad (0 \leq \texttt{i} < 10)$$

and hence the number of cows is given by

$$\text{number of hits} - \text{number of bulls}$$

Here's an example (using a "debugging" version of `proj3` that shows the frequency vectors):

[1] Actually, the other person doesn't really have to be a person. It could be another computer program.

[2] This is a really neat solution, but it's somewhat subtle. I would not expect you to think this up on your own. That's why I'm telling it to you.

```
sobolev@dsm:proj3$ proj3-debug
Need help (0/1)? 0
Actual solution: 3 6 7 5
Guess #1? 3 3 2 1
index:           0 1 2 3 4 5 6 7 8 9
                 --------------------
soln frequency:  0 0 0 1 0 1 1 1 0 0
guess frequency: 0 1 1 2 0 0 0 0 0 0
minimum:         0 0 0 1 0 0 0 0 0 0
sum minimum:     1
Bulls: 1, cows: 0

Guess #2? 2 3 1 4
index            0 1 2 3 4 5 6 7 8 9
                 --------------------
soln frequency:  0 0 0 1 0 1 1 1 0 0
guess frequency: 0 1 1 1 1 0 0 0 0 0
minimum:         0 0 0 1 0 0 0 0 0 0
sum minimum:     1
Bulls: 0, cows: 1

Guess #3? 3 6 7 5
index:           0 1 2 3 4 5 6 7 8 9
                 --------------------
soln frequency:  0 0 0 1 0 1 1 1 0 0
guess frequency: 0 0 0 1 0 1 1 1 0 0
minimum:         0 0 0 1 0 1 1 1 0 0
sum minimum:     4
Bulls: 4, cows: 0

Congratulations!

Play again (0/1)? 0
sobolev@dsm:proj3$
```

Some things to think about:

1. You are to do this using only the techniques found in Chapters 1 through 5 of the text. In other words, you don't need to read ahead to figure out how to do this. As always, you *must* design and code your solution before you set foot into the lab.

2. You should do this project, in a subdirectory `proj3` of your `~/private/cs1` directory. See the Project 1 handout for further discussion, and adjust accordingly.

3. The "share directory"

   `~agw/class/cs1/share/proj3`

   (on the Departmental Linux machines) for this project contains two executable versions of this program, named `proj3-agw` and `proj3-agw-debug`, along with a stub version `proj3-stub.cc`, which you should use as the starting point for your solution. All of these should be copied to your working directory for this assignment.

   - The difference between `proj3-agw` and `proj3-agw-debug` is that the latter prints out the display shown above, giving the full solution, as well as the frequency vectors and their minima, along with the

sum minimum; it also will produce the same pseudorandom numbers for the solution vectors each time you play it. The former simply plays the game, with different pseudorandom numbers for the solution vectors each time the game is played. In other words, you should use the debug version when writing and debugging the program. To go back and forth between the versions, make sure that the line

```
const bool debug = false;
```

has the obvious value.

- The C++ file `proj3-stub.cc` gives you the "scaffolding" that will allow you to concentrate on the game logic. See the next item for details.

4. In my design, `main()` does the following:

   - Offer help, if needed, using a function named `offer_help()`.
   - Repeatedly do the following:
     - Generate a solution. This is handled by a function named `generate_solution()`.
     - Play one game, with the given solution. This is handled by a function named `play_one_game()`.
     - If the player won the game, issue a congratulatory message; otherwise, issue a sympathy message and display the actual solution.
     - Ask whether player wants to play another game, starting a new game if answer is in the affirmative.

   I highly suggest that you follow this design, unless you can come up with something that you can convince me is better.

   Here's a more detailed description of the functions involved:

   - The function with prototype

     ```
     void offer_help();
     ```

     asks the player whether she wants help, giving help if the answer is in the affirmative. *This is a function you need to write yourself.*

   - The function with prototype

     ```
     vector<int> generate_solution(int num_slots, int range_top);
     ```

     generates a Bulls and Cows solution, returning it as a vector. The first parameter is the number of slots in the solution vector, and the second is an upper bound on the values that each solution element can attain. More precisely, after executing the statement

     ```
     vector<int> solution = generate_solution(num_slots, range_top);
     ```

     the elements of `solution` will be distinct, randomly-chosen values in the range `[0, range_top)`. Since you are not expected to know anything about random numbers at this point in your career, the file `proj3-stub.cc` contains this function's definition.

   - The function with prototype

     ```
     bool play_one_game(vector<int> solution, int range_top);
     ```

     plays one game. Here, `solution` is the vector containing the solution,[3] with `range_top` the same as in `generate_solution()`. The return value is `true` if the player guesses the solution, and `false` if the player gives up in disgust. *This is a function you need to write yourself.*

   - The function with prototype

     ```
     int count_cows(int bulls, vector<int> guess,
                    vector<int> solution, int range_top);
     ```

[3] I hope this doesn't come as too much of a shock.

assumes that `bulls` is the number of bulls, that `guess` is the current vector of guesses, and that `solution` is the vector containing the solution. Here, `range_top` is the same as above. This function returns the number of cows associated with all this data. I found the function `compute_frequency()`, mentioned below, to be a big help here. *This is a function you need to write yourself.*

- The function with prototype

```
void print_debug_info(vector<int> solution_frequency,
                      vector<int> guess_frequency,
                      vector<int> min_frequency, int total_hits);
```

  prints a display similar to that found on page 2 of this handout. If you're running in debug mode, then `count_cows()` should invoke same after each set of guesses has been read and processed. I have magnanimously given you the full implementation of this function in `proj3-stub.cc`.

- The function with prototype

```
vector<int> compute_frequency(vector<int> data, int range_top);
```

  returns a frequency vector. The *i*th element of the frequency vector is the number of time that *i* appears in the `data` vector. Once again, `range_top` is as above. *This is a function you need to write yourself.*

To make things absolutely clear: your task consists of defining the functions that only exist in stub form.

The file `proj3-stub.cc` contains stub versions of all the functions that you are to implement. It should compile "as is" (although it won't do anything very interesting).

5. For what it's worth, I found that actually playing the game (as a human competitor) was harder than writing the program.

6. Use the `photo` program so you can have something to turn in. *Don't do this until the program is working!* The commands you should issue are as follows:

```
photo
cat proj3.cc
g++ -o proj3 proj3.cc
proj3
exit
```

   You are to use the non-debugging version here; I don't want to see the debug info.

   You should play enough games to convince me that the program works properly. This may done by playing several games within one execution of `proj3`. Alternatively, you can run `proj3` several times, playing one game for each execution of `proj3`. Or you can do some combination of the above. Once again, see previous project handouts if you're still unclear about `photo`, as well as the relevant entry in the Departmental online help document, found at

`http://www.dsm.fordham.edu/help`

Good luck!