Universidade Federal do Rio Grande do Norte
Instituto Metaópole Digital

Programming Language I • IMD0030

◁ Keno Game Programming Assignment ▷
16 de março de 2016

1 Introduction

In this programming assignment you are going to write a program that plays a simple text-based version of the game Keno. A free online graphic version of the game can be found here.

Keno is a popular gambling game with similarities to a lottery or bingo. Players place a bet and choose anywhere from one to twenty numbers between 1 and 80, inclusive. Once the players have chosen their numbers, the game generates twenty random numbers between 1 and 80, and players receive a payoff based on how many numbers they picked that matched the chosen numbers.

For example, if a player picked seven numbers and all seven were chosen (very unlikely!), she might win around $\$10.000$ for a ten dollar bet. The actual payoffs are based on the probabilities of hitting k numbers out of n chosen.

This assignment will introduce you to classes in C++, by offering you an opportunity to design and program a real world based application.

2 Game Overview

To begin a game, your program should be able to read a player's bet from a ASCII file. The player's bet file name should be passed into the program as arguments of the function `main()`, from the command line. Remember that the integer parameter `int argc` (argument count) is the number of arguments passed into the program, whereas the `char * argv[]` is the listing of all arguments passed into the program, including the program's name, stored at `argv[0]`.

Here is how you should provide a bet file to the keno program.

```
>./keno bet_12spots.dat
```

You find below an example of a valid bet file.

```
1500.0
3
21 12 64
```

As you may notice, a typical bet file has a set of three data lines, representing:

- The player's initial credit (`IC`), expressed as a real value.
- The number of rounds (`NR`) the game should run, represented by an integer.
- A set of up to 15 unique numbers in any order, separated by at least one white-space.

Your program should validate the bet file by ensuring that all the number the player picked, called *spots*, are in the proper range $[1, 80]$, and that each number appears only once in the bet file. This means that any repeated occurrences of a number in the *spot* list should be ignored.

The input file may have blank lines between valid lines, and leading white-spaces at the beginning of any valid line. In case the bet file contains more than 15 numbers, your program should consider only the first 15 unique valid numbers.

2.1 Gameplay

After a valid bet is processed, the game runs `NR` rounds, waging in each round `IC/NR` credits. For each individual round the game should randomly pick 20 winning numbers, the *draw*, and display them on the screen. The player's bet numbers, i.e. the *spots*, are then compared to the *draw* numbers to determine how many of them match. The set of correct numbers picked by the player are called *hits*. The number of *hits* determines the *payout factor*, which, in turn, is multiplied by the round's wage to determine whether the player wins or looses credits.

The payout factor comes from the *payout table*. A payout table is chosen based on the number of *spots* present in a bet, and it determines the player's expected return, based on the number of *hits* obtained in a round. Table 1 shows all the 15 possible payout tables.

	Hits															
# spots bet	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	3														
2	0	1	9													
3	0	1	2	16												
4	0	0.5	2	6	12											
5	0	0.5	1	3	15	50										
6	0	0.5	1	2	3	30	75									
7	0	0.5	0.5	1	6	12	36	100								
8	0	0.5	0.5	1	3	6	19	90	720							
9	0	0.5	0.5	1	2	4	8	20	80	1200						
10	0	0	0.5	1	2	3	5	10	30	600	1800					
11	0	0	0.5	1	1	2	6	15	25	180	1000	3000				
12	0	0	0	0.5	1	2	4	24	72	250	500	2000	4000			
13	0	0	0	0.5	0.5	3	4	5	20	80	240	500	3000	6000		
14	0	0	0	0.5	0.5	2	3	5	12	50	150	500	1000	2000	7500	
15	0	0	0	0.5	0.5	1	2	5	15	50	150	300	600	1200	2500	10000

Tabela 1: The payout table for up to 15 *spots*. Each line of the table corresponds to a payout scale based on the number of *spots* in a bet (left most column). For example, suppose a player placed a 100 credits bet on a 5 *spots* ticket and got 3 hits (for which the payout rate is 3); in this case the player would receive 300 credits back.

2.2 Text-based Interface

After a valid bet is read and processed, the game should display all the bet information on the screen, as well as the corresponding payout table.

Next, the program should display, for each `NR` round, the 20 numbers randomly picked, the player's *hits*, the payout rate received, and the amount of credit won or lost.

Finally, the program displays on the screen a summary of the game, showing the total amount of money the player won/lost after finishing all drawing rounds.

See bellow one possible text-based output for the bet presented in Section 2.

```
>>> Preparing to read bet file [data/bet_03.dat], please wait...
--------------------------------------------------
>>> Bet successfully read!
    You are going to wage a total of $1500 dollars.
    Going for a total of 3 rounds, waging $500 per round.

    Your bet has 3 numbers. They are: [ 12 21 64 ]
         -------+---------
         Hits   | Payout
         -------+---------
         0      | 0
         1      | 1
         2      | 2
         3      | 16
         --------------------------------------------------
         This is round #1 of 3, and your wage is $500. Good luck!
         The hits are: [ 3 6 12 20 21 23 26 27 28 31 32 35 45 48 55 59 63 64 69 74 ]

         You hit the following number(s) [ 12 21 64 ], a total of 3 hits out of 3
         Payout rate is 16, thus you came out with: $8000
         Your net balance so far is: $9000 dollars.
         --------------------------------------------------
         This is round #2 of 3, and your wage is $500. Good luck!
         The hits are: [ 2 3 7 10 15 17 18 21 23 28 29 33 37 40 41 43 50 71 72 79 ]

         You hit the following number(s) [ 21 ], a total of 1 hit out of 3
         Payout rate is 1, thus you came out with: $500
         Your net balance so far is: $9000 dollars.
         --------------------------------------------------
         This is round #3 of 3, and your wage is $500. Good luck!
         The hits are: [ 4 8 10 16 20 23 28 30 32 34 45 46 50 51 52 63 64 68 74 80 ]

         You hit the following number(s) [ 64 ], a total of 1 hit out of 3
         Payout rate is 1, thus you came out with: $500
         Your net balance so far is: $9000 dollars.
>>> End of rounds!
--------------------------------------------------

===== SUMMARY =====
>>> You spent in this game a total of $1500 dollars.
>>> Hooray, you won $7500 dollars.  See you next time! ;-)
>>> You are leaving the Keno table with $9000 dollars.
```

3 Implementation Details

You program should present at least one class. We suggest to model a keno bet as the `KenoBet` class, which should follow the organization presented in Code 1.

While developing the game, try to use as much as possible the functions from the `<algorithm>` library. They are design to simplify typical programming tasks (called *boilerplate code*) in an efficient fashion.

It is very likely that you program will need to sort arrays of data. In this case, you **should** code

Code 1 The `KenoBet` class header.

```cpp
class KenoBet {
    public:
        //! Creates a Keno bet. It defines the maximum spots a bet may have.
        explicit KenoBet( unsigned int _maxNumSpots = 15 );
        // Adds a number to the spots only if the number is not already there.
        // @param _spot The number we wish to include in the bet.
        // @return T if number chosen is successfully inserted; F otherwise.
        bool addNumber( int _spot );
        // Sets the amount of money the player is betting.
        // @param _wage The wage.
        // @return True if we have a wage above zero; false otherwise.
        bool setWage( float _wage );
        //! Resets a bet to an empty state.
        void reset( void );
        // Retrieves the player's wage on this bet.
        // @return The wage value.
        float getWage( void ) const;
        // Returns to the current number of spots in the player's bet.
        // @return Number of spots present in the bet.
        size_t numChosen( void ) const;
        // Determine how many spots match the hits passed as argument.
        // @param _hits List of hits randomly chosen by the computer.
        // @return An vector with the list of hits.
        std::vector< int >
        getHits( std::vector<int> & _hits ) const;
        // Return an vector<int> with the spots the player has picked so far.
        // @return The vector<int> with the player's spots picked so far.
        std::vector<int> getSpots( void ) const;
    private:
        std::vector<int> m_spots; //<! The player's bet.
        float m_wage;             //<! The player's wage
        unsigned int m_maxSpots;  //<! Max # of spots allowed for this bet.
};
```

your own sorting routine, relying on algorithms such as quick or insertion sort. You may use the `std::sort` or `std::qsort` but only for comparison purposes, to make sure your sorting routine is working properly.

Make sure to extensively test your program. Run several test cases, with valid and invalid bet files. You may also ask friends or colleagues to test your program and give you some valuable feedback regarding the user interface, and the fun factor.

4 Project Evaluation

You should hand in a complete program, without compiling errors, tested and fully documented. The assignment will be credit according to the following criteria:-

1. Receives input data via command line (5 credits);
2. Correctly handles the input bet file, treating both regular and problematic cases accordingly (20 credits);
3. Codes correctly the `KenoBet` class, according to the description provided in Code 1 (25 credits);
4. Executes correctly the amount of rounds defined in the input bet file (15 credits);
5. Identifies correctly the *hits* and the player's payoff for every round (20 credits), and;
6. Displays correctly the information requested in Section 2.2 (15 credits).

The following situations may *take credits out* of your assignment, if they happen during the evaluation process:-

- Compiling and/or run time errors (up to -20 credits)
- Missing code documentation in Doxygen style (up to -10 credits)
- Memory leak (up to -10 credits)
- Missing README file (up to -20 credits).

The README file (Markdown file format recommended here) should contain a brief description of the game, and how to run it. It also should describe possible errors, limitations, or issues found. Do not forget to include the author(s) name(s)!

Extra credits may be awarded to those assignments which have been correctly and completely finished, and present a graphical user interface developed with SFML. An example of such interface is shown in Figure 1.

Good Programming Practices

During the development process of your assignment, it is strongly recommend to use the following tools:-

- Doxygen: professional code documentation;
- Git: version control system;
- Valgrind: tracks memory leaks, among other features;
- gdb: debugging tool, and;
- Makefile: helps building and managing your programming projects.

Try to organize you code in several folders, such as `src` (for `.cpp` files), `include` (for header files `.h`), `bin` (for `.o` and executable files) and `data` (for storing input files).

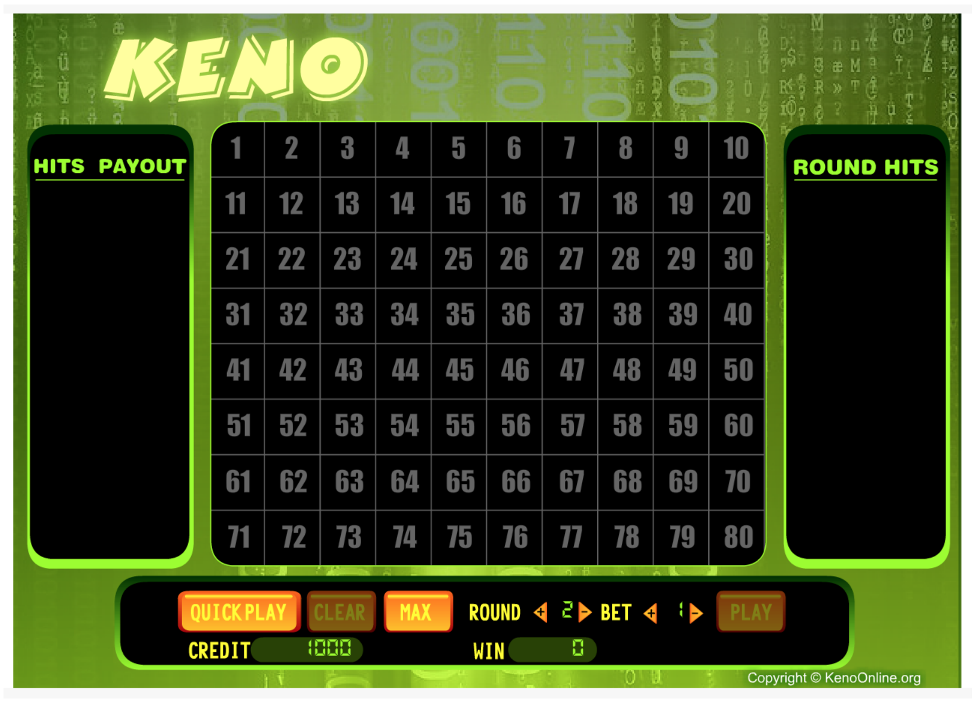

Figura 1: Screenshot of a Keno online game, found at http://www.kenoonline.org.

5 Authorship and Collaboration Policy

This is a pair assignment. You may work in a pair or alone. If you work as a pair, comment both members' names atop every code file, and try to balance evenly the workload. Only one of you should submit the program via Sigaa.

Any team may be called for an interview. The purpose of the interview is twofold: to confirm the authorship of the assignment and to identify the workload assign to each member. During the interview, any team member should be capable of explaining any piece of code, even if he or she has not written that particular piece of code. After the interview, the assignment's credits may be re-distributed to better reflect the true contribution of each team member.

The cooperation among students is strongly encouraged. It is accepted the open discussion of ideas or development strategies. Notice, however, that this type of interaction should not be understood as a free permission to copy and use somebody else's code. This is may be interpreted as plagiarism.

Any two (or more) programs deemed as plagiarism will automatically receive **zero** credits, regardless of the real authorship of the programs involved in the case. If your project uses a (small) piece of code from someone else's, please provide proper acknowledgment in the README file.

6 Work Submission

Only one team member should submit a single zip file containing the entire project. This should be done only via the proper link in the Sigaa's virtual class.

◀ The End ▶