Projeto 1 - Linguagem de Programação 1
Implementação do Jogo Keno, com interface textual

1 Introdução

Neste trabalho iremos desenvolver uma versão com interface textual do jogo Keno. Uma versão online do jogo pode ser encontrada aqui.

Keno é um jogo de apostas muito popular parecido com loteria ou bingo. Os jogadores apostam escolhendo entre 1 e 20 números únicos no intervalo entre 1 e 80, inclusive. Quando os jogadores escolhem seus números, o jogo gera vinte números aleatórios entre 1 e 80. Baseado na quantidade de números que o jogador acertou, e no valor que ele apostou, ele recebe um valor de volta.

Por exemplo, se um jogador escolhe sete números e todos os sete são sorteados, o jogador pode ganhar cerca de 1000 vezes o valor apostado! A tabela de valores de retorno é baseada na probabilidade de que sejam acertados k números entre n escolhidos.

Neste trabalho também aprenderemos um pouco de Classes em C++, com o intuito de encorajar a utilização de encapsulamento e objetos que são muito importantes quando falamos de qualidade de software.

2 Resumo do Jogo

Para iniciar o jogo, seu programa deve ser capaz de ler as apostas de um arquivo de apostas em formato texto - o arquivo de apostas deve ser passado como argumento da função main() via linha de comando. Isso pode ser feito modificando a assinatura da função main para receber dois argumentos, ***argc*** e ***argv***. Um exemplo de como utilizar argumentos da função main() pode ser visto abaixo (Código 1):

Código 1 recebendo entradas a partir da linha de execução do programa.

```
int main(int argc, char *argv[]){
  //imprime todos os argumentos recebidos na execução
  //do programa, para testar, compile e faça
  //./a.out arg1 arg2 arg2
  //o primeiro argumento é sempre o nome
  //do programa (a.out, neste exemplo)
  for(int i=0; i<argc; i++)
    cout<<"Argumento["<<i<<"]: "<<argv[i]<<endl;
  return 0;
}
```

Desta forma, o arquivo de apostas deve ser fornecido como argumento durante a execução do programa, conforme pode ser visto abaixo:

```
./keno bet_12stpors.dat
```

Logo abaixo é possível visualizar um exemplo válido para um arquivo de apostas.

```
1500.0
3
21 12 64
```

As três linhas ilustradas para arquivo de apostas significam, em ordem:

- número real representando a quantidade de crédito inicial do jogador (*initial credit* ou IC);
- número inteiro representando a quantidade de rodadas que serão executadas (*number of rounds* ou NR);
- conjunto de até 15 números inteiros únicos em qualquer ordem separados por espaço (chamado de *spots*).

2.1 Validação do arquivo de apostas

Seu programa **deverá validar** o arquivo de apostas para garantir alguns requisitos. Quando algum requisito não for satisfeito, a aposta deverá ser rejeitada e uma mensagem de erro correspondente deverá ser impressa. Alguns requisitos que deverão ser satisfeitos incluem:

- Repetição de números;
- Quantidade de números na aposta superior a 15 números;
- Presença de caracteres estranhos (exemplo: strings);
- Formatação diferente daquela proposta para o arquivo de apostas;
- Arquivo de apostas inexistente.

Os requisitos descritos acima não são exaustivos. Outros requisitos poderão ser definidos caso sejam identificados.

2.2 Iniciando o jogo

Depois que uma aposta válida é processada, o jogo executa **NR** rodadas, apostando em cada uma valor igual a **IC/NR**. Para cada rodada o jogo escolhe aleatoriamente 20 números vencedores e os mostra na tela. Os números apostados pelo jogador, os *spots*, são comparados com os números sorteados para determinar quantos números o jogador acertou naquela rodada. O conjunto dos números que o jogador acertou são chamados de *hits*. O número de *hits* determina o fator de retorno que é multiplicado pelo valor apostado naquele turno, determinando, assim, se o jogador ganha ou perde dinheiro.

O fator de retorno vem da tabela de retorno (*payoff table*). A tabela de retorno determina o fator de retorno de uma rodada com base no valor apostado, no número de *spots* da aposta e na quantidade de *hits* daquela rodada. A tabela que usaremos está definida abaixo:

	hits															
spots	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: A tabela de retorno para 15 *spots*. Cada linha corresponde a uma escala de retorno com base na quantidade de *spots*. Por exemplo, suponha que um jogador aposta 100 créditos em uma cartela com 5 *spots* e obtém 3 *hits* (o valor de retorno, de acordo com a tabela é 3); Neste caso, o jogador ganha 300 créditos naquela rodada.

2.3 Interface de texto

Após ler uma aposta válida, o jogo deve mostrar todas as informações de aposta na tela bem como a tabela de retorno correspondente.Em seguida, o programa deve mostrar para cada rodada os 20 números sorteados, os *hits* do jogador, o valor de retorno obtido e a quantidade de créditos que o jogador ganhou ou perdeu.

Por fim, o programa deve imprimir um sumário do jogo, mostrando o total de créditos que o jogador perdeu ou ganhou após finalizadas as NR rodadas.

Abaixo um exemplo possível de interface:

```
>>> Lendo arquivo de apostas [data/bet_03.dat], por favor aguarde..
----------------------------------------------------------------------
>>> Aposta lida com sucesso!
    Você apostará um total de $1500 créditos.
    Jogará um total de 3 rodadas, apostando $500 créditos por rodada

    Sua aposta tem 3 números, eles são: [ 12 21 64 ]
        ----------+-----------
        Hits      | Retorno
        0         | 0
```

```
        1         | 1
        2         | 2
        3         | 16
        ------------------------------------------------------------
        Esta é a rodada #1 de 3, sua aposta é $500. Boa sorte!
        Os números sorteados são: [ 3 6 12 20 21 23 26 27 28 31 32 35 45 48 55
59 63 64 69 74 ]

        Você acertou os números [ 12 21 64 ], um total de 3 hits de 3
        Sua taxa de retorno é 16, assim você sai com: $8000
        Você possui um total de: $9000 créditos.
        ------------------------------------------------------------
        Esta é a rodada #2 de 3, sua aposta é $500. Boa sorte!
        Os números sorteados são: [ 2 3 7 10 15 17 18 21 23 28 29 33 37 40 41 43
50 71 72 79 ]

        Você acertou os números [ 21 ], um total de 1 hit de 3
        Sua taxa de retorno é 1, assim você sai com: $500
        Você possui um total de: $9000 créditos.
        ------------------------------------------------------------
        Esta é a rodada #3 de 3, sua aposta é $500. Boa sorte!
        Os números sorteados são: [ 4 8 10 16 20 23 28 30 32 34 45 46 50 51 52
63 64 68 74 80 ]

        Você acertou os números [ 64 ], um total de 1 hit de 3
        Sua taxa de retorno é 1, assim você sai com: $500
        Você possui um total de: $9000 créditos.
>>> Fim das rodadas!
------------------------------------------------------------------------

======= Sumário =======
>>> Você gastou um total de $1500 créditos
>>> Hooray! você ganhou $7500 créditos!
>>> Você está saindo do jogo com um total de $9000 créditos.
```

3 Detalhes de Implementação

Seu programa deve ter ao menos uma Classe. O modelo sugerido é o modelo abaixo, conforme ilustrado no código 2 através da classe ***KenoBet***.

Ao desenvolver seu programa utilize preferencialmente as funções e/ou estruturas da STL, uma vez que elas são projetadas para resolver problemas genéricos de forma mais eficiente - essa é uma das principais diferenças entre C e C++.

É provável que seu programa precise usar ordenação de arrays, neste caso você **deve escrever** sua própria função de ordenação, usando algoritmos como o *quick sort* ou *insertion sort*. Você pode usar os algoritmos de ordenação da STL **`std::sort`** ou **`std::qsort`** apenas para comparações.

Código 2 - classe KenoBet

```
using number_type = unsigned short int; //<! data type for a keno hit.
using cash_type = float; //<! Defines the wage type in this application.
using set_of_numbers_type = std::vector< number_type >;

class KenoBet {
    public:
        //! Creates an empty Keno bet.
        KenoBet( ) : m_wage(0)
        { /* empty */ };

        /*! 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 add_number( number_type 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 set_wage( cash_type wage_ );

        //! Resets a bet to an empty state.
        void reset( void );

        /*! Retrieves the player's wage on this bet.
            @return The wage value. */
        cash_type get_wage( void ) const;

        /*! Returns to the current number of spots in the player's bet.
            @return Number of spots present in the bet. */
        size_t size( 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. */
        set_of_numbers_type
        get_hits( const set_of_numbers_type & hits_ ) const;

        /*! Return a vector< spot_type > with the spots the player has picked so far.
            @return The vector< spot_type > with the player's spots picked so far. */
        set_of_numbers_type get_spots( void ) const;

    private:
        set_of_numbers_type m_spots;  //<! The player's bet.
        cash_type m_wage;             //<! The player's wage
};
```

3.1 Sobre a entrega

O código deverá vir acompanhado de arquivos de apostas que servirão para validar o seu programa.

Além do uso de classe(s), a entrega deverá incluir um arquivo **README** no formato Markdown, contendo as seguintes informações:

- Como compilar o projeto;

- Como executar o projeto;
- Como executar o conjunto dos testes planejados por você (ou grupo).
- Limitações ou funcionalidades não implementadas no programa.

Boas Práticas de Programação

No desenvolvimento do projeto, recomendamos preferencialmente o uso das seguintes ferramentas:

- **Doxygen**: ferramenta de geração automática de documentação de código;
- **Valgrind e/ou *address sanitizer***: ferramentas de identificação de acesso inválido à memória ou *memory leaks*;
- **GDB:** ferramenta de *debugging*;
- **Makefile ou Cmake**: ferramentas de automação do processo de compilação.

Tente organizar seu código em diferentes diretórios, por exemplo </*src*> para arquivos .cpp, </*include*> para arquivos .hpp e .h, </*bin*> *ou* </*build*> para os .o ou executáveis e </*data*> para arquivos de entrada do programa.