Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

Linguagem de Programação I • IMD0030
– 1ª Avaliação, 29 de março de 2016 –

A PROVA TEM A DURAÇÃO DE **100 MINUTOS**.
LEIA ATENTAMENTE AS INSTRUÇÕES ABAIXO ANTES DE INICIAR A PROVA

- ⋆ *Esta é uma prova individual ou em dupla com consulta restrita.*
- ⋆ *Você terá 100 minutos para a realização da prova. As respostas devem ser fornecidas na forma de um arquivo digital compactado contendo seus programas.*

1 Apresentação

Nesta avaliação você deve implementar uma versão baseada em texto do jogo de cartas **Super Trunfo**. A implementação do jogo deverá ser realizada em duas etapas, com submissões via Sigaa separadas. A primeira etapa deve ser realizada no horário definido para esta avaliação, enquanto que a segunda deve ser realizada e entregue em até 24h contados a partir de 11h de hoje.

2 Gameplay do Jogo Super Trunfo

O jogo é baseado em um conjunto de cartas como as da Figura 1. Cada carta possui um *identificador* único e um conjunto de *atributos* relacionados com o tema das cartas. O exemplo da Figura 1 tem como tema aviões de caça. Nele é possível observar os identificadores no canto superior esquerdo, como "D1", "E1", etc.; os identificadores são formados por uma letra e um número entre 1 e 9. Os atributos do exemplo são informações sobre aviões como velocidade, comprimento, envergadura, etc.

Figura 1: Exemplo de cartas de Super Trunfo de Aviões de Caça. A primeira carta é marcada como Super Trunfo.

Na preparação da partida cada jogador (mínimo 2) recebe um certo número k de cartas a partir do *deck* original[1], formando seu *monte* individual de cartas. Após a partilha das cartas uma partida pode ser iniciada.

Uma partida de Super Trunfo é organizada em rodadas. Em uma rodada, cada jogador só pode utilizar a carta que estiver no topo de seu monte. Um jogador escolhido aleatoriamente inicia a primeira rodada indicando qual atributo ele deseja comparar com os demais jogadores. Todos os jogadores, então, depositam sobre a mesa suas respectivas cartas retiradas do topo de cada monte; vence a rodada o jogador que possuir a carta com o *maior* valor para o atributo indicado pelo jogador que iniciou a rodada. O jogador vencedor recolhe todas as cartas da rodada e as armazena embaixo de seu monte. O vencedor também inicia a próxima rodada escolhendo um novo atributo da (nova) carta que está no topo do seu monte. As rodadas se sucedem até que um dos jogadores fique com todas as cartas, sendo o vencedor da partida.

Carta Super Trunfo

A carta marcada como Super Trunfo (ST) bate todas as outras cartas independente dos dados dos atributos das cartas dos oponentes. Contudo, se a carta de algum oponente for do tipo "$X1$", onde X é uma letra qualquer, a carta ST perde seu poder. Por exemplo, em uma rodada com 3 jogadores e as seguintes cartas da rodada $\{$ "ST", "D2", "E4" $\}$ vence o "ST", independente do valor do atributo selecionado. Por outro lado, se as cartas fossem $\{$ "ST", "D2", "E1" $\}$, a "E1" teria vencido a disputa.

Se em uma rodada com "ST" houver mais de uma carta no formato "$X1$", sendo X uma letra, o vencedor é escolhido com as regras normais aplicadas apenas entre as cartas do formato "$X1$".

3 Tarefa: Implementar Super Trunfo

Nesta avaliação você deve implementar uma simulação de um partida Super Trunfo com jogadas automáticas. Em outras palavras, a partir de uma configuração inicial de jogadores e cartas recebidas, seu programa deve realizar uma simulação de jogadas sucessivas e automáticas, até o final da partida, e determinar o vencedor.

Os dados para configurar uma partida serão fornecidos através de *argumentos de linha de comando*. Os argumentos devem, necessariamente, ser fornecidos na seguinte ordem:

- ⋆ Um nome de arquivo ASCII contendo as cartas.
- ⋆ Quantidade de jogadores: um inteiro $1 < n \leq 10$ representando a quantidade de jogadores da partida.
- ⋆ Quantidade de cartas por jogador: um inteiro $0 < k \leq 20$ representando a quantidade de cartas que devem ser distribuídas a cada jogador no início da partida.

Veja abaixo um exemplo de chamada do programa `supertrunfo`, na qual o programa lê as cartas do arquivo "`cartas.dat`", configura e executa uma simulação de partida Super Trunfo para 4 jogadores, cada um recebendo 5 cartas.

[1] Em algumas versões todas as cartas do *deck* são distribuídas entre os jogadores.

```
$ ./supertrunfo cartas.dat 4 5
```

Seu programa deve validar os argumento de entrada e certificar-se que a quantidade de cartas lidas é suficiente para obedecer a configuração solicitada. Em caso de falha, você deve emitir uma mensagem explicando o problema ocorrido, como falha na leitura do arquivo, quantidade de cartas insuficientes, valores de n e k fora da faixa aceitável, etc.

Junto com este arquivo PDF você receberá *(a)* um arquivo `saida.txt` contendo um exemplo de saída de uma simulação, que pode ser usado como modelo, e; *(b)* um arquivo `cards.dat` contendo a descrição textual de 40 cartas Super Trunfo.

3.1 Leitura das cartas

Para facilitar o processo de leitura, assuma que o arquivo não possui erros de formatação. A descrição de uma carta é formada por 8 linhas representando, respectivamente, o ID da aeronave, nome, fabricante (e país de origem), ano de fabricação do modelo, velocidade de cruzeiro (em Km/h), autonomia de voo (em Km), comprimento (em metros) e envergadura (em metros). Cada carta é separada da definição da próxima por exatamente uma linha em branco. Veja abaixo um exemplo de definição de uma carta.

```
A1
Boeing 737-2C3/Adv
Cruzeiro - Brazil
1967
927
4260
30.53
28.35
```

3.2 Classes básicas

Você deve começar desenvolvendo uma modelagem para o problema baseado em pelo menos 3 classes, descritas a seguir. Para cada classe são sugeridos alguns métodos, mas outros podem ser definidos para se adequar ao projeto do programa.

- ⋆ `STCard`: Representa uma carta de Super Trunfo. Uma carta possui uma lista de atributos, que podem ser representados por `std::string`.
  - `Construtor`: recebe os atributos da carta via argumento e os guarda internamente.
  - `getAttribute`: retorna o valor de um atributo especificado.
  - `displayCard`: imprime os dados de uma carta na saída padrão.
- ⋆ `STPlayer`: Representa um jogador de Super Trunfo. Um jogador possui uma coleção de objetos do tipo `STCard`.
  - `Construtor`: recebe como argumento o nome do jogador.
  - `name`: retorna o nome do jogador.
  - `popCard`: remove e retorna a carta no topo do monte de cartas do jogador.
  - `chooseRandomAttrib`: escolhe aleatoriamente um atributo e o retorna. Este método é usado pelo jogador da rodada para escolher qual atributo jogar.

  - `addCard`: adiciona no final do monte uma carta passada como argumento.
  - `displayCards`: exibe na saída padrão as cartas atuais do jogador.
  - `nCards`: retorna a quantidade de cartas que o jogador possui atualmente.

- ⋆ `STGame`: Implementa o gerenciamento de uma partida Super Trunfo. Esta classe armazena o *deck* original de cartas `STCard` e uma lista de jogadores `STPlayer`.
  - `readDeckFromFile`: carrega as cartas do arquivo em um container de `STCards`.
  - `addPlayer`: adiciona um jogador passado como argumento à lista de jogadores ativos.
  - `dealCards(k)`: distribui k cartas a cada uma dos jogadores presentes.
  - `displayDeck`: exibe na saída padrão o conjunto completo de cartas lidas.
  - `displayPlayers`: exibe na saída padrão cada jogador ativo e suas cartas.
  - `run`: método que contém o laço principal que simula as várias rodadas de uma partida. Quando este método termina, uma partida chegou ao fim.
  - `displayStatusRound`: exibe o estado atual de uma rodada, apresentando os jogadores ativos com suas respectivas cartas e o atributo escolhido pelo jogador da vez.
  - `displayCardsOnTable`: exibe na saída padrão o conjunto de cartas que cada jogador depositou sobre a "mesa" em uma rodada.
  - `winner`: retorna o jogador vencedor, após o término do método `run()`.

3.3 Código Cliente

Veja abaixo como seria uma visão em alto nível do código cliente instanciando as classes da Seção 3.2.

```
int main( int argc, char *argv[] ) {
    std::string cardsFileName;  // Holds the cards file name.
    std::ifstream inputFile;    // Stream object to handle the input file.

    // (1): Process any command line arguments.

    // (2): Create a game object.
    STGame myGame;
    if ( ! myGame.readDeckFromFile( inputFile ) ) {
       // ERROR MESSAGE GOES HERE.
    }
    // (3): Show deck on screen.
    myGame.displayDeck();
    // (4): Create and add 3 players to the game object.
    myGame.addPlayer( new STPlayer("Player #1")  );
    myGame.addPlayer( new STPlayer("Player #2")  );
    myGame.addPlayer( new STPlayer("Player #3")  );
    // (5): Deal the same amount of cards to all players.
    if ( ! myGame.dealCards( N_CARDS ) )// Deals N_CARDS to each player. {
        // ERROR MESSAGE GOES HERE
    }
    // (6): Run a game with several rounds.
    auto nRounds = myGame.run();
    // (7): Get winner and display the winner.
    auto winner = myGame.winner();
    std::cout << "\n\n>>> The winner after " << nRounds
              << " rounds played is: \"" << winner->name() << "\"\n\n";

    return EXIT_SUCCESS;
}
```

4 Parte II: Implementar um Gameplay Diferente para Super Trunfo

Na versão original do Super Trunfo (descrita na Seção 2), para cada rodada, após a definição do atributo, são consideradas apenas as cartas que estão no topo dos montes de cada jogador.

Neste exercício você deve implementar uma outra forma de rodar a simulação do jogo Super Trunfo. Nela você deve simular uma "Inteligência Artificial" (IA) rudimentar que escolhe do monte de cada jogador a carta que possui o maior valor para o atributo da rodada. Esta é a carta que deve ser depositada na "mesa" virtual. Tente desenvolver uma IA que utiliza a carta "ST" de maneira a obter, sempre que possível, o maior benefício em uma rodada. Sugere-se, então, a criação do método `chooseCardAI(attrib)` na classe `STPlayer`.

Provavelmente você vai precisar ordenar as cartas de alguma forma. Neste caso você **deve** usar um algoritmo de ordenação eficiente e desenvolvido por você. Não serão aceitas soluções que utilizem qualquer tipo de função de ordenação do STL ou de qualquer outra biblioteca.

O modo de simulação ora descrito deverá ser ativado por um novo argumento em linha de comando: `-ai`. Veja como ficaria a execução do programa `supertrunfo` ativando o novo *gameplay*.

```
$ ./supertrunfo cartas.dat 4 5 -ai
```

5 Avaliação e Recomendações

O trabalho pode ser desenvolvido em duplas. Você deve submeter o que conseguir desenvolver nos primeiros 100 minutos, que vai corresponder a 40% do valor total da nota, ou 40 créditos. Os outros 60 créditos serão distribuídos para o restante do projeto, entregue em até 24 horas contadas a partir das 11h do dia de hoje.

Serão levados em consideração na hora de avaliar o código critérios como: organização do projeto em pastas como arquivos separados de cabeçalho e implementação para cada classe; sua capacidade em construir classes corretamente, aderindo, sempre que possível, às sugestões fornecidas neste documento; o uso de características estudadas anteriormente como ponteiro para função, funções template e functors; correta implementação de algoritmos de ordenação (se necessário); presença de um bom gerenciamento de memória, visando criar e passar objetos de maneira eficiente; qualidade e inteligibilidade da interface textual do programa; apresentação de código claro, documentado, testado e correto, e; e domínio das classes do STL. Recomenda-se utilizar, sempre que possível, os algoritmos presentes em `<algorithm>`.

Submeta junto com o seu programa um arquivo `README.md` com informações sobre o projeto, o que funciona e o que não funciona, bem como os bugs presentes. Não esqueça de indicar os membros da equipe e fornecer instruções sobre como compilar o projeto. Forneça uma listas dos tópicos estudados que a equipe utilizou no projeto, como ponteiro para função, algoritmos de ordenação, functors, algoritmos do STL, etc.

$\sim$ FIM $\sim$