**Trabalho Prático 1**

**Poker Face**

**Carlos Henrique Brito Malta Leão**

**Matrícula: 2021039794**

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte – MG – Brasil

chbmleao@ufmg.br

1. **Introdução**

Pôquer é um famoso jogo de azar que alimenta o imaginário de muitos com promessas de riqueza rápida e, infelizmente, muitas pessoas não estão dispostas a gastar suas economias dessa forma. Pensando nisso, o problema proposto por esse trabalho prático foi a criação de um sistema onde qualquer um pode jogar, mesmo aqueles que não sabem quase anda das regras do jogo. Dessa forma, esta documentação explica como foi criado e estruturado esse sistema, em que o programa cuida da etapa de decisão, onde vê-se quem ganhou, e do montante de dinheiro virtual que cada jogador terá ao final das diversas rodadas.

Essa documentação tem como objetivo explicitar como foi realizada a implementação desse sistema, além de realizar uma análise explicativa sobre como o sistema do jogo funciona. Além disso, também serão explicitados a robustez, abstração e desempenho do programa, possibilitando um melhor entendimento de como o sistema funciona na prática.

Para resolver o problema supracitado, foi criado um programa na linguagem C++, que utiliza três principais classes: *Card*, *Player* e *Round*, que representam, respectivamente, uma carta comum de baralho, um jogador e uma rodada de pôquer. Além disso, também são utilizadas algumas classes auxiliares: *Vector*, *Node* e *List*, que representam um vetor alocado dinamicamente e uma implementação de lista encadeada alocada dinamicamente, utilizando a classe *Node* para representar cada nó encadeado. Ademais, temos a estrutura *memlog*, que auxilia no processo de análise de desempenho do programa e as funções *msgassert*, que são utilizadas de forma a aumentar a robustez do programa.

Por fim, ao decorrer dessa documentação, alguns aspectos sobre o trabalho serão melhor explicados, como a descrição da implementação na seção 2 e a análise de complexidade na seção 3. Em seguida, na seção 4 e 5 serão explicitadas as estratégias de robustez e a análise experimental. Por fim, teremos a conclusão do trabalho, sumarizando o que foi aprendido durante seu desenvolvimento.

1. **Método**

Durante essa seção, será realizada uma descrição da implementação do programa estruturado na linguagem C++. Dessa forma, serão detalhadas as estruturas de dados, classes e funções implementadas.

**2.1 Estruturas de Dados**

A implementação do programa teve como base duas estruturas de dados, um vetor dinâmico e uma lista encadeada alocada dinamicamente. Primeiramente, o vetor foi escolhido para armazenar conjuntos de objetos com uma quantidade estática, já que durante o processo de leitura do arquivo de entrada, entrada.txt, já sabemos a quantidade de rodadas, jogadores e cartas, o que torna desnecessária a implementação de uma estrutura de dados dinâmica. Porém, não há como saber, incialmente, o número de ganhadores de cada jogada, por isso, foi escolhida a implementação de uma lista encadeada que possui custo constante para a remoção de elementos no fim dela.

Primeiramente, o vetor foi implementado utilizando alocação estática, recebendo um tamanho máximo. Dessa forma, essa estrutura permite acesso aleatório a qualquer posição em tempo , além de permitir percorrer a lista em ambas direções caso necessário, começando na posição e terminando com o i-ésimo item na posição . Além disso o vetor é implementado de forma parametrizada, utilizando *templates*, dessa forma, é possível armazenar sequências de rodadas, jogadores ou cartas.

Por fim, esta classe apresenta algumas funções específicas, que abstraem e aumentam a robustez do código:

* *getItems():* Retorna o vetor de itens.
* *accessVector():* Função utilizada exclusivamente para o processo de análise de desempenho do programa.
* *writeElement(T item):* Armazena um item na próxima posição do vetor que ainda não foi armazenada.
* *readElement(int pos):* Retorna o elemento da posição desejada.
* *printVector():* Função utilizada exclusivamente no processo de desenvolvimento para a depuração do código.
* *bubbleSort():* Ordena vetores de jogadores (de acordo com seu dinheiro final) ou cartas (de acordo com sua numeração).

Um diagrama esquemático do vetor estático implementado pode ser visto na *Figura 1*.

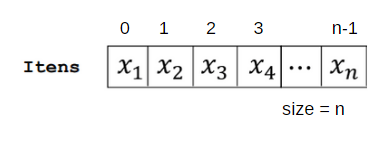


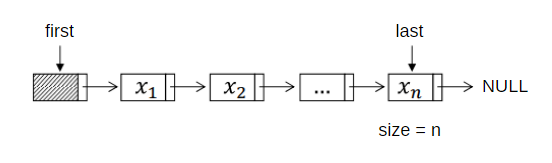
Figura – Diagrama do Vetor

Em seguida, a implementação da lista encadeada demandou uma complexidade maior que o vetor. Nela, itens da lista são armazenados em posições não contíguas da memória, dessa forma, foi necessária a utilização de células (classe *Node*) que são encadeadas usando apontadores. Essa lista utiliza alocação dinâmica, permitindo o crescimento e redução de tamanho, o que é muito útil para armazenar os vencedores de cada rodada, já que cada rodada pode apresentar mais de um vencedor.

Por fim, esta classe apresenta algumas funções específicas, que abstraem e aumentam a robustez do código:

* getSize(): Retorna o tamanho atual da lista.
* isEmpty(): Retorna se a lista está vazia.
* getPlayer(int pos): Retorna o item localizado na posição indicada.
* insertAtStart(Player\* player): Insere o item na primeira posição.
* insertAtEnd(Player\* player): Insere o item na última posição.
* insertAtPosition(Player\* player, int pos): Insere o item na posição indicada.
* removeAtStart(): Remove o item da primeira posição.
* removeAtEnd(): Remove o item da última posição.
* removeAtPosition(): Remove o item da posição indicada.
* search(string name): Retorna o item com o nome igual ao que está sendo procurado.
* print(): Função utilizada exclusivamente para o processo de depuração do código.
* clean(): Função que limpa a lista encadeada, excluindo todas as suas células.
* setPosition(int pos): Função privada auxiliar que participa do processo de inserção e remoção de itens.

Essa classe apresenta também alguns atributos privados, como *size*, que armazena o tamanho atual da lista. Além disso, existem dois endereços de memória importantes: *first* e *last*, que armazenam a primeira e a última célula da lista. O diagrama a seguir (*Figura 2*) explicita visualmente a lista encadeada implementada.



**2.2 Classes**

Para modularizar a implementação, e tornar o programa mais abstrato e prático para o usuário, foram construídas seis classes. Três delas já foram apresentadas no tópico anterior, sendo elas: *List*, *Node* e *Vector*, representando respectivamente a lista encadeada, a célula da lista e o vetor. Além dessas, também são de extrema importância as classes: *Card*, *Player* e *Round*, que representam uma carta de baralho, um jogador e uma rodada do jogo. Analisaremos cada uma delas de forma separada.

**2.2.1 *Card***

A classe Card apresenta dois atributos principais, o número (inteiros de 1 a 13) e o naipe da carta (P, O, E, C). Cada carta pode assumir um número de 1 a 13,