Trabalho Prático 1:
Poker Face

Vinicius Cardoso Antunes
Matricula: 2020027210

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

viniciusc@ufmg.br

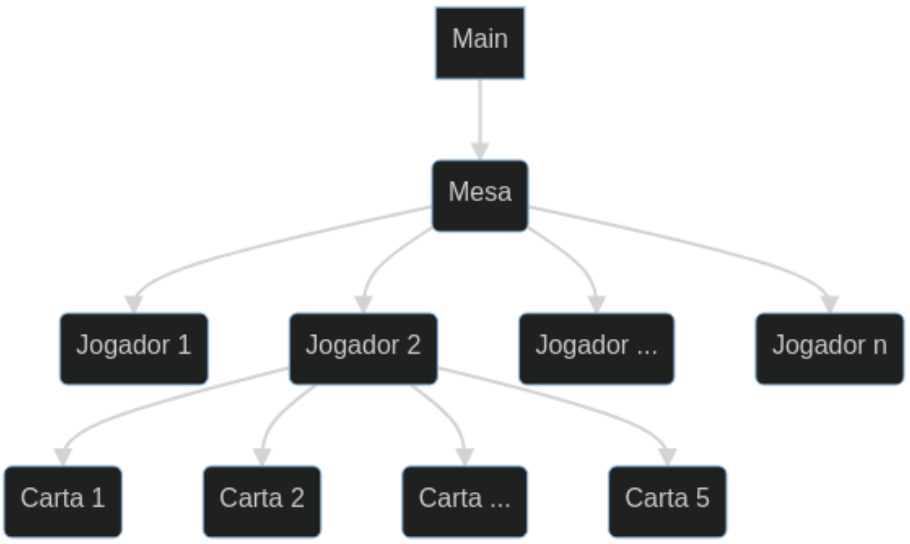
1. Introdução:

Este trabalho consiste em desenvolver um jogo de poker na linguagem C++, as interações serão realizadas através de arquivos de texto de entrada e os resultados serão colocados em uma arquivo de texto de saída de maneira ordenada. Para realizar este trabalho foi necessário conhecimentos em estruturas de dados, métodos de ordenação e programação orientada a objetos em C++.

2. Método:

2.1 Estruturas de Dados:

O projeto foi implementado utilizando listas que representam múltiplos objetos de uma classe específica, na figura abaixo pode ser melhor visualizado como as estruturas ficaram organizadas:



As classes foram implementadas em arquivos de header (.hpp) e são: Carta, Jogador e Mesa.

2.2 Classes

Carta

A classe Carta é responsável por armazenar os atributos e métodos que representam as cartas do baralho.

Atributos:

- valor (inteiro)
- naipe (caractere)
- carta (string)

Com os atributos criados dessa maneira, é possível criar uma carta a partir de uma string, como “1C”, que seria um Ás de Copas, a partir disso, essa string é convertida em valores que representam o naipe e o valor.

O naipe é definido apenas atribuindo o último elemento da string a variável naipe. Para converter a string em um valor inteiro foi necessário criar uma cópia da string, remover o último elemento e converter o valor restante utilizando a função *stoi()*.

Métodos:

- Carta()
- Carta(string carta)
- SetValor(int valor)
- GetValor() - inteiro
- SetNaipe(char naipe)
- GetNaipe() - caractere
- SetCarta(string carta)
- GetCarta() - string

Carta() e Carta(string carta) são os construtores do objeto, sendo que Carta() cria o objeto com atributos vazios e Carta(string carta) cria o objeto com os atributos passados através da string carta.

Os Setters e Getters são os responsáveis por definir e obter as propriedades de cada atributo da classe; a carta, o valor e o naipe.

Jogador

A classe Jogador é responsável por representar cada jogador da mesa, ela possui também uma constante NUMERO_DE_CARTAS, que é o valor ‘5’.

Atributos:

- nome (string)
- dinheiro (inteiro)
- valor_aposta (inteiro)
- valor_mao (inteiro)
- mao[NUMERO_DE_CARTAS] (Carta)

Os principais atributos da classe Jogador são o seu nome e o dinheiro, que são as informações mostradas no resultado final da partida.

Os outros atributos podem ser considerados auxiliares, pois eles são responsáveis por tornar o projeto mais fácil de ser implementado pois representam características temporárias dos jogadores, que podem variar de acordo com a rodada.

valor_aposta: armazena a aposta que o jogador realizou na rodada e este valor é subtraído do montante de dinheiro do jogador.

valor_mao: armazena o valor da mão do jogador como um inteiro para que seja mais simples a comparação entre os valores das mãos dos jogadores, o jogador com o maior valor é o vencedor da rodada. “HC” é representado pelo número 1, “OP” pelo 2, consecutivamente até o “RSF”, representado pelo valor 10.

mao[NUMERO_DE_JOGADORES]: armazena as 5 cartas do jogador em uma lista do tipo “Carta”.

Métodos:

- Jogador()
- Jogador(string nome, int dinheiro, int valor_aposta)
- SetNome(string nome)
- GetNome - string
- SetDinheiro(int dinheiro)
- GetDinheiro() - inteiro
- SetAposta(int aposta) - booleano
- SetValorMao(string valor_mao)
- GetValorMao() - inteiro
- *GetMao() - string
- Premiar(int premio)
- AdicionarCarta(string carta)

Os métodos Jogador() e Jogador(string nome, int dinheiro, int valor) são os contrutores da classe.

os setters e getters são utilizados referentes aos atributos nome, dinheiro, valor_aposta e valor_mao, o método SetValorMao() recebe uma string referente ao valor da mão do jogador, como “OP” e converte em um numero de valor correspondente, como foi explicado acima.

GetMao() é responsável pela lógica principal do jogo, que é ranquear a mão do jogador de acordo a lista de cartas que ele possui, esta etapa pode ser dividida em 3 partes: a análise de repetições de valor, as repetições de naipe e a sequência.

A partir das repetições de valor, pode-se verificar se a mão forma 1 ou 2 pares, uma trica ou quadra, além disso, se há uma trinca e um par, o jogador forma um *Full House*.

Na repetição de naipe, é possível verificar se a mão formou um *Flush*.

A sequência é verificada analisando se todas as cartas são diferentes e se a primeira e a última tem uma distância de 4 de valor, isso só é possível pois as cartas são ordenadas na mão a partir de outro método que será explicado em breve.

Os demais casos, como o *Straight Flush*, *Royal Straight* são obtidos quando a mão possui múltiplas características, como ser um *Flush* e um *Straight* ou sendo um *Straight Flush* de 10 a 13 e com um 1 (Ás) no início.

Premiar(): realiza a premiação do vencedor adicionando o prêmio ao seu montante de dinheiro.

AdicionarCarta(): adiciona as cartas a mão do jogador a partir da lógica de que se o primeira carta do jogador for “VAZIO”, ele adiciona a carta na primeira posição, caso não, adiciona na próxima e assim consecutivamente. Quando a mão do jogador está cheia, as cartas são ordenadas a partir do método de Ordenação *Bubble Sort*, que neste caso tem uma boa eficiência pois o número de cartas na mão do jogador é pequeno (5 cartas).

Mesa:

A classe Mesa é a classe principal, é nela que ocorrem as rodadas e os jogadores são criados, possui a maior parte dos atributos que são passados pelo jogador.

Atributos:

- numero_de_rodadas (inteiro)
- dinheiro_inicial (inteiro)
- numero_de_jogadores (inteiro)
- valor_do_pingo (inteiro)
- valor_aposta (inteiro)
- pote (inteiro)
- nome_jogador (string)
- jogada (string)
- carta (string)
- valor_mao (string)
- nome_vencedor (string)
- jogada_vencedor (string)
- invalida (booleano)
- jogadores (Jogador)

numero_de_rodadas: numero de vezes que os jogadores irão fazer as suas apostas na mesa, no final de cada rodada é gerado um pote, que é o montante de dinheiro apostado por todos os jogadores.

dinheiro_inicial: montante inicial de dinheiro de cada jogador

numero_de_jogadores: número de jogadores que irão apostar na rodada, não confundir com o número total de jogadores, que é o número de jogadores na primeira rodada.

valor_do_pingo: aposta mínima de cada jogador, é definida no início de cada rodada.

valor_aposta: valor, além do pingo, que cada jogador irá apostar na sua vez.

pote: montante de dinheiro apostado por todos os jogadores da rodada.

nome_jogador: nome do jogador

jogada: representa a mão que o jogador formou, como “um par”, “dois pares”.

carta: string que representa a carta, inicia com o valor e depois o naipe.

valor_mao: inteiro que representa o valor da jogada.

nome_vencedor: nome do jogador vencedor da rodada.

jogada_vencedor: maior jogada da rodada.

invalida: booleano que representa se uma rodada é invalida, é verdadeiro quando um jogador tenta apostar mais dinheiro do que possui.

jogadores: uma lista de objetos do tipo Jogador.

Métodos:

Partida(): inicia a partida de Poker

GetNrVencedores(Jogador *jogadores, int numero_jogadores): calcula quantos jogadores ganharam a rodada, é útil pois se houver mais de um vencedor, a lógica de apresentação dos resultados é diferente.

GetVencedor(Jogador *jogadores, int numero_jogadores): retorna o indice do jogador vencedor da rodada.

GetVencedores(Jogador *jogadores, int numero_vencedores): retorna uma lista de inteiros com os indices dos jogadores que empataram vencendo a rodada.

2.3 Função Main

A função Main cria um objeto da classe Mesa, onde ocorre o jogo.

3. Análise de Complexidade:

Para a análise de complexidade, é necessário observar duas variáveis; o número de rodadas e o número de jogadores, pois o número de cartas é constante e igual a 5.

Aqui, vamos considerar:

- r: número de rodadas
- j: número de jogadores
- v: número de vencedores

Jogador::AdicionarCarta(): Essa função analisa se mao do jogador esta vazia e adiciona uma carta, nas próximas chamadas, a função irá analisar se na posição possui uma carta e caso sim, pula para a próxima posição. Além disso, quando a mão já estiver cheia, a função realiza a ordenação utilizando o método da bolha. Considerando que o número máximo de jogadores por rodada seja 10 devido a ser um único baralho, essa função possui pior caso **$O(j^2)$**.

Jogador::GetMao(): Aqui o máximo de loops é 1, onde se analisa o número de repetições do valor, logo: **O(j).**

Mesa::GetNrVencedores(): Possui um loop que passa pelo número de jogadores e compara se há um ou mais vencedores: **O(j).**

Mesa::GetVencedor(): Percorre o número de jogadores e retorna o indice do jogador vencedor: **O(j).**

Mesa::GetVencedores(): Percorre o número de vencedores e retorna uma lista de indices dos jogadores vencedores: **O(v).**

Mesa::Partida(): A partida possui um loop com o número de rodadas, dentro dele há um loop para percorrer cada jogadores e dentro de cada jogador, são percorridas as cartas, logo a complexidade é a junção de **O(r)**, **O(j)** e **O(1)**, com isso: **O(r*j).**

4. Estratégias de Robustez:

Caso o arquivo de entrada não exista ou seja inválido, o programa gera uma exceção para o usuário.

Caso o jogador aposte sem que tenha a quantia necessária, por exemplo, o jogador tenha 50 e o pingo é 100, ou se ele apostar 100, a rodada é inválida, retornando ao usuário “0 0 I”, em que I indica que é uma rodada inválida.

5. Conclusões:

Neste trabalho foi criado um jogo de Poker em C++ utilizando estruturas de dados (Lista) e métodos de ordenação (Bubble Sort). Além disso, foi realizada uma análise dos custos de desempenho e utilização de memória e tempo de execução.

Foi possível observar que algumas funções possuem um custo muito mais elevado do que outras, como por exemplo a função de adicionar cartas à mão do jogador, pois possuem estruturas de repetição aninhadas, pois além de percorrer a mão do jogador, precisa ordená-la quando ela está completa.

Dentre os aprendizados, podem ser citados a manipulação de arquivos de texto tanto para escrita quanto para leitura, os conceitos de programação orientada a objetos em geral, como a abstração das classes que devem ser utilizadas, os atributos e métodos, os métodos construtores, *Setters* e *Getters* e o conceito de um objeto possuir outros objetos, como a mão do jogador ser uma lista de cartas.

6. Bibliografia:

GEEKS FOR GEEKS. **Bubble Sort**. Disponível em: www.geeksforgeeks.org/bubble-sort . Acesso em 28 mai. 2022.

TUTORIALS POINT. **C++ Files and Streams**. Disponível em: www.tutorialspoint.com/cplusplus/cpp_files_streams.htm . Acesso em 28 mai. 2022.

AGOSTINHO BRITO. **Programação Orientada a Objetos em C++**. Disponível em: agostinhobritojr.github.io/tutorial/cpp/ . Acesso em 28 mai. 2022

7. Instruções para compilação e execução:

- Acesse o diretório TP
- A partir de um terminal Linux, compile os arquivos do TP utilizando o comando *make*.
- Será gerado na pasta ‘obj’ os object files
- Será gerado na pasta ‘bin’ o arquivo run.out, que é o executável do projeto.
- Execute o arquivo run.out com o comando bin/run.out <arquivo_entrada.txt>
- Será gerado um arquivo saida.txt