Avaliação de dispensa LP1

Instruções Gerais

Os problemas a seguir devem ser resolvidos usando C++, qualquer dúvida que você tenha, fique à vontade para contatar a comissão por email: isaacfranco@imd.ufrn.br, ramon.fontes@imd.ufrn.br e julio.melo@imd.ufrn.br.

Para os programas que seguem, SEMPRE que houver a necessidade de escrever classes, divida o programa em .h e .cpp. Para cada programa deverá haver um .h e um .cpp(quando necessário) para cada classe, bem como um .cpp para o arquivo principal (main.cpp).

A nota desta avaliação é composta por uma avaliação do código entregue e da desenvoltura do aluno em uma entrevista, em termos do domínio dos conceitos envolvidos e da qualidade nas respostas das indagações do entrevistador. O aluno que obtiver nota 7 ou superior será considerado aprovado e portanto, dispensado da disciplina.

Para que você seja entrevistado é necessária a entrega de ao menos duas das três questões listadas abaixo. **Você não precisa entregar as 3 questões para ser aprovado na proficiência**, no entanto, fazer as 3 te concede alguns pontos adicionais. A entrevista será feita via google meets em horário previamente marcado pelo professor responsável.

Questão 1

Escreva um programa que simule o jogo da forca. O seu jogo deve contemplar as seguintes premissas:

- Um arquivo de entrada "words.txt", que possua pelo menos 50 palavras escolhidas por você, e outro arquivo “scores.txt” que deve ser usado para guardar os scores dos jogadores. Esses arquivos devem ser fornecidos via entrada do programa usando os argumentos passados para o main(argc e argv).
- Inicialmente o programa deverá exibir um menu simples perguntando se o usuário quer iniciar o jogo ou se quer ver os scores anteriores.
    - Caso o usuário escolha iniciar o jogo: Uma palavra do arquivo de entrada é escolhida aleatoriamente pelo programa e vários espaços em branco separados por underline devem ser gerados.
    - Exemplo: Considerando que a palavra aleatória seja CARRO. Logo, deverá ser impresso na tela:

```
Pontos: 0




 _ _ _ _ _
```

    - O jogador irá tentar adivinhar cada letra da palavra aleatória. Cada palpite correto equivale a 1 ponto, por letra na palavra, e cada palpite errado perde 1 ponto. Quando o jogador acerta a última letra da palavra ele ganha 2 pontos adicionais. Acertar todas as letras da palavra sem qualquer erro dá 5 pontos adicionais.
    - O jogador terá 6 tentativas, uma parte da figura do jogo da forca é exibida para cada palpite errado, se o palpite for correto as letras da palavra são exibidas. Exemplo: primeiro a cabeça, depois o tronco, braços e pernas. Quando a forca completa for exibida, significa que o jogo acabou. A palavra que foi sorteada deve ser mostrada. Exemplo:

```
Pontos: -4
 o
/|\
/ \
_ A R R _
Você perdeu a palavra era CARRO!
```

    - Ao acertar uma palavra o jogador deve ser perguntado se quer continuar jogando ou parar. Caso ele escolha continuar jogando, uma palavra **diferente** deve ser sorteada, os pontos são mantidos e o jogo recomeça.
    - Caso o jogador escolha parar ou o jogo tenha terminado devido a um *game over*, o programa deve pedir o nome do jogador. O programa deverá então guardar todas as palavras acertadas por aquele jogador junto com seu score final no arquivo “score.txt” que foi recebido no inicio do programa.

```
Julio; CARRO, CART, CORAÇAO; 10
Isaac; -4
Ramon; PROFESSOR, CASA, DISCORD, TESTE, CONCORDO; 60
```

- O arquivo score.txt deverá ser usado para listar os scores dos jogadores caso o usuário selecione, no início do jogo, para visualizar os scores anteriores.
    - Na hora de exibir os scores anteriores crie um formato de tabela separando, nome, palavras e pontuação final no seguinte formato:

```
NOME  | PALAVRAS                  | PONTOS
Julio | CARRO, CART, CORAÇAO      | 10
Isaac |                           | -4
Ramon | PROFESSOR, CASA, DISCORD  | 60
      | TESTE, CONCORDO           |
```

    - Veja que as palavras foram alinhadas, neste caso tente ser mais estético do que funcional alinhando as colunas de NOME, PALAVRAS e PONTOS. A quantidade máxima de palavras por linha deve ser 3, caso um jogador tenha acertado 4 palavras ou mais quebre a linha e continue como no formato.

Use **ao menos três classes** no seu programa: Uma classe para modelar o jogo e as interações com o usuário, uma para modelar o banco de dados de palavras e outra para modelar o score. **Não use atributos públicos e nem faça *gets* e *sets* em containers** (caso queira operar em containers que são atributos das classes faça elemento a elemento usando métodos da classe)!

Questão 2

Escreva um programa que representa um cadastro de usuários sem repetição de CPF. Seu programa deve:

1) Ter uma classe chamada Usuario com o nome (string) e um número cpf (int) como atributos.
2) O programa deve ter uma **coleção** chamada "cadastro", que armazena ***referencias*** de instâncias da classe Usuário. Para desambiguar o termo, a ideia é que você **use alocação dinâmica** para instanciar a classe usuário.
3) No início do programa programa ele deverá, repetidamente:
    a) Solicitar o nome e o cpf do usuário a se cadastrar
    b) Verificar se já existe algum usuário com esse **CPF** no cadastro
        i) Caso não exista, cria um novo usuário e adiciona no cadastro
        ii) Caso exista exibir uma mensagem que o usuário já foi cadastrado
    c) O programa deve ter uma forma de encerrar o processo de cadastro, por exemplo: você pode perguntar ao usuário se ele quer continuar ou não a cada laço.
4) Ao finalizar a operação de cadastro, exiba no programa outro menu contendo as seguintes opções:
    a) Buscar no cadastro por CPF: pede ao usuário o valor do cpf, realiza a busca e exibe o Nome e CPF da pessoa buscada; O programa deve exibir uma mensagem de erro caso o cpf não tenha sido cadastrado no sistema.
    b) Buscar no cadastro por nome: pede ao usuário o valor do nome, realiza a busca e exibe o Nome e CPF da pessoa buscada; O programa deve exibir uma mensagem de erro caso o cpf não tenha sido cadastrado no sistema.
    c) Modificar um cadastro por CPF: pede ao usuário o valor do cpf e depois pede um novo valor de Nome e de CPF; o sistema deverá atualizar no cadastro os novos valores de nome e cpf para o CPF pesquisado; O programa deve exibir uma mensagem de erro caso o cpf não tenha sido encontrado.
    d) Remover um cadastro por CPF: pede ao usuário o valor do cpf, realiza a busca e remove a pessoa com o cpf cadastrado; O programa deve exibir uma mensagem de erro caso o cpf não tenha sido cadastrado no sistema.
    e) Listar todas as pessoas cadastradas. O sistema deve listar todas as pessoas cadastradas no formato: CPF - Nome
5) O programa não deve ter vazamentos de memória (advindos do uso de alocação dinâmica de forma equivocada).

Questão 3

Seja o arquivo maze.dat um arquivo de texto qualquer que descreve um labirinto, de acordo com as seguintes informações:

1) Na primeira linha um conjunto com 2 números decimais determina a quantidade linhas e de colunas que o labirinto a ser descrito irá conter.
2) A partir da segunda linha o labirinto é descrito por um conjunto de caracteres
    a) “#” define uma parede do labirinto - não é possível atravessar paredes
    b) “ “ define um espaço vazio que pode ser usado
    c) “P” define a posição inicial no labirinto que deve ser levada em consideração
    d) “E” define a saída do labirinto que deve ser buscada - este é o objetivo final do jogo

A Figura 1 traz um exemplo de arquivo maze.dat válido.:

```
15 31
################################
#   P                          #
# ########################## #
#                          # #
# ######################## # #
# #                      # # #
# # ################### # # # #
# # #     E           # # # #
# # #  ################ # # #
# # #                   # # #
# # ###################### # #
# #                        # #
# ########################## #
#                            #
################################
```

Figura1. Exemplo de arquivo contendo um labirinto 15x31

Desenvolva um programa que receba o arquivo maze.dat através do argumento da linha de comando (exemplo: ./meuprograma maze.dat) e funcione da forma como descrita abaixo: que contemple as informações descritas abaixo:

1) Inicialmente o programa deverá validar o arquivo, de acordo com as restrições descritas abaixo:
    a) Podem ter apenas um P e um E
    b) Não podem ter outros caracteres que não os descritos anteriormente.
    c) A quantidade de linhas e colunas sinalizadas não pode ser menor ou igual a 0.
    d) A quantidade de linhas e colunas devem corresponder ao labirinto descrito no arquivo texto

    Caso qualquer uma destas restrições seja violada o programa deve terminar apontando qual restrição não foi respeitada.

2) Depois da validação o programa deverá ler o arquivo e gerar para gerar o seguinte resultado em um arquivo solution.dat:
    a) Os espaços vazios do arquivo maze.dat devem ser preenchidos a partir do ponto de partida P até o objetivo final E e;
    b) Caso exista uma solução de P até E, sinalize a solução no labirinto usando os caracteres “>, <, ^, v” para indicar qual direção deve ser tomada a partir da posição inicial P e das demais posições recorrentes.
    c) Caso não exista solução, o arquivo solution.dat deve exibir, no mesmo formato que o tópico b, os caminhos testados, e depois da última linha que descreve o labirinto, deverá ser impressa a seguinte mensagem: “Não foram encontradas soluções”;

Seguem dois exemplos de saída possíveis:

```
15 31
################################
#   P>>>>>>>>>>>>>>>>>>>>>>>>v#
# ############################v#
#>>>>>>>>>>>>>>>>>>>>>>>>>>>v#v#
#^##########################v#v#
#^#>>>>>>>>>>>>>>>>>>>>v  #v#v#
#^#^##################v# #v#v#
#^#^#     E<<<<<<<<<<<<# #v#v#
#^#^#  ################## #v#v#
#^#^#                     #v#v#
#^#^########################v#v#
#^#^<<<<<<<<<<<<<<<<<<<<<<<#v#
#^############################v#
#^<<<<<<<<<<<<<<<<<<<<<<<<<<<#
################################
```

```
14 31
################################
#<<<P>>>>>>>>>>>>>>>>>>>>>>>>>#
################################
#                           # #
# ########################## # #
# #                       # # #
# # ################### # # # #
# # #     E             # # # #
# # #  ################## # # #
# # #                     # # #
# # ####################### # #
# #                         # #
# ########################### #
################################
Não foram encontradas soluções
```

Exemplos de arquivos de saída possíveis para o problema

Use **ao menos duas classes** no seu programa: Uma classe para modelar o labirinto e outra classe para guardar a solução. **Não use atributos públicos e nem faça *gets* e *sets* em containers** (caso queira operar em containers que são atributos das classes faça elemento a elemento usando métodos da classe)!