UNIVERSIDADE FEDERAL DO CEARÁ

UNIVERSIDADE FEDERAL DO CEARÁ - CAMPUS CRATEÚS

DISCIPLINA: FUNDAMENTOS DE PROGRAMAÇÃO (2017.1)

PROF. ARNALDO BARRETO VILA NOVA

TRABALHO FINAL DE FUNDAMENTOS DE PROGRAMAÇÃO

I. TEMAS
II. CRITÉRIOS DE AVALIAÇÃO
III. DATAS

I. TEMAS

1. Agenda de telefones. Cada pessoa deve ter os seguintes dados:
- Nome
- E-mail
- Endereço
- Telefone
- Observações (string para alguma observação especial)
- Data de Nascimento (struct com dia, mês e ano)

- Definir a estrutura acima. **(1.0)**
- Declarar um vetor, chamado *agenda*, com capacidade de agendar até 100 pessoas.
- Funções para:
    - Inserir pessoa. **(1.0)**
    - Remover pessoa. **(1.0)**
    - Alterar pessoa. **(1.0)**
    - Buscar um nome: imprime todos os dados das pessoas com esse nome. **(1.0)**
    - Buscar os aniversariantes por dia e mês de aniversário: imprime os nomes de todas as pessoas que fazem aniversário nesse dia e mês informados. **(1.0)**
    - Buscar os aniversariantes de um determinado mês: imprime os nomes de todos os aniversariantes do mês informado **(1.0)**
    - Imprimir os nomes e telefones de todas as pessoas da agenda **(0.5)**
    - Imprimir o nome e idade da pessoa mais velha da agenda e da pessoa mais nova. **(1.0)**
- O programa deve ter um menu principal oferecendo as opções acima em repetição com opção para encerrar. **(0.5)**
- Ao iniciar, o programa deve importar os dados de um arquivo externo (caso exista), e ao encerrar deve exportar os dados para este mesmo arquivo **(1.0)**.

2. Registro de funcionários de uma empresa. Cada funcionário deve ter os seguintes dados:

- Nome
- Idade
- Gênero (‘M’ ou ‘F’)
- CPF
- Data de Nascimento (struct com dia, mês e ano)
- Data de Contratação (struct com dia, mês e ano)
- Cargo que ocupa
- Salário

- Definir a estrutura acima. **(1.0)**
- Declarar um vetor, chamado *rh*, com capacidade de agregar até 100 funcionários.
- Funções para:
    - Inserir funcionário. **(1.0)**
    - Alterar funcionário. **(1.0)**
    - Remover funcionário. **(1.0)**
    - Buscar funcionário por nome, CPF ou cargo: imprime os dados dos funcionários com um dado nome, CPF ou cargo **(1.0)**
    - Buscar funcionário por intervalo de salários: imprime os nomes e CPFs de todos os funcionários que tenham salário entre dois valores digitados pelo usuário **(1.0)**
    - Imprimir o valor total de pagamento mensal de funcionários **(1.0)**
    - Imprimir os nomes e cargos de todos os funcionários **(0.5)**
    - Imprimir o funcionário com salário mais alto, o funcionário mais antigo na empresa e o funcionário mais velho. **(1.0)**
- O programa deve ter um menu principal oferecendo as opções acima em repetição com opção para encerrar. **(0.5)**
- Ao iniciar, o programa deve importar os dados de um arquivo externo (caso exista), e ao encerrar deve exportar os dados para este mesmo arquivo **(1.0)**.

3. Produtos de um supermercado. Cada produto deve ter os seguintes dados:

- Nome
- Preço
- Quantidade em estoque
- Código no sistema
- Fabricante
- Data de Vencimento (struct com dia, mês e ano)

- Definir a estrutura acima. **(1.0)**
- Declarar um vetor, chamado produtos, com capacidade de cadastrar até 100 produtos.
- Funções para:
    - Inserir produto. **(1.0)**
    - Alterar produto. **(1.0)**
    - Remover produto. **(1.0)**
    - Buscar por nome ou código: imprime os dados do produto com o nome ou código dado. **(1.0)**
    - Buscar fabricante: imprime os produtos de um fabricante **(1.0)**
    - Imprimir o valor total dos produtos do supermercado (quantidade em estoque * preço). **(1.0)**
    - Imprimir o nome e preço dos produtos do supermercado **(0.5)**
    - Buscar produtos com data de validade mais próxima: imprime o nome, a quantidade e o preço dos produtos que estão mais próximos de alcançar a data de validade. **(1.0)**
- O programa deve ter um menu principal oferecendo as opções acima em repetição com opção para encerrar. **(0.5)**
- Ao iniciar, o programa deve importar os dados de um arquivo externo (caso exista), e ao encerrar deve exportar os dados para este mesmo arquivo **(1.0)**

4. Pedidos em um restaurante. Cada pedido deve ter os seguintes dados:
- Nome do item
- Quantidade
- Observação (tipo string com alguma observação especial)
- Horário do pedido (struct com hora e minutos)
- Mesa do pedido

- Definir a estrutura acima. **(1.0)**
- Declarar um vetor, chamado pedido, com capacidade de cadastrar até 100 pedidos.
- Funções para:
    - Inserir pedido **(1.0)**
    - Alterar pedido **(1.0)**
    - Remover pedido **(1.0)**
    - Buscar por mesa: imprime todos os pedidos dessa mesa. **(1.0)**
    - Buscar por nome do item: imprime todos os pedidos deste item. **(1.0)**
    - Imprimir o pedido mais antigo: imprime os dados dos pedidos que estão esperando por mais tempo. **(1.0)**
    - Imprimir todos os pedidos **(0.5)**
    - Imprimir o pedido com maior quantidade. **(1.0)**
- O programa deve ter um menu principal oferecendo as opções acima em repetição com opção para encerrar. **(0.5)**
- Ao iniciar, o programa deve importar os dados de um arquivo externo (caso exista), e ao encerrar deve exportar os dados para este mesmo arquivo **(1.0)**

5. Livro de receitas. Cada receita deve ter os seguintes dados:
- Nome
- Ingredientes (tipo struct com nome, quantidade e preço)
- Descrição da receita
- Tempo de Preparo (struct com horas e minutos)

- Definir a estrutura acima. **(1.0)**
- Declarar um vetor com capacidade para até 100 receitas.
- Funções para:
    - Inserir receita **(1.0)**
    - Alterar receita **(1.0)**
    - Remover receita **(1.0)**
    - Buscar por nome: Imprime as receitas com esse nome. **(1.0)**
    - Buscar por ingrediente: imprimir o nome das receitas com este ingrediente e a quantidade que a receita usa. **(1.0)**
    - Imprimir o nome e ingredientes da receita com o maior valor total dos ingredientes (se tiver mais de uma receita com este mesmo valor, imprimir todas). **(1.0)**
    - Imprimir todos as receitas. **(0.5)**
    - Imprimir a receita com menor tempo de preparo e a receita com maior tempo de preparo. **(1.0)**
- O programa deve ter um menu principal oferecendo as opções acima em repetição com opção para encerrar. **(0.5)**
- Ao iniciar, o programa deve importar os dados de um arquivo externo (caso exista), e ao encerrar deve exportar os dados para este mesmo arquivo **(1.0)**

6. Registro de atletas olímpicos. Cada atleta deve ter os seguintes dados:
- Nome
- Data de Nascimento (tipo struct com dia, mês e ano)
- País
- Esporte
- Modalidades (até 5 modalidades)
- Medalhas (vetor com número de medalhas de ouro, prata e bronze)

- Definir a estrutura acima. **(1.0)**
- Declarar um vetor com capacidade de agregar até 100 atletas.
- Funções para:
    - Inserir atleta **(1.0)**
    - Alterar atleta **(1.0)**
    - Remover atleta **(1.0)**
    - Buscar por nome: imprime os dados do atleta com esse nome (se tiver mais de um atleta com o mesmo nome, imprime os dados de todos). **(1.0)**
    - Buscar por País: imprimir todos os atletas do país e quantas medalhas de ouro, prata e bronze cada atleta tem. **(1.0)**
    - Buscar por esporte: imprimir o nome de todos os atletas daquele esporte e a quantidade de medalhas de ouro, prata e bronze de cada atleta. **(1.0)**
    - Imprimir relatório de todos os países e total de medalhas de cada um. **(1.0)**
    - Imprimir os dados do atleta com maior total de medalhas **(0.5)**
- O programa deve ter um menu principal oferecendo as opções acima em repetição com opção para encerrar. **(0.5)**
- Ao iniciar, o programa deve importar os dados de um arquivo externo (caso exista), e ao encerrar deve exportar os dados para este mesmo arquivo **(1.0)**

7. Coleção de HQ. Cada HQ deve ter os seguintes dados:
- Título
- Volume
- Número
- Editora
- Desenhista
- Roteirista
- Observações (string com alguma observação especial)
- Data de lançamento (tipo struct com dia, mês e ano)

- Definir a estrutura acima. **(1.0)**
- Declarar um vetor com capacidade de agregar até 100 itens.
- Funções para:
    - Inserir HQ **(1.0)**
    - Alterar HQ **(1.0)**
    - Remover HQ **(1.0)**
    - Buscar por título e volume: imprime os dados da HQ com o título e volume informados (se tiver mais de uma HQ com o mesmo título e volume, imprime os dados de todas). **(1.0)**
    - Buscar por editora: imprimir os dados de todas HQs da editora informada. **(1.0)**
    - Buscar por roteirista ou desenhista: imprimir os dados de todas as HQ do roteirista ou desenhista informado. **(1.0)**
    - Imprimir relatório com o nome, volume e número de todas as revistas. **(0.5)**
    - Imprimir os dados da HQ mais antiga **(1.0)**
- O programa deve ter um menu principal oferecendo as opções acima em repetição com opção para encerrar. **(0.5)**
- Ao iniciar, o programa deve importar os dados de um arquivo externo (caso exista), e ao encerrar deve exportar os dados para este mesmo arquivo **(1.0)**

8. Jogo de Forca. O jogador deve acertar o maior número de palavras dizendo uma letra por vez, podendo errar no máximo 5 vezes. O jogo segue os seguintes passos:

1 – Início do jogo: uma palavra é sorteada dentre as palavras cadastradas

2 – Usuário informa uma letra: caso exista aquela letra na palavra sorteada o programa deve informar em quais posições da palavra tem aquela letra. Caso não exista a letra, é contabilizado um erro.

3 – O passo 2 é repetido até que: o jogador acerte a palavra, sorteando uma nova palavra, continuando o jogo; ou até que ele complete 6 erros, terminando o jogo e voltando para o menu inicial.

4 – Ao terminar o jogo, devem ser armazenados os dados do jogador com:

- Nome do jogador
- Quantidade de palavras que acertou
- Quantidade de letras palpitadas

- Definir a estrutura acima **(0.5)**;
- Declarar um vetor com capacidade de armazenar até 100 palavras e outro vetor para armazenar até 100 jogadores.
- Funções para:
    - Inserir palavra **(1.0)**
    - Alterar palavra **(1.0)**
    - Remover palavra **(1.0)**
    - Jogar. **(4.0)**
    - Buscar jogador com maior pontuação: imprimir os dados do jogador que acertou o maior número de palavras (se tiver mais de um jogador com a mesma pontuação, mostrar todos). **(1.0)**
- O programa deve ter um menu principal oferecendo as opções acima em repetição com opção para encerrar. **(0.5)**
- Ao iniciar, o programa deve importar os dados de arquivos externos (um para palavras e outro para os dados dos jogadores), e ao sair deve exportar todos os dados para estes mesmos arquivos **(1.0)**

9. Tic Tac Toe. O também chamado Jogo da Velha pode ser estendido para variações de qualquer tamanho de tabuleiro, para qualquer tamanho de sequências ou ainda qualquer número de jogadores. Aqui, o programa deve rodar um Tic Tac Toe para três jogadores que competem para formar uma sequência de 4 peças em um tabuleiro 5x5. O jogo deverá seguir os seguintes passos:

1 – Início do jogo: recebe nome dos jogadores. O tabuleiro está vazio.
2 – Jogador 1 informa uma posição vazia onde vai colocar sua peça (linha e coluna). O segundo jogador faz o mesmo e, logo depois, o terceiro também.
3 – A cada jogada deve ser realizada uma verificação de vitória daquele jogador, ou seja, deve ser verificado se a partir daquela posição que ele colocou sua peça existe uma sequência de peças dele em alguma direção.
4 – A partida termina quando algum jogador tem a vitória (contabilizando a vitória do jogador) ou quando não há mais posições livres para jogar. Neste último caso, a partida termina em empate.

Antes de iniciar uma partida, deve ser informado o nome dos três jogadores para contabilizar vitórias e empates. Então, o cadastro de jogadores deve ter os seguintes dados:

- Nome
- Número de vitórias (inicialmente igual a zero)
- Número de empates (inicialmente igual a zero)

- Definir a estrutura acima **(0.5)**;
- Declarar um vetor com capacidade de armazenar até 100 jogadores
- Funções para:
    - Inserir jogador **(1.0)**
    - Alterar jogador **(1.0)**
    - Remover jogador **(1.0)**
    - Jogar partida
        - Realizar jogadas **(1.0)**
        - Verificar vitória **(2.0)**
        - Contabilizar vitória ou empate no cadastro dos jogadores **(1.0)**
    - Buscar jogador com maior pontuação: imprimir os dados do jogador com maior número de vitórias (se tiver mais de um jogador com a mesma pontuação, mostrar todos). **(1.0)**
- O programa deve ter um menu principal oferecendo as opções acima em repetição com opção para encerrar. **(0.5)**
- Ao iniciar, o programa deve importar os dados dos jogadores de arquivo externo (caso exista), e ao sair deve exportar os dados para este mesmo arquivo **(1.0)**

II. CRITÉRIOS DE AVALIAÇÃO

- Peso total do trabalho: 10.0
- Peso do código (nota da equipe): 5.0
- Peso da apresentação (nota individual): 5.0

DO CÓDIGO (NOTA DO GRUPO)

O código será analisado de acordo com os pesos de cada etapa do programa definidos na descrição do tema. Além da funcionalidade de cada etapa, serão considerados a organização do código, comentários e lógica de como foi realizada a tarefa. Não será considerado quem fez qual parte do programa.

DA APRESENTAÇÃO (NOTA INDIVIDUAL)

A apresentação deve ser de 20 min, onde deverá ser mostrado o programa em funcionamento e um passo a passo de como o programa foi implementado. Todos os integrantes da equipe devem participar da apresentação e será avaliado o domínio sobre a lógica e as instruções do programa por parte de cada um, com possível arguição ao final da apresentação.

III. DATAS

- Entrega do código: até o dia **12 de Julho.**
- Apresentações: dia **13 de Julho** nos horários normais de aula
- Obs.: A entrega do código após o horário definido acarretará na perda de 3 pontos no peso da nota do código.