Segundo Exercício-Programa

Prof. Luciano Antonio Digiampietri

Prazo máximo para a entrega: 17/07/2022

1 Sistema de Gerenciamento de um Curso

Neste trabalho, você deverá desenvolver um sistema simplificado para o gerenciamento de um curso. De uma maneira resumida (que será detalhada ao longo deste enunciado), um curso é composto por turmas que tem seus ministrantes, os quais gerenciam um conjunto de alunos (matriculados na respectiva turma). Há apenas três características dos alunos a serem gerenciadas: a nota, a frequência e o status do aluno na respectiva turma. Um aluno pode ter aulas com mais de um professor (ministrante). Por simplificação, cada curso pode ter até 10 ministrantes e cada ministrante pode ter até 20 alunos em sua turma.

Detalhamento:

Nesta simplificação existem as seguintes estruturas principais, descritas a seguir: *PESSOA*, *MINISTRANTE*, *ALUNOMATRICULADO* e *CURSO*.

***PESSOA*:** é uma estrutura que possui dois campos (*nome* do tipo ponteiro para caracteres e *nusp* do tipo inteiro).

***ALUNOMATRICULADO*:** possui um campo chamado *pessoa* do tipo PESSOA e três campos adicionais: um campo do tipo char chamado de *status* cujo valor será 'M' para o aluno recém matriculado na turma e poderá ter outros cinco valores possíveis dependendo da nota e da frequência do aluno no final da disciplina; e dois campos do tipo inteiro: *nota* que armazena a nota do aluno (de 0 [zero] a 100) e *frequencia* que armazena a frequência do aluno (de 0 [zero] a 100).

***MINISTRANTE*:** possui um campo chamado *pessoa* do tipo PESSOA e dois campos adicionais: um arranjo de referências/ponteiros para ALUNOMATRICULADO, chamado *alunos* e um inteiro chamado *numAlunos* que indica quantos alunos há dentro do arranjo *alunos*.

***CURSO*:** possui dois campos: um arranjo de referências/ponteiros para MINISTRANTE, chamado *ministrantes* e um inteiro chamado *numMinistrantes* que indica quantos ministrantes o curso possui (estes ministrantes estarão no arranjo *ministrantes*).

O código fornecido para este EP já possui várias funções implementadas, conforme será detalhado adiante, e possui uma função ***main*** que, quando executada, cria um cenário de um curso com ministrantes e alunos e testa diversas funções envolvidos neste EP. Esta função não fará parte da avaliação do EP e serve apenas para te auxiliar nos testes de seu EP (ela não

necessariamente cobre todos os testes possíveis para este EP).

Já foram implementadas funções para criar/inicializar cada uma das "entidades" relacionadas ao sistema: *novoAluno*, *novoMinistrante* e *novoCurso*.

Há, também, duas funções já implementadas para exibir dados relacionados ao sistema, são elas: *void imprimirAlunos(MINISTRANTE min1)*, que exibe algumas informações dos alunos da turma do ministrante *min1* (ver código fornecido para mais informações); e *void imprimirDadosCurso(CURSO c1)*, que exibe algumas informações do curso *c1*, de seus ministrantes e dos alunos dessas turmas (ver código fornecido para mais informações).

As seguintes funções são relacionadas aos ministrantes (as duas primeiras já estão implementadas, a terceira deverá ser implementada/completada por você).

- *bool adicionarAluno(MINISTRANTE* min1, ALUNOMATRICULADO* aluno)*: função para adicionar um aluno (ou mais especificamente uma referência a um aluno) no arranjo de alunos do ministrante referenciado por *min1*. Caso o número de alunos seja igual a 20, não deve adicionar e deve retornar *false*. Caso contrário, há duas situações: 1ª: o aluno já consta no arranjo de referências a alunos (verifique isso usando o nusp), neste caso o aluno não deve ser reinserido e a função deve retornar *false*; 2ª: o aluno passado como parâmetro não consta no arranjo de alunos, neste caso o aluno deve ser adicionado na posição *numAlunos*, este campo (apontado por min1) deve ser incrementado em 1 e a função deve retornar *true*.

- *void darPontoParaATurma(MINISTRANTE min1)*: função para dar um ponto extra (10 pontos no esquema de notas adotado) para todos os alunos da turma ministrada por *min1*, desde que o aluno não esteja reprovado por faltas. Para cada um dos alunos referenciados no arranjo *alunos* do ministrante *min1*, esta função deve: não fazer nada, caso a frequência do aluno seja menor do que 70; caso contrário, deve chamar a função *ganharPontoExtra* passando, como parâmetros, o endereço do aluno atual e o valor 10.

- *void arredondarParaCinco(MINISTRANTE min1, int valorMinimo)*: função para arredondar a nota de todos os alunos presentes no arranjo de alunos do ministrante *min1*, de acordo com a seguinte regra: não fazer nada para o aluno, caso sua frequência seja menor do que 70. Caso contrário, há duas situações: 1ª: se a nota do aluno atual for maior ou igual a 50 ou menor do que valor mínimo (*valorMinimo*), não faça nada. 2ª: caso contrário, atualize a nota do aluno atual para 50.

Há quatro funções relacionadas aos alunos (que deverão ser implementadas/completadas por você):

- *bool preRequisitoFraco(ALUNOMATRICULADO* a1)*: função que retorna *true* caso, para o aluno referenciado por *a1*, o campo *nota* seja maior ou igual a 30 e o campo *frequencia* seja maior ou igual a 70. Caso contrário, deverá retornar *false*.

- *bool ganharPontoExtra(ALUNOMATRICULADO* a1, int valor)*: função para o aluno atual (referenciado por *a1*) obter ponto(s) extra(s) de acordo com o valor passado por parâmetro. Caso o valor do parâmetro seja menor ou igual a zero, a função deve retornar *false*. Caso contrário há duas situações: 1ª: se o valor do parâmetro mais o valor da nota for maior do que 100, a nota deve ser atualizada para 100 e a função deve retornar *true*. 2ª: caso contrário, a nota deverá ser atualizada para o valor atual da nota mais o valor passado como parâmetro e a função deve retornar *true*.

- *bool atualizarNotaEFrequencia(ALUNOMATRICULADO* a1, int nota, int freq)*: função para o aluno atual (referenciado por *a1*) ter sua nota e sua frequência atualizadas. Caso o valor de qualquer um dos parâmetros *nota* ou *freq* seja(m) negativo(s) ou maior(es) do que 100, a função deve retornar *false*. Caso contrário, os campos *nota* e *fequencia* do aluno referenciado por a1 devem ser atualizados de acordo com o valor dos parâmetros e a função deverá retornar *true*.

- *void atualizarStatus(ALUNOMATRICULADO* a1)*: função para o aluno atual (referenciado por *a1*) ter seu status na turma/disciplina atualizado de acordo com sua nota e sua frequência. Há cinco atualizações de status possíveis. 1ª: Caso a nota seja menor do que 50 e a frequência seja menor do que 70 o status deve ser atualizado para **E** (reprovado por nota **E** frequência); 2ª: Caso a nota seja menor do que 30 e a frequência maior ou igual do que 70, o status deve ser atualizado para **N** (reprovado por **N**ota); 3ª: Caso a nota seja maior ou igual a 50 e a frequência menor do que 70, o status deve ser atualizado para **F** (reprovado por **F**altas); 4ª: Caso a nota seja maior ou igual a 30 e menor do que 50 e a frequência seja maior ou igual a 70, o status deve ser atualizado para **R** (aluno em **R**ecuperação); 5ª: Caso a nota seja maior ou igual a 50 e a frequência seja maior ou igual a 70, o status deve ser atualizado para **A** (aluno **A**provado).

Além da função que exibe informações relacionadas a um curso, há outras duas funções relacionada ao curso, uma já implementada relacionada a atualização do status dos alunos. Já a outra deverá ser implementada por você. Note que esta função tem um comportamento muito parecido com outra função já implementada no EP, chamada *adicionarAluno*.

bool atualizarStatusDoCurso(CURSO c1): função para atualizar o status de todos os alunos do curso *c1*. Para cada um dos alunos das turmas de cada ministrante do curso *c1*, esta função deve invocar a função *atualizarStatus*, passando o endereço do aluno como parâmetro.

bool adicionarMinistrante(CURSO c1, MINISTRANTE* ministrante)*: função para adicionar um ministrante (uma referência a um ministrante) no arranjo de ministrantes do curso referenciado por *c1*. Caso o número de ministrantes seja igual a 10, não deve adicionar e deve retornar *false*. Caso contrário, há duas situações: 1ª: o ministrante já consta (já é referenciado) no arranjo de ministrantes (verifique isso usando o *nusp*), neste caso o ministrante não deve ser reinserido e a função deve retornar *false*; 2ª: o ministrante passado como parâmetro

não consta no arranjo de ministrantes: o ministrante deve ser adicionado na posição *numMinistrantes*, este campo deve ser incrementado em 1 e a função deve retornar *true*.

1.1 Material a Ser Entregue

Um arquivo, denominado *NUSP*.c (sendo NUSP o seu número USP, por exemplo: 123456789.c), contendo seu código, incluindo todas as funções solicitadas e qualquer outra função adicional que ache necessário. Para sua conveniência, `completeERenomeie.c` será fornecido, cabendo a você então completá-lo e renomeá-lo para a submissão.

Atenção!

1. Não modifique as assinaturas das funções já implementadas e/ou que você deverá completar!
2. Para avaliação, as diferentes funções serão invocadas diretamente (individualmente ou em conjunto com outras funções). Em especial, qualquer código dentro da função `main()` será ignorado.

2 Entrega

A entrega será feita única e exclusivamente via sistema e-Disciplinas, até a data final marcada. Deverá ser postado no sistema um arquivo .c, tendo como nome seu número USP:

```
seuNumeroUSP.c   (por exemplo, 12345678.c)
```

Não esqueça de preencher o cabeçalho constante do arquivo .c, com seu nome, número USP e turma etc.

A responsabilidade de postagem é exclusivamente sua. Por isso, submeta e certifique-se de que o arquivo submetido é o correto (fazendo seu download, por exemplo). Problemas referentes ao uso do sistema devem ser resolvidos com antecedência.

3 Avaliação

A nota atribuída ao EP terá como foco principal a funcionalidade solicitada, porém não esqueça de se atentar aos seguintes aspectos:

1. Documentação: se há comentários explicando o que se faz nos passos mais importantes e para que serve o programa (tanto as funções quanto o programa em que estão inseridas);
2. Apresentação visual: se o código está legível, identado etc;
3. Corretude: se o programa funciona.

Além disso, algumas observações pertinentes ao trabalho, que influem em sua nota, são:

- Este exercício-programa deve ser elaborado individualmente;

- Não será tolerado plágio, em hipótese alguma;
- Exercícios com erro de sintaxe (ou seja, erros de compilação), receberão nota ZERO.