Segundo Exercício-Programa

Prof. Luciano Antonio Digiampietri

Prazo máximo para a entrega: 09/07/2023

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 e por alunos. Os alunos são matriculadas nas turmas e têm o desempenho nessas turmas registrado. Cada aluno poderá estar matriculado em diversas turmas ao mesmo tempo, mas não em duas turmas de uma mesma disciplina.

Detalhamento:

Nesta simplificação existem as seguintes estruturas principais, descritas a seguir: *TURMA*, *DESEMPENHO*, *ALUNO* e *CURSO*.

***TURMA*:** tem por objetivo representar uma turma. É uma estrutura que possui cinco campos: *codigoDisciplina* do tipo inteiro que corresponde ao código numérico da disciplina, *nomeDisciplina* do tipo ponteiro para caracteres que aponta para o nome da disciplina; *totalVagas* do tipo inteiro que contém o número de vagas oferecidas na turma; *numAlunos* do tipo inteiro que contém o número atual de alunos na turma; e *nusps* do tipo ponteiro para inteiro que aponta para um arranjo que contém o número USP de cada um dos alunos da turma.

***DESEMPENHO*:** tem por objetivo representar o desempenho de um aluno em uma turma. Possui quatro campos: *turma* que é do tipo ponteiro para TURMA e aponta para a turma a que o desempenho se refere; *nota* do tipo inteiro que contém a nota do aluno na respectiva turma (de 0 [zero] a 100); *frequencia* do tipo inteiro que contém a frequência do aluno na respectiva turma (de 0 [zero] a 100); e *status* do tipo char cujo valor será 'M' para o aluno recém matriculado na turma e poderá ter outros dois valores possíveis dependendo da nota e da frequência do aluno ('A' para aprovado e 'R' para reprovado).

***ALUNO*:** tem por objetivo representar um aluno e seu histórico escolar. Possui cinco campos: *nome* do tipo ponteiro para caracteres que aponta para o nome do aluno; *nusp* do tipo inteiro que representa o número USP do aluno; *historico* do tipo arranjo de *DESEMPENHO* que corresponde ao histórico escolar do aluno, com suas matrículas e desempenho nas turmas; *turmasNoHistorico* do tipo inteiro que indica em quantas turmas o aluno já se matriculou; e *status* do tipo char cujo valor será 'M' para o aluno que ainda não conclui seu curso e poderá ter outros dois valores possíveis dependendo de seu desempenho no conjunto de turmas ('F'

para formado e 'J' para jubilado).

***CURSO*:** tem por objetivo representar um curso que é composto por turmas e alunos. Possui quatro campos: *alunos* do tipo arranjo de ponteiros para ALUNOs; *numAlunos* do tipo inteiro que armazena a quantidade de alunos no curso; *turma* do tipo arranjo de ponteiros para TURMAs; e *numTurmas* do tipo inteiro que armazena a quantidade de turmas no curso.

Adicionalmente, o EP possui **quatro constantes** utilizadas no Gerenciamento do Curso: *MAX_TURMAS* que define o número máximo de turmas que um aluno pode cursar; *DISCIPLINAS_NECESSARIAS* que define o número de turmas nas quais um aluno precisa ser aprovado para se formar; *MAX_ALUNOS_CURSO* correspondendo ao número máximo possível de alunos em um curso; *MAX_TURMAS_CURSO* que indica o número máximo de turmas de um curso.

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 turmas e alunos e testa diversas funções envolvidas 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*, *novaTurma*, *novoCurso* e *novoDesempenho*.

Há, também, cinco funções já implementadas para exibir dados relacionados ao sistema, são elas: *void imprimirDadosAluno(ALUNO* a1)*, que exibe as informações de um aluno específico, incluindo informações sobre as turmas presentes no histórico escolar; e *void imprimirDadosTurma(TURMA* t1)*, que exibe informações sobre uma dada turma, incluindo a lista de números USP de seus alunos; *void imprimirDadosCurso(CURSO c1)*, que exibe a lista de alunos e de turmas de um dado curso; *void imprimirDadosTodosAlunos(CURSO c1)*, que exibe de forma detalhada as informações de todos os alunos de uma dado curso; *void imprimirDadosTodasTurmas(CURSO c1)*, que exibe de forma detalhada as informações de todas as turmas de uma dado curso.

Há cinco funções que você deverá implementar/completar:

- *bool adicionarAlunoAoCurso(ALUNO* aluno, CURSO* curso)*: função para adicionar um aluno (ou mais especificamente uma referência a um aluno) no arranjo de alunos do curso cuja referência/endereço de memória foi passado como parâmetro (*curso*). A função deverá retornar *false* caso o valor do endereço do parâmetro *aluno* ou do parâmetro *curso* seja igual a *NULL*. Também deverá retornar *false* caso o número de alunos no curso já seja igual à *MAX_ALUNOS_CURSO* ou se o aluno referenciado por *aluno* já pertença ao arranjo de alunos do curso (verifique isso com base no número USP do aluno). Caso contrário, o endereço do aluno passado como parâmetro deve ser inserido no arranjo de alunos, na posição *numAlunos* e este campo (apontado por *curso*) deve ser incrementado em 1 e a função deve retornar *true*.
- *bool adicionarTurmaAoCurso(TURMA* turma, CURSO* curso)*: função para adicionar

uma turma (ou mais especificamente uma referência a uma turma) no arranjo de turmas do curso cuja referência/endereço de memória foi passado como parâmetro (*curso*). A função deverá retornar *false* caso o valor do parâmetro *turma* ou do parâmetro *curso* seja igual a *NULL*. Também deverá retornar *false* caso o número de turmas no curso já seja igual à *MAX_TURMAS_CURSO* ou se a turma referenciado por *turma* já pertença ao arranjo de turmas do curso (verifique isso com base no endereço de memória que consta na variável *turma*, pois a mesma turma não deve ser inserida duas vezes, mas **é possível haver duas turmas diferentes de uma mesma disciplina**). Caso contrário, a turma passada como parâmetro deve ser inserida no arranjo de turmas, na posição *numTurmas* e este campo (apontado por *curso*) deve ser incrementado em 1 e a função deve retornar *true*.

- *bool cadastrarNota(ALUNO* aluno, int codigoDisciplina, int nota, int freq)*: função para atualizar o desempenho do aluno em uma turma. Esta função deverá retornar *false* caso o endereço recebido como parâmetro (*aluno*) tenha valor *NULL* ou se o parâmetro *nota* ou o parâmetro *frequencia* tenha valor menor do que zero ou maior do que 100. Adicionalmente, a função também deverá retornar *false* se o aluno referenciado pelo parâmetro *aluno* não possuir um *DESEMPENHO* em seu histórico escolar (*historico*) para a disciplina com código igual ao valor do parâmetro *codigoDisciplina* e cujo *status* seja igual a 'M' (matriculado). Caso contrário, o respectivo *DESEMPENHO* dentro do histórico escolar do aluno dever ser atualizado com a respectiva nota e frequência (passadas como parâmetro) e o *status* do *DESEMPENHO* deve ser atualizado para 'A' em caso de aprovação (nota maior ou igual a 50 e frequência maior ou igual a 70) ou para 'R' (reprovado), caso contrário, e a função deverá retornar *true*.

- *void atualizarStatusAluno(ALUNO* aluno)*: função para atualizar o status do aluno em relação a sua situação geral no curso, isto é: **M**atriculado, **F**ormado ou **J**ubilado. Caso o número total de disciplinas nas quais o aluno foi aprovado seja maior ou igual a *DISCIPLINAS_NECESSARIAS*, então o campo *status* deve ser atualizado para 'F'. Se, por outro lado, o número máximo de turmas nas quais ele pode se matricular (*MAX_TURMAS*) menos o número de disciplinas nas quais ele foi reprovado for menor do que o número de disciplinas necessárias para se formar (*DISCIPLINAS_NECESSARIAS*) o campo *status* deve ser atualizado para 'J', pois não será mais possível se formar. Por fim, caso nenhuma dessas situações ocorra, o status deve permanecer 'M' (matriculado).

- *bool matricularAlunoEmTurma(ALUNO* aluno, TURMA* turma)*: função para matricular um aluno em uma turma. A função deverá retornar *false* caso o endereço *aluno* ou *turma* seja igual a *NULL*. Caso contrário, a função deverá chamar a função *atualizarStatusAluno*, passando *aluno* como parâmetro, isto será necessário para verificar o *status* atualizado do aluno. A função também deverá retornar *false* se o número de alunos na turma (*numAlunos*) referenciada por *turma* for igual ao número total de vagas (*totalVagas*) dessa turma ou se o aluno atual tiver um *status* geral diferente de 'M' (ou seja, se já tiver se formado ou jubilado) ou se o número de turmas no histórico do aluno (*turmas-*

NoHistorico) for igual a *MAX_TURMAS*. A função também deverá retornar *false* caso o aluno já estiver matriculado nessa turma (isto deve ser verificado na lista de números USP da respectiva turma) ou estiver matriculado em outra turma da mesma disciplina ou já tenha sido aprovado nessa disciplina (é necessário verificar se ele tem um *DESEMPENHO* em seu histórico escolar nessa disciplina com status 'M' ou 'A', se isso ocorrer a função deverá retornar *false*)[1]. Caso contrário, um novo *DESEMPENHO* deverá ser inserido (já existe uma função implementada no código que cria e retorna um novo *DESEMPENHO*) no histórico escolar do aluno (*historico*), na posição *turmasNoHistorico* e o valor do campo *turmasNoHistorico* deverá ser incrementado em um; o número USP do aluno deverá ser inserido no arranjo apontado por *nusps* da respectiva turma na posição *numAlunos* e o campo *numAlunos* da turma deve ser incrementado em um; por fim, a função deverá 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.

[1] Note que é possível a um aluno se matricular em uma nova turma de uma disciplina na qual ele já foi reprovado (desde que já não esteja matriculado ou já tenha sido aprovado nessa disciplina).

3 Avaliação

A nota atribuída ao EP será focada nas funcionalidades solicitadas, 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 a função quanto o programa em que está inserida);
2. Apresentação visual: se o código está legível, identado etc;
3. Corretude: se o programa funciona.

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

- Este exercício-programa deve ser elaborado individualmente;
- Não será tolerado plágio;
- Exercícios com erro de sintaxe (ou seja, erros de compilação), receberão nota ZERO.