DCC/ICEx/UFMG — Gleison Mendonça e Luigi Soares
DCC 203 - Programação e Desenv. de Software I — **Prova Substitutiva - Turmas TA1, TA2, TM3**
2º semestre de 2022 — 15/12/2022

Estudante:	**Matrícula:**

Instruções:

i. Este exame possui 4 página(s). Verifique se sua cópia do exame está completa.

ii. **Esta prova é sem consulta**: você não tem permissão para consultar o livro texto, suas notas de aula, a Internet, seus colegas ou quaisquer outras pessoas ou fontes para concluir o exame.

iii. Se você acredita que alguma pergunta esteja subespecificada, anote as suposições que você teve que fazer para chegar a sua resposta e justifique-as como parte de sua resposta à pergunta.

iv. **Lembre-se de indentar o código** de maneira apropriada e atente-se a organização, clareza e legibilidade do código!

Utilize os espaços a seguir, referentes aos arquivo de cabeçalho `curso.h` e `aluno.h`, para a definição das estruturas solicitadas na primeira questão (se atente aos nomes dos campos!).

curso.h

```c
#ifndef CURSO_H_
#define CURSO_H_
typedef struct Curso {

  unsigned int id;
  char *nome;
  char departamento[4]; // Sigla de 3 caracteres + \0 = tamanho 4

} Curso;
#endif // CURSO_H_
```

aluno.h

```c
#ifndef ALUNO_H_
#define ALUNO_H_

#include <stdbool.h>
#include "curso.h"

typedef struct Aluno {

  char matricula[7]; // 6 digitos + \0
  char *nome;
  Curso *curso;

} Aluno;

typedef struct VetorAlunos {

  Aluno *alunos;
  unsigned int capacidade;
  unsigned int n;

} VetorAlunos;

bool salvarAlunos(VetorAlunos v, char *nome_arquivo);
#endif // ALUNO_H_
```

1. (4 pontos) Complete as estruturas `Curso`, `Aluno` e `VetorAlunos`, presentes no arquivos de cabeçalho `curso.h` e `aluno.h`, de acordo com as seguintes especificações (dica: lembre-se de considerar o \0 nas strings):

   (a) (1 ponto) Curso: `id` (inteiro não negativo), `nome` (string, tamanho variável, pode conter espaços) e `departamento` (string, sigla de 3 caracteres)

   (b) (1,5 pontos) Aluno: `matricula` (string, 6 dígitos), `nome` (string, tamanho variável, pode conter espaços) e `curso` (ponteiro para a estrutura Curso)

   (c) (1,5 pontos) VetorAlunos: `alunos` (arranjo de Alunos, tamanho variável), `capacidade` (inteiro não negativo) e `n` (inteiro não negativo)

2. (a) (3 pontos) Complete a função `f` a seguir.

aluno.c

```
1  VetorAlunos f(VetorAlunos v, Curso curso) {
2    VetorAlunos r;
3
4    r.capacidade = 10;
5    r.n = 0;
6
7    // Aloque espaco de acordo com a capacidade inicial:
8    r.alunos = (Aluno *)malloc(r.capacidade * sizeof(Aluno));
9
10   for (int i = 0; i < v.n; i++) {
11     if (r.n >= r.capacidade) {
12       // Dobre a capacidade:
13       r.capacidade *= 2;
14
15       // Realoque o espaco para comportar novos alunos:
16       r.alunos = (Aluno *)realloc(r.alunos, r.capacidade * sizeof(Aluno);
17     }
18
19     if (v.alunos[i].curso->id == curso.id) {
20       r.alunos[r.n] = v.alunos[i];
21       r.n++;
22     }
23   }
24
25   return r;
26 }
```

   (b) (6 pontos) Descreva o que faz a função `f` apresentada acima. Como você renomearia a função `f`? Mostre a execução e o resultado da função, caso ela tenha sido chamada com o segundo parâmetro sendo o curso de Ciência da Computação (ID 1) e o primeiro parâmetro sendo um vetor com os seguintes alunos:

   i. Matrícula: 111111, Nome: Chico, Curso: Ciência da Computação (DCC, ID 1)
   ii. Matrícula: 222222, Nome: Dara, Curso: Enfermagem (ENA, ID 2)
   iii. Matrícula: 333333, Nome: Mário, Curso: Ciência da Computação (DCC, ID 1)

   Resposta: A função `f` percorre os alunos do vetor, filtrando-os de acordo com o ID do curso passado como parâmetro e produzindo um novo vetor apenas com os alunos deste curso específico. Portanto, nomes adequados para a função seriam nomes como `filtrarAlunos` ou `filtrarPorCurso`. A função aloca um espaço inicial suficiente para armazenar 10 alunos e, quando o espaço previamente armazenado se esgota, ele é dobrado. Note que a quantidade de alunos, controlada pelo campo `n` da estrutura `VetorAlunos`, é sempre menor ou igual a `capacidade` do vetor de alunos. Assumindo a entrada descrita acima, o resultado seria um vetor contendo somente os dois alunos do curso de Ciência da Computação (CC). Na primeira iteração, será comparado o ID do curso do primeiro aluno, o Chico, com o ID do curso de CC e este aluno será inserido no vetor. Na segunda iteração, o ID do curso é diferente do ID do curso de CC; portanto, a aluna Dara não será adicionada ao vetor. Por fim, o terceiro aluno, o Mário, será adicionado ao vetor.

   (c) (6 pontos) Implemente a função `f` acima de forma **RECURSIVA**. Isto é, sua função **NÃO pode conter laços de repetição (`for`, `while`, `do-while`)**. Sua função deve seguir o mesmo protótipo da função `f`: o

retorno deve ser uma estrutura `VetorAlunos`, e deve possuir dois parâmetros dos tipos: `VetorAlunos` e `Curso`. Se desejar, você pode criar funções auxiliares, mas estas também **NÃO podem conter laços de repetição**.

Resposta com uso de funções auxiliares:

aluno.c

```c
void filtrarPorCursoAux(VetorAlunos v, Curso curso, int i, VetorAlunos *r) {
  if (i >= v.n) return; // Caso base, percorremos todos os alunos

  if (r->n >= r->capacidade) {
    r->capacidade *= 2;
    r->alunos = (Aluno *) realloc(r->alunos, r->capacidade * sizeof(Aluno));
  }

  // Se o aluno esta cursando o curso passado por parametro,
  // adicione-o ao vetor de alunos e incremente o campo
  // 'n' para indicar que o numero de alunos aumentou.
  if (v.alunos[i].curso->id == curso.id) {
    r->alunos[r->n] = v.alunos[i];
    r->n++;
  }

  // Caso recursivo:
  filtrarPorCursoAux(v, curso, i + 1, r);
}

VetorAlunos filtrarPorCurso(VetorAlunos v, Curso curso) {
  VetorAlunos r;

  r.n = 0;
  r.capacidade = 10;
  r.alunos = (Aluno *) malloc(r.capacidade * sizeof(Aluno));

  filtrarPorCursoAux(v, curso, 0, &r);
  return r;
}
```

Resposta com apenas uma função:

aluno.c

```c
VetorAlunos filtrarPorCurso(VetorAlunos v, Curso curso) {
  // Caso base: nao ha nenhum aluno no vetor. Nesse caso, apenas alocamos um
  // vetor com capacidade 10, mas sem nenhum aluno armazenado de fato.
  if (v.n == 0) {
    VetorAlunos r;

    r.n = 0;
    r.capacidade = 10;
    r.alunos = (Aluno *) malloc(r.capacidade * sizeof(Aluno));

    return r;
  }

  // Decrementa v.n para verificar o subarranjo que vai ate a posicao anterior.
  // Note que atualizar o valor do campo n nao influencia no valor do campo
  // fora da funcao, ja que VetorAlunos foi passado por valor e nao por
  // referencia. Outro ponto importante: perceba que a chamada recursiva
  // acontece primeiro. Logo, primeiro fazemos todas as chamadas e so depois
  // voltamos "compondo" os resultados de cada chamada.
  v.n--;
  VetorAlunos r = filtrarPorCurso(v, curso);
```

```
22
23    // v.n foi decrementado, logo agora corresponde ao indice do ultimo aluno.
24    if (v.alunos[v.n].curso->id != curso.id) {
25      return r;
26    }
27
28    if (r.n >= r.capacidade) {
29      r.capacidade *= 2;
30      r.alunos = (Aluno *) realloc(r.alunos, r.capacidade * sizeof(Aluno);
31    }
32
33    r.alunos[r.n] = v.alunos[v.n];
34    r.n++;
35
36    return r;
37  }
```

3. (6 pontos) Implemente uma função `salvarAlunos`, que recebe como parâmetro um vetor de alunos e o nome do arquivo (verifique o arquivo de cabeçalho `aluno.h`). Você deve criar o arquivo caso ele não exista ou sobrescrever o conteúdo do arquivo se ele já existir. Você deve salvar a matrícula e o nome de cada aluno, além do ID do curso (um aluno por linha). Você deve retornar falso caso não tenha sido possível abrir o arquivo ou verdadeiro caso contrário. O arquivo final deve obedecer o seguinte formato:

```
1  Matricula,Nome,Curso
2  111111,Chico,1
3  222222,Dara,2
```

Resposta:

aluno.c

```
1  bool salvarAlunos(VetorAlunos v, char *nome_arquivo) {
2    FILE *fp = fopen(nome_arquivo, "w");
3    if (fp == NULL) return false;
4
5    fprintf(fp, "Matricula,Nome,Curso\n");
6    for (int i = 0; i < v.n; i++) {
7      fprintf(fp, "%s,%s,%d\n", v.alunos[i].matricula, v.alunos[i].nome,
8              v.alunos[i].curso->id);
9    }
10
11   fclose(fp);
12   return true;
13 }
```