Departamento de Informática – PUC-Rio
INF1620 - Estruturas de Dados

Terceira Lista de Exercícios – 2005.1

1. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   char matricula[8];
   char turma;
   float p1;
   float p2;
   float p3;
   struct lista* prox;
};
typedef struct lista Lista;
```

   para representar o cadastro de alunos de uma disciplina, implemente uma função que insira um novo nó em uma lista encadeada definida pela estrutura acima. O novo nó deve ser inserido na lista de tal forma que os nós da lista encadeada estejam sempre em ordem alfabética do nome do aluno. Essa função deve obedecer o protótipo:

```
Lista* insere_ord(Lista* l, char* nome, char* matricula,
    char turma, float p1, float p2, float p3);
```

   Obs.: Essa função retorna a lista alterada.
   Dica: Utilize a função `strcmp`, definida no arquivo de cabeçalhos `string.h`, para comparar duas cadeias de caracteres. O cabeçalho e o modo de uso de `strcmp` são descritos a seguir:

```
int strcmp (char* s1, char* s2);
```

   - `strcmp(s1, s2) == 0` se as cadeias s1 e s2 são iguais
   - `strcmp(s1, s2) > 0` se a cadeia s1 é maior (alfabeticamente) que s2
   - `strcmp(s1, s2) < 0` se a cadeia s1 é menor (alfabeticamente) que s2

2. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   char matricula[8];
   char turma;
   float p1;
   float p2;
   float p3;
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar o cadastro de alunos de uma disciplina, implemente uma função que insira um novo nó em uma lista encadeada definida pela estrutura acima. O novo nó deve ser inserido na lista de tal forma que os nós da lista encadeada estejam sempre em ordem crescente de média (onde a média do aluno é calcula pela fórmula $\frac{p_1+p_2+p_3}{3}$). Essa função deve obedecer o protótipo:

```
Lista* insere_ord(Lista* l, char* nome, char* matricula,
    char turma, float p1, float p2, float p3);
```

Obs.: Essa função retorna a lista alterada.

3. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   int matricula;
   char departamento[21];
   float salario;
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar o cadastro de funcionários de uma empresa, implemente uma função que insira um novo nó em uma lista encadeada definida pela estrutura acima. O novo nó deve ser inserido na lista de tal forma que os nós da lista encadeada estejam sempre em ordem alfabética (crescente). Essa função deve obedecer o protótipo:

```
Lista* insere_ord(Lista* l, char* nome, int matricula,
    char* departamento, float salario);
```

Obs.: Essa função retorna a lista alterada.
Dica: Utilize a função `strcmp`, definida no arquivo de cabeçalhos `string.h`, para comparar duas cadeias de caracteres. O cabeçalho e o modo de uso de `strcmp` são descritos a seguir:

```
int strcmp (char* s1, char* s2);
```

- `strcmp(s1, s2) == 0` se as cadeias s1 e s2 são iguais
- `strcmp(s1, s2) > 0` se a cadeia s1 é maior (alfabeticamente) que s2
- `strcmp(s1, s2) < 0` se a cadeia s1 é menor (alfabeticamente) que s2

4. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   int matricula;
   char departamento[21];
   float salario;
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar o cadastro de funcionários de uma empresa, implemente uma função que insira um novo nó em uma lista encadeada definida pela estrutura acima. O novo nó deve ser inserido na lista de tal forma que os nós da lista encadeada estejam sempre em ordem **decrescente** de salário. Essa função deve obedecer o protótipo:

```
Lista* insere_ord(Lista* l, char* nome, int matricula,
   char* departamento, float salario);
```

Obs.: Essa função retorna a lista alterada.

5. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   char matricula[8];
   char turma;
   float p1;
   float p2;
   float p3;
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar o cadastro de alunos de uma disciplina, implemente uma função que retire um nó de uma lista encadeada definida pela estrutura acima. Essa função deve receber a matrícula do aluno a ser retirado do cadastro e retorna a lista sem o aluno. Essa função deve obedecer o protótipo:

```
Lista* retira(Lista* l, char* matricula);
```

Dica: Utilize a função `strcmp`, definida no arquivo de cabeçalhos `string.h`, para comparar duas cadeias de caracteres. O cabeçalho e o modo de uso de `strcmp` são descritos a seguir:

```
int strcmp (char* s1, char* s2);
```

- `strcmp(s1, s2) == 0` se as cadeias s1 e s2 são iguais
- `strcmp(s1, s2) > 0` se a cadeia s1 é maior (alfabeticamente) que s2
- `strcmp(s1, s2) < 0` se a cadeia s1 é menor (alfabeticamente) que s2

6. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   int matricula;
   char departamento[21];
   float salario;
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar o cadastro de funcionários de uma empresa, implemente uma função que retire um nó de uma lista encadeada definida pela estrutura acima. Essa função deve receber o nome do funcionário a ser retirado do cadastro e retorna a lista sem o funcionário. Essa função deve obedecer o protótipo:

```
Lista* retira(Lista* l, char* nome);
```

Dica: Utilize a função `strcmp`, definida no arquivo de cabeçalhos `string.h`, para comparar duas cadeias de caracteres. O cabeçalho e o modo de uso de `strcmp` são descritos a seguir:

```
int strcmp (char* s1, char* s2);
```

- `strcmp(s1, s2) == 0` se as cadeias s1 e s2 são iguais
- `strcmp(s1, s2) > 0` se a cadeia s1 é maior (alfabeticamente) que s2
- `strcmp(s1, s2) < 0` se a cadeia s1 é menor (alfabeticamente) que s2

7. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   char matricula[8];
   char turma;
   float p1;
   float p2;
   float p3;
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar o cadastro de alunos de uma disciplina, implemente uma função que crie uma cópia de uma lista encadeada definida pela estrutura acima. Essa função deve receber como parâmetro a lista a ser copiada e retornar uma cópia dessa lista. Essa função deve obedecer o protótipo:

```
Lista* copia(Lista* l);
```

8. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   int matricula;
   char departamento[21];
   float salario;
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar o cadastro de funcionários de uma empresa, implemente uma função que crie uma cópia de uma lista encadeada definida pela estrutura acima. Essa função deve receber como parâmetro a lista a ser copiada e retornar uma cópia dessa lista. Essa função deve obedecer o protótipo:

```
Lista* copia(Lista* l);
```

9. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   char telefone[15];
   char celular[15];
   char endereco[101];
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar uma agenda de telefones, implemente uma função que crie uma cópia de uma lista encadeada definida pela estrutura acima. Essa função deve receber como parâmetro a lista a ser copiada e retornar uma cópia dessa lista. Essa função deve obedecer o protótipo:

```
Lista* copia(Lista* l);
```

10. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   char matricula[8];
   char turma;
   float p1;
   float p2;
   float p3;
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar o cadastro de alunos de uma disciplina, implemente uma função que teste se duas listas encadeadas, definidas pela estrutura acima, são iguais, isto é, se as listas possuem a mesma seqüência de informações. Essa função deve obedecer o protótipo:

```
int listas_iguais(Lista* l1, Lista* l2);
```

Obs.: A função `listas_iguais` deve retornar 1 se `l1` e `l2` possuem a mesma seqüência de informações, e 0 caso contrário.
Dica: Utilize a função `strcmp`, definida no arquivo de cabeçalhos `string.h`, para comparar duas cadeias de caracteres. O cabeçalho e o modo de uso de `strcmp` são descritos a seguir:

```
int strcmp (char* s1, char* s2);
```

- `strcmp(s1, s2) == 0` se as cadeias s1 e s2 são iguais
- `strcmp(s1, s2) > 0` se a cadeia s1 é maior (alfabeticamente) que s2
- `strcmp(s1, s2) < 0` se a cadeia s1 é menor (alfabeticamente) que s2

11. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   int matricula;
   char departamento[21];
   float salario;
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar o cadastro de funcionários de uma empresa, implemente uma função que teste se duas listas encadeadas, definidas pela estrutura acima, são iguais, isto é, se as listas possuem a mesma seqüência de informações. Essa função deve obedecer o protótipo:

```
int listas_iguais(Lista* l1, Lista* l2);
```

Obs.: A função `listas_iguais` deve retornar 1 se `l1` e `l2` possuem a mesma seqüência de informações, e 0 caso contrário.
Dica: Utilize a função `strcmp`, definida no arquivo de cabeçalhos `string.h`, para comparar duas cadeias de caracteres. O cabeçalho e o modo de uso de `strcmp` são descritos a seguir:

```
int strcmp (char* s1, char* s2);
```

- `strcmp(s1, s2) == 0` se as cadeias s1 e s2 são iguais
- `strcmp(s1, s2) > 0` se a cadeia s1 é maior (alfabeticamente) que s2
- `strcmp(s1, s2) < 0` se a cadeia s1 é menor (alfabeticamente) que s2

12. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   char telefone[15];
   char celular[15];
   char endereco[101];
   struct lista* prox;
};
typedef struct lista Lista;
```

para representar uma agenda de telefones, implemente uma função que teste se duas listas encadeadas, definidas pela estrutura acima, são iguais, isto é, se as listas possuem a mesma seqüência de informações. Essa função deve obedecer o protótipo:

```
int listas_iguais(Lista* l1, Lista* l2);
```

Obs.: A função `listas_iguais` deve retornar 1 se `l1` e `l2` possuem a mesma seqüência de informações, e 0 caso contrário.
Dica: Utilize a função `strcmp`, definida no arquivo de cabeçalhos `string.h`, para comparar duas cadeias de caracteres. O cabeçalho e o modo de uso de `strcmp` são descritos a seguir:

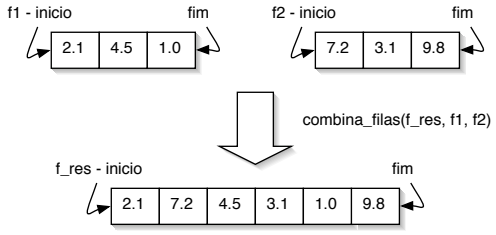
```
int strcmp (char* s1, char* s2);
```

- `strcmp(s1, s2) == 0` se as cadeias s1 e s2 são iguais
- `strcmp(s1, s2) > 0` se a cadeia s1 é maior (alfabeticamente) que s2
- `strcmp(s1, s2) < 0` se a cadeia s1 é menor (alfabeticamente) que s2

13. Considere a existência de um tipo abstrato Fila de números de ponto flutuante, cuja interface está definida no arquivo `fila.h` da seguinte forma:

```
typedef struct fila Fila;
Fila* cria(void);
void insere (Fila* f, float v);
float retira (Fila* f);
int vazia (Fila* f);
void libera (Fila* f);
```

Sem conhecer a representação interna desse tipo abstrato `Fila` e usando apenas as funções declaradas no arquivo `fila.h`, implemente uma função que receba três filas, `f_res`, `f1` e `f2`, e transfira alternadamente os elementos de `f1` e `f2` para `f_res`, conforme ilustrado a seguir:



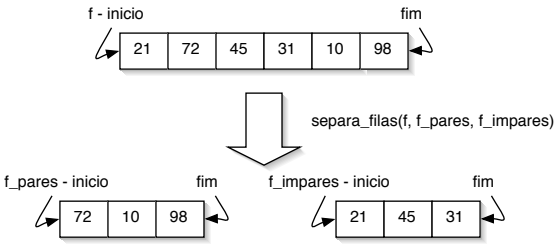
Note que, ao final dessa função, as filas `f1` e `f2` vão estar vazias e a fila `f_res` vai conter todos os valores que estavam originalmente em `f1` e `f2` (inicialmente `f_res` pode ou não estar vazia). Essa função deve obedecer o protótipo:

```
void combina_filas (Fila* f_res, Fila* f1, Fila* f2);
```

14. Considere a existência de um tipo abstrato Fila de números inteiros, cuja interface está definida no arquivo `fila.h` da seguinte forma:

```
typedef struct fila Fila;
Fila* cria(void);
void insere (Fila* f, int v);
int retira (Fila* f);
int vazia (Fila* f);
void libera (Fila* f);
```

Sem conhecer a representação interna desse tipo abstrato `Fila` e usando apenas as funções declaradas no arquivo `fila.h`, implemente uma função que receba três filas, `f`, `f_impares` e `f_pares`, e separe todos os valores guardados em `f` de tal forma que os valores pares são movidos para a fila `f_pares` e os valores ímpares para `f_impares`, conforme ilustrado a seguir:



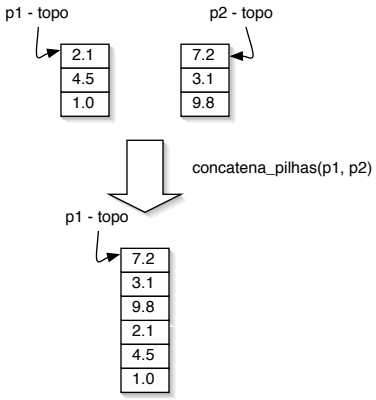
Note que, ao final dessa função, a fila `f` vai estar vazia. Essa função deve obedecer o protótipo:

```
void separa_filas (Fila* f, Fila* f_pares, Fila* f_impares);
```

15. Considere a existência de um tipo abstrato Pilha de números de ponto flutuante, cuja interface está definida no arquivo `pilha.h` da seguinte forma:

```
typedef struct pilha Pilha;
Pilha* cria(void);
void push (Pilha* p, float v);
float pop (Pilha* p);
int vazia (Pilha* p);
void libera (Pilha* p);
```

Sem conhecer a representação interna desse tipo abstrato `Pilha` e usando apenas as funções declaradas no arquivo `pilha.h`, implemente uma função que receba duas pilhas, `p1` e `p2`, e passe todos os elementos da pilha `p2` para o topo da pilha `p1`. A figura a seguir ilustra essa concatenação de pilhas:



Note que ao final dessa função, a pilha `p2` vai estar vazia e a pilha `p1` conterá todos os elementos das duas pilhas. Essa função deve obedecer o protótipo:

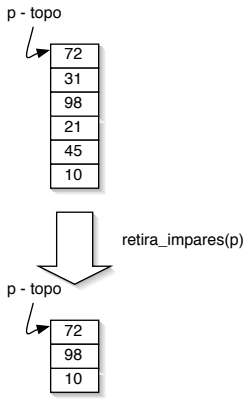
```
void concatena_pilhas (Pilha* p1, Pilha* p2);
```

Dica: Essa função pode ser implementada mais facilmente através de uma solução recursiva ou utilizando uma outra variável pilha auxiliar para fazer a transferência dos elementos entre as duas pilhas.

16. Considere a existência de um tipo abstrato Pilha de números inteiros, cuja interface está definida no arquivo `pilha.h` da seguinte forma:

```
typedef struct pilha Pilha;
Pilha* cria(void);
void push (Pilha* p, int v);
int pop (Pilha* p);
int vazia (Pilha* p);
void libera (Pilha* p);
```

Sem conhecer a representação interna desse tipo abstrato `Pilha` e usando apenas as funções declaradas no arquivo `pilha.h`, implemente uma função que receba uma pilha e retire todos os elementos impares dessa pilha. A figura a seguir ilustra o resultado dessa função sobre uma pilha:



Essa função deve obedecer o protótipo:

```
void retira_impares (Pilha* p);
```

Dica: Essa função pode ser implementada mais facilmente através de uma solução recursiva ou utilizando uma outra variável pilha auxiliar.

17. Considerando as seguintes declarações de uma árvore binária

```
struct arv {
   int info;
   struct arv* esq;
   struct arv* dir;
};
typedef struct arv Arv;
```

implemente uma função que, dada uma árvore, retorne a quantidade de nós que guardam números pares. Essa função deve obedecer o protótipo:

```
int pares (Arv* a);
```

18. Considerando as seguintes declarações de uma árvore binária

```
struct arv {
   int info;
   struct arv* esq;
   struct arv* dir;
};
typedef struct arv Arv;
```

implemente uma função que, dada uma árvore, retorne a quantidade de nós que guardam valores maiores que um determinado valor `x` (também passado como parâmetro). Essa função deve obedecer o protótipo:

```
int maiores (Arv* a, int x);
```

19. Considerando as seguintes declarações de uma árvore binária

```
struct arv {
   int info;
   struct arv* esq;
   struct arv* dir;
};
typedef struct arv Arv;
```

implemente uma função que, dada uma árvore, retorne a quantidade de folhas dessa árvore. Essa função deve obedecer o protótipo:

```
int folhas (Arv* a);
```

20. Considerando as seguintes declarações de uma árvore binária

```
struct arv {
   int info;
   struct arv* esq;
   struct arv* dir;
};
typedef struct arv Arv;
```

implemente uma função que, dada uma árvore, retorne a quantidade de nós que possuem **apenas um** filho. Essa função deve obedecer o protótipo:

```
int um_filho (Arv* a);
```

21. Considerando as seguintes declarações de uma árvore binária

```
struct arv {
   int info;
   struct arv* esq;
   struct arv* dir;
};
typedef struct arv Arv;
```

implemente uma função que, dada uma árvore, retorne a quantidade de nós que não são folhas, isto é, nós que possuem **pelo menos um** filho. Essa função deve obedecer o protótipo:

```
int intermediarios (Arv* a);
```

22. Considerando as seguintes declarações de uma árvore genérica

```
struct arvgen {
   int info;
   struct arvgen* primeiro_filho;
   struct arvgen* proximo_irmao;
};
typedef struct arvgen ArvGen;
```

implemente uma função que, dada uma árvore, retorne a quantidade de nós que guardam números pares. Essa função deve obedecer o protótipo:

```
int pares (ArvGen* a);
```

23. Considerando as seguintes declarações de uma árvore genérica

```
struct arvgen {
   int info;
   struct arvgen* primeiro_filho;
   struct arvgen* proximo_irmao;
};
typedef struct arvgen ArvGen;
```

implemente uma função que, dada uma árvore, retorne a quantidade de nós que guardam valores maiores que um determinado valor `x` (também passado como parâmetro). Essa função deve obedecer o protótipo:

```
int maiores (ArvGen* a, int x);
```

24. Considerando as seguintes declarações de uma árvore genérica

```
struct arvgen {
   int info;
   struct arvgen* primeiro_filho;
   struct arvgen* proximo_irmao;
};
typedef struct arvgen ArvGen;
```

implemente uma função que, dada uma árvore, retorne a quantidade de folhas dessa árvore. Essa função deve obedecer o protótipo:

```
int folhas (ArvGen* a);
```

25. Considerando as seguintes declarações de uma árvore genérica

```
struct arvgen {
   int info;
   struct arvgen* primeiro_filho;
   struct arvgen* proximo_irmao;
};
typedef struct arvgen ArvGen;
```

implemente uma função que, dada uma árvore, retorne a quantidade de nós que possuem **apenas um** filho. Essa função deve obedecer o protótipo:

```
int um_filho (ArvGen* a);
```

26. Considerando as seguintes declarações de uma árvore genérica

```
struct arvgen {
   int info;
   struct arvgen* primeiro_filho;
   struct arvgen* proximo_irmao;
};
typedef struct arvgen ArvGen;
```

implemente uma função que, dada uma árvore, retorne a quantidade de nós que não são folhas, isto é, nós que possuem **pelo menos um** filho. Essa função deve obedecer o protótipo:

```
int intermediarios (ArvGen* a);
```