Terceira Lista de Exercícios de Estruturas de Dados - 2002.2

1 Listas Encadeadas

1. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   float info;
   struct lista* prox;
};
typedef struct lista Lista;
```

   implemente uma função que tenha como valor de retorno o comprimento de uma lista encadeada, isto é, calcule o número de nós de uma lista. Essa função deve obedecer o protótipo:

```
int comprimento (Lista* l);
```

2. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   int info;
   struct lista* prox;
};
typedef struct lista Lista;
```

   implemente uma função que receba como parâmetros uma lista encadeada e um número inteiro `n` e retorne o número de nós da lista que possuem um campo `info` com valores **maiores** do que `n`. Essa função deve obedecer o protótipo:

```
int maiores (Lista* l, int n);
```

3. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char info;
   struct lista* prox;
};
typedef struct lista Lista;
```

   implemente uma função que receba como parâmetros uma lista encadeada e dois caracteres (`original` e `novo`) e troque todas as ocorrências do caractere `original` pelo caractere `novo`. Essa função deve obedecer o protótipo:

```
void troca (Lista* l, char original, char novo);
```

Dica: Para não precisar modificar o encadeamento da lista, basta alterar apenas o campo `info` dos nós da lista.

4. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   int info;
   struct lista* prox;
};
typedef struct lista Lista;
```

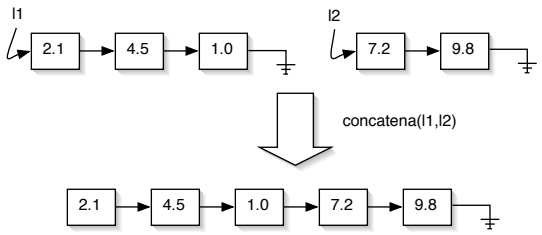
implemente uma função que tenha como valor de retorno um ponteiro para o **último** nó de uma lista encadeada que contém um determinado valor inteiro `x`. Essa função deve obedecer o protótipo:

```
Lista* ultimo_x (Lista* l, int x);
```

5. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   float info;
   struct lista* prox;
};
typedef struct lista Lista;
```

implemente uma função que receba duas listas encadeadas e retorne a lista resultante da concatenação das duas listas recebidas como parâmetro, isto é, após a concatenação, o último elemento da primeira lista deve apontar para o primeiro elemento da segunda lista, conforme ilustrado a seguir:



Essa função deve obedecer o protótipo:

```
Lista* concatena (Lista* l1, Lista* l2);
```

6. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   int info;
   struct lista* prox;
};
typedef struct lista Lista;
```

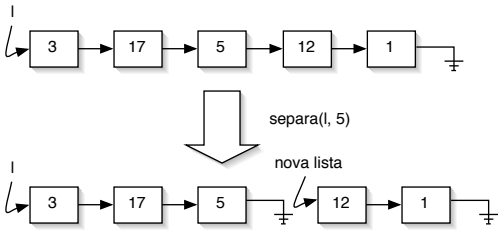
implemente uma função que receba como parâmetro uma lista encadeada e um valor inteiro `n`, retire da lista **todas** as ocorrências de `n`, e retorne a lista resultante. Essa função deve obedecer o protótipo:

```
Lista* retira_n (Lista* l, int n);
```

7. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   int info;
   struct lista* prox;
};
typedef struct lista Lista;
```

implemente uma função que receba como parâmetro uma lista encadeada e um valor inteiro `n`, e divida a lista em duas, de tal forma que a segunda lista comece no primeiro nó logo após a primeira ocorrência de `n` na lista original. A figura a seguir ilustra essa separação:



Essa função deve obedecer o protótipo:

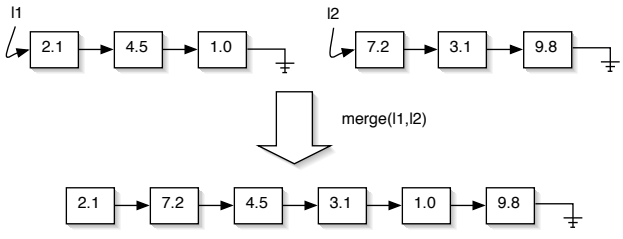
```
Lista* separa (Lista* l, int n);
```

onde o parâmetro `l` representa a lista que deve ser dividida e `n` o valor do nó que deve ser usado como ponto de quebra da lista. A função retorna um ponteiro para a segunda sub-divisão da lista original, enquanto `l` vai continuar apontando para o primeiro elemento da primeira sub-divisão da lista.

8. ***** Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   float info;
   struct lista* prox;
};
typedef struct lista Lista;
```

implemente uma função que construa uma nova lista a partir do intercalamento dos nós de outras duas listas. Essa função deve receber como parâmetro as duas listas a serem intercaladas e retornar a lista resultante, conforme ilustrado a seguir:



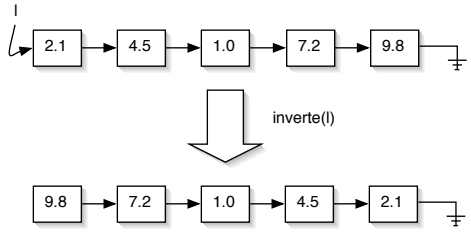
Essa função deve obedecer o protótipo:

```
Lista* merge (Lista* l1, Lista* l2);
```

9. ***** Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   float info;
   struct lista* prox;
};
typedef struct lista Lista;
```

implemente uma função que receba como parâmetro uma lista encadeada e inverta o encadeamento de seus nós, retornando a lista resultante. Após a execução dessa função, cada nó da lista vai estar apontando para o nó que originalmente era seu antecessor, e o último nó da lista passará a ser o primeiro nó da lista invertida, conforme ilustrado a seguir:



Essa função deve obedecer o protótipo:

```
Lista* inverte (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 imprima o número de matrícula, o nome, a turma e a média de todos os alunos que **não** foram aprovados na disciplina. Os dados de cada aluno reprovado na disciplina devem ser impressos em uma única linha, seguindo o formato

```
numero_de_matricula nome_do_aluno turma media
```

Assuma que o critério para aprovação é

$$\frac{p1 + p2 + p3}{3} \geq 5.0$$

Essa função deve obedecer o protótipo:

```
void imprime_reprovados (Lista* turmas);
```

11. 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 imprima o número de matrícula, o nome, a turma e a média de todos os alunos que tiveram a média de suas três provas maior ou igual a um determinado valor passado como parâmetro. Os dados de cada aluno selecionado devem ser impressos em uma única linha, seguindo o formato

```
numero_de_matricula nome_do_aluno turma media
```

A média do aluno é calculada pela fórmula $\frac{p1+p2+p3}{3}$. Essa função deve obedecer o protótipo:

```
void imprime_media (Lista* turmas, float media);
```

onde o parâmetro `turmas` representa a lista com o cadastro de alunos e o parâmetro `media` é o valor da média mínima que queremos usar para selecionar os dados dos alunos que devem ser impressos.

12. 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 imprima o número de matrícula, o nome, a turma e a média de todos os alunos que pertencem a uma determinada turma. Os dados de cada aluno da turma selecionada devem ser impressos em uma única linha, seguindo o formato

```
numero_de_matricula nome_do_aluno turma media
```

A média do aluno é calculada pela fórmula $\frac{p1+p2+p3}{3}$. Essa função deve obedecer o protótipo:

```
void imprime_turma (Lista* turmas, char turma);
```

onde o parâmetro `turmas` representa a lista com o cadastro de alunos e o parâmetro `turma` indica a turma desejada.

13. 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

14. 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.

15. 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

16. 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.

17. 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

18. 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

19. 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);
```

20. 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);
```

21. Considerando as seguintes declarações de uma lista encadeada

```
struct lista {
   char nome[81];
   char telefone[15];
   char celular[15];
   char endereco[101];
};
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);
```

22. 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

23. 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

24. Considerando as seguintes declarações de uma lista encadeada

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

2 Pilhas e Filas

1. 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);
```

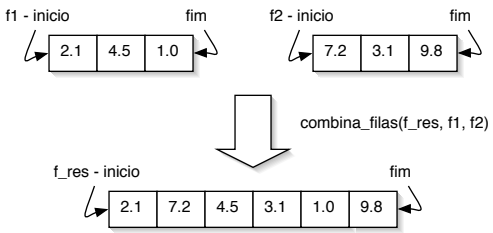
Sem conhecer a representação interna desse tipo abstrato `Pilha` e usando apenas as funções declaradas no arquivo `pilha.h`, implemente uma função que receba uma pilha como parâmetro e retorne o valor armazenado em seu topo, sem remover este valor da pilha. Essa função deve obedecer o protótipo:

```
float topo (Pilha* p);
```

2. 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:



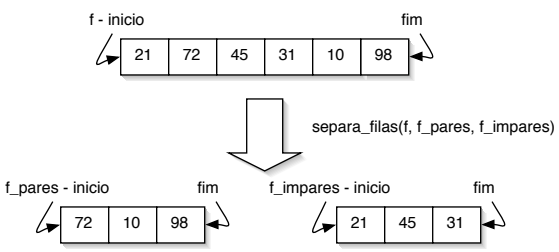
Note que, ao final dessa função, as filas `f1` e `f2` vão estar vazias e a fila `f_res` vai conter todos os valores que estavam originalmente em `f1` e `f2` (inicialmente `f_res` pode ou não estar vazia). Essa função deve obedecer o protótipo:

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

3. 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:



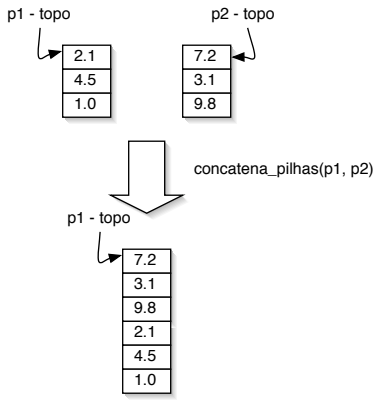
Note que, ao final dessa função, a fila `f` vai estar vazia. Essa função deve obedecer o protótipo:

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

4. 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:

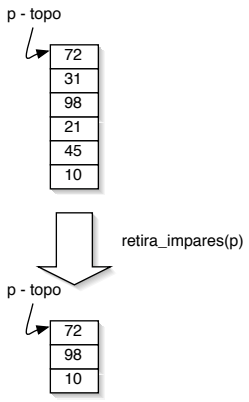
```
void concatena_pilhas (Pilha* p1, Pilha* p2);
```

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.

5. 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.

6. * 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 uma pilha como parâmetro e retorne como resultado uma cópia dessa pilha. Essa função deve obedecer o protótipo:

```
Pilha* copia_pilha (Pilha* p);
```

Obs.: Ao final da função `copia_pilha`, a pilha `p` recebida como parâmetro deve estar no mesmo estado em que ela começou a função.
Dica: Essa função pode ser implementada mais facilmente através de uma solução recursiva ou utilizando uma outra variável pilha auxiliar.

7. * 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 teste se duas pilhas são iguais ou não. Essa função deve obedecer o protótipo:

```
int pilhas_iguais (Pilha* p1, Pilha* p2);
```

onde `p1` e `p2` são as duas pilhas que devem ser comparadas. A função `pilhas_iguais` deve retornar 1 se `p1` e `p2` forem iguais, e 0 caso contrário.
Obs.: Ao final da função `pilhas_iguais`, as duas pilhas recebidas como parâmetro devem estar no mesmo estado em que elas começaram a função.
Dica: Essa função pode ser implementada mais facilmente através de uma solução recursiva ou utilizando uma outra variável pilha auxiliar.

3 Árvores Binárias

1. 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);
```

2. 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);
```

3. 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);
```

4. 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);
```

5. 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);
```

4 Árvores Genéricas

1. 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);
```

2. 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);
```

3. 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);
```

4. 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);
```

5. 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);
```