

Tópico: **Listas, Filas e Pilhas**

1. Implemente todas as funções abaixo de acordo com os protótipos declarados:

```
// estrutura de nó para lista encadeada
typedef struct no {
    int info;
    struct no *prox;
} Lista;

// Testa se uma lista é vazia
// Entrada: lista
// Retorno: 1 se a lista é vazia ou 0 caso contrário
// Pré-condição: nenhuma
// Pós-condição: nenhuma
int vazia(Lista * l);

// Insere um elemento na lista
// Entrada: lista e elemento a ser inserido
// Retorno: lista alterada
// Pré-condição: nenhuma
// Pós-condição: elemento é inserido na lista
Lista* insere(Lista* l, int info);

// Remove um elemento da lista
// Entrada: lista e elemento a ser removido
// Retorno: lista alterada
// Pré-condição: nenhuma
// Pós-condição: elemento é removido da lista
Lista* remove(Lista* l, int info);

// Imprime os elementos da lista
// Entrada: lista
// Retorno: nenhum
// Pré-condição: nenhuma
// Pós-condição: os elementos são impressos no console
void imprime(Lista* l);

// Inverte uma lista
// Entrada: lista
// Retorno: lista invertida
// Pré-condição: nenhuma
// Pós-condição: lista original não é modificada
Lista* inverte(Lista* l);

// Concatena duas listas
// Entrada: listas a serem concatenadas
// Retorno: lista resultante da concatenação
// Pré-condição: nenhuma
// Pós-condição: listas originais não são modificadas
Lista* concatena(Lista* l1, Lista* l2)
```

```
// Intercala duas listas
// Entrada: listas a serem intercaladas
// Retorno: lista resultante da intercalação
// Pré-condição: nenhuma
// Pós-condição: listas originais permanecem inalteradas
Lista* merge(Lista* l1, Lista* l2)
```

(a) 1ª abordagem:

- as funções são implementadas de maneira iterativa
- os elementos podem estar desordenados
- pode haver elementos repetidos e a operação **remove** retira apenas a 1ª ocorrência do elemento

(b) 2ª abordagem:

- as funções devem ser implementadas de maneira recursiva
- os elementos podem estar desordenados
- pode haver elementos repetidos e a operação **remove** retira apenas a 1ª ocorrência do elemento

(c) 3ª abordagem:

- as funções devem ser implementadas de maneira recursiva
- os elementos devem estar sempre ordenados
- pode haver elementos repetidos e a operação **remove** retira todas as ocorrências do elemento

(d) implemente a função `Lista* append(Lista* l1, Lista* l2)` que concatena l2 no final de l1 retornando l1 modificada.

(e) implemente a função `int conta_ocorrencias(Lista* l, int x)` que retorna o número de ocorrências do elemento x na lista.

(f) implemente a função `Lista* elimina_repetidos(Lista* l)` que elimina elementos repetidos, deixando somente uma única ocorrência de cada elemento.

2. Implemente uma estrutura de dados que faça uso de lista encadeada e represente um conjunto tal qual na Matemática, isto é, não há elementos repetidos. Além disso, implemente funções que efetue as seguintes operações:

(a) `int pertence(Lista* l, int x)` que retorna 1 caso x pertença a l ou 0 caso contrário.

(b) união de 2 conjuntos

(c) interseção de 2 conjuntos

(d) diferença entre 2 conjuntos

(e) `int esta_contido(Lista* l1, Lista* l2)` que retorna 1 caso l2 esteja contido em l1 ou 0 caso contrário.

(f) `void imprime(Lista* l)` que imprime os elementos do conjunto l

3. Considere o código abaixo:

```
struct no {
    int info;
    struct no * prox;
};
```

```
typedef struct { // estrutura para lista encadeada com cabeça e cauda
    struct no* cabeca;
    struct no* cauda;
} Lista;
```

Defina protótipos e implemente funções para:

- verificar se uma lista é vazia
- inserir um elemento na cabeça
- inserir um elemento na cauda
- remover um dado elemento da lista

4. Considere o código abaixo:

```
// estrutura de nó para lista duplamente encadeada
typedef struct no {
    int info;
    struct no *ant;
    struct no *prox;
} Lista;
```

Implemente as mesmas funções definidas no exercício 1, mantendo o mesmo protótipo.

5. Considere o código abaixo:

```
// estrutura de nó para lista duplamente encadeada circular
typedef struct no {
    int info;
    struct no *ant;
    struct no *prox;
} Lista;
```

Implemente as mesmas funções definidas no exercício 1, mantendo o mesmo protótipo.

6. Implemente as funções abaixo de acordo com os protótipos declarados:

```
// estrutura de nó para lista encadeada
struct no {
    int info;
    struct no *prox;
};

// estrutura para fila
struct fila {
    int n; // número de elementos
    struct no* prim;
    struct no* ultimo;
};

// Testa se a fila está vazia
// Entrada: fila
// Retorno: 1 se a fila é vazia ou 0 caso contrário
// Pré-condição: ponteiro não nulo p/ estrutura fila
// Pós-condição: nenhuma
int vazia(struct fila * f);

// Insere um elemento na fila
// Entrada: fila e inteiro a ser inserido
// Retorno: nenhum
// Pré-condição: ponteiro não nulo para estrutura fila
// Pós-condição: elemento é inserido na fila
void enqueue(struct fila * f, int info);
```

```

// Remove um elemento da fila
// Entrada: fila e inteiro a ser removido
// Retorno: nenhum
// Pré-condição: ponteiro não nulo para estrutura fila
// Pós-condição: elemento é removido da fila
void dequeue(struct fila* f, int info);

// Imprime os elementos da fila
// Entrada: fila
// Retorno: nenhum
// Pré-condição: ponteiro não nulo para estrutura fila
// Pós-condição: os elementos são impressos no console
void imprime(struct fila* f);

```

Além disso, implemente a função `void inverte(struct fila * f)` que inverte a ordem dos elementos na fila, usando somente as operações usuais da estrutura de fila.

7. Implemente as funções declaradas no código abaixo:

```

// estrutura de nó para lista encadeada
struct no {
    int info;
    struct no *prox;
};

// Testa se uma pilha é vazia
// Entrada: pilha
// Retorno: 1 se a pilha é vazia ou 0 caso contrário
// Pré-condição: nenhuma
// Pós-condição: nenhuma
int vazia(struct no * st);

// Empilha um elemento na pilha
// Entrada: pilha e inteiro a ser empilhado
// Retorno: pilha alterada
// Pré-condição: nenhuma
// Pós-condição: elemento é empilhado
struct no* push(struct no* st, int info);

// Desempilha um elemento da pilha
// Entrada: pilha
// Retorno: pilha alterada
// Pré-condição: nenhuma
// Pós-condição: elemento do topo é removido
struct no* pop(struct no* st, int info);

// Lê o topo da pilha
// Entrada: pilha
// Retorno: elemento do topo da pilha ou -1 se a pilha está vazia
// Pré-condição: nenhuma
// Pós-condição: retorna o topo da pilha
int topo(struct no* st);

// Imprime os elementos da pilha
// Entrada: pilha
// Retorno: nenhum

```

```
// Pré-condição: nenhuma  
// Pós-condição: os elementos são impressos no console  
void imprime(struct no* st);
```