Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento Acadêmico de Informática (DAINF)
Estrutura de Dados I
Professor: Rodrigo Minetto
Lista de exercícios (lista com encadeamento duplo com ponteiro para cauda)

Exercício 1) Projete uma função **imprimir**, para uma lista com encadeamento duplo, tal que a ordem de impressão dos itens é na direção da cabeça para a cauda — considere que a lista tem ponteiros de **acesso direto** tanto para a **cabeça** quanto para a **cauda**. Utilize o seguinte protótipo para a sua função:

```
void print_head_to_tail (List *l);
```

Exercício 2) Projete uma função **imprimir**, para uma lista com encadeamento duplo, tal que a ordem de impressão dos itens é na direção da cauda para a cabeça — considere que a lista tem ponteiros de **acesso direto** tanto para a **cabeça** quanto para a **cauda**. Utilize o seguinte protótipo para a sua função:

```
void print_tail_to_head (List *l);
```

Exercício 3) Projete uma função para **inserir** um item na cauda de uma lista com encadeamento duplo — considere que a lista tem ponteiros de **acesso direto** tanto para a **cabeça** quanto para a **cauda**. Utilize o seguinte protótipo para a sua função:

```
void insert_back (List *l, int elem);
```

Observação: verifique a corretude da sua solução ao imprimir a lista na direção cabeça para cauda (e vice-versa) para validar o encadeamento.

Exercício 4) Projete uma função para **remover** um item na cauda de uma lista com encadeamento duplo — considere que a lista tem ponteiros de **acesso direto** tanto para a **cabeça** quanto para a **cauda**. Utilize o seguinte protótipo para a sua função:

```
void remove_back (List *l);
```

Observação: verifique a corretude da sua solução ao imprimir a lista na direção cabeça para cauda (e vice-versa) para validar o encadeamento.

Exercício 5) Projete uma função para **remover** a primeira ocorrência de um item k em uma lista com encadeamento duplo — considere que a lista tem ponteiros de **acesso direto** tanto para a **cabeça** quanto para a **cauda**. Utilize o seguinte protótipo para a sua função:

```
void remove_k (List *l, int k);
```

Se existir mais de uma ocorrência de k na lista (itens repetidos) eliminar a primeira ocorrência. Observação: verifique a corretude da sua solução ao imprimir a lista na direção cabeça para cauda (e vice-versa) para validar o encadeamento.

Ps. você pode usar outras funções para compor a solução deste exercício.

Exercício 6) Projete uma estrutura de dados **fila** baseada em uma lista com encadeamento duplo — considere uma lista com ponteiros de **acesso direto** a **cabeça** e **cauda**. O objetivo é projetar e verificar a corretude das seguintes funções:

```
void enqueue (List *l, int elem);
int dequeue (List *l);
int front (List *l);
int empty_queue (List *l);
```

Exercício 7) Projete uma estrutura de dados **pilha** baseada em uma lista com encadeamento duplo — considere uma lista com ponteiros de **acesso direto** a **cabeça** e **cauda**. O objetivo é projetar e verificar a corretude das seguintes funções:

```
void push (List *l, int elem);
int pop (List *l);
int peek (List *l);
int empty_stack (List *l);
```

Exercício 8) Qual a complexidade, utilizando a notação assintótica $\mathcal{O}(?)$, para as seguintes operações abaixo, implementadas em uma lista com encadeamento duplo com ponteiros de acesso direto a **cabeça** e **cauda**).

	Melhor caso	Pior caso
enqueue		
dequeue		
push		
pop		

Exercício 9) Qual tipo de operação em uma lista encadeada que tem a mesma complexidade tanto em uma lista com encadeamento simples, como encadeamento duplo, ou encadeamento duplo com acesso direto aos elementos da cabeça e cauda?

Exercício 10) Projete uma função para localizar e retornar o valor do elemento que se situa aproximadamente no meio de uma lista com encadeamento duplo — considere uma lista com ponteiros de **acesso direto** a **cabeça** e **cauda**. Utilize o seguinte protótipo para a sua função:

```
int middle (List *l);
```

Se a lista tiver um número par de elementos então retorne o valor da esquerda.

Exercício 11) Projete uma função que determina se uma cadeia de números inteiros armazenados em nós consecutivos de uma lista com encadeamento duplo — com ponteiros de acesso direto a **cabeça** e **cauda** — é palíndromo (**true**) ou não (**false**).

```
|1|<->|2|<->|1|        (true)
|1|<->|2|<->|2|<->|1|  (true)
|1|<->|2|<->|3|        (false)
```

Considere o seguinte protótipo para a sua função:

```
int palindrome (List *l);
```

Exercício 12) Aplicações em criptografia envolvem a manipulação de números inteiros com centenas de números — o que não é possível ao utilizar os maiores tipos permitidos na linguagem C (o tipo unsigned long long consegue representar como maior inteiro o valor 18,446,744,073,709,551,615 (0xffffffffffffffff)). Listas encadeadas são utilizadas em bibliotecas para manipulações destes números especiais. Observe que ao somar dois números, começamos com os dígitos menos significativos em direção aos mais significativos; logo suponha que queiramos somar dois números $x = 1240$ e $y = 699$, então estes podem ser representados como uma lista duplamente encadeada com um dígito por nó da seguinte forma:

```
     h                 t           h           t
x = |1|<->|2|<->|4|<->|0|     y = |6|<->|9|<->|9|   tal que   head(h) e tail(t)
```

Usando o ponteiro para a cauda de ambas as listas começamos com o dígito menos significativo de ambos os números (0 e 9), e inserimos o resultado da soma (9) na lista duplamente encadeada de resultado em z

```
     h                 *           h           *            *
x = |1|<->|2|<->|4|<->|0|     y = |6|<->|9|<->|9|       z = |9|
```

então avançamos uma posição em x e y e realizamos novamente a soma (9 + 4), mas note que o resultado (13) ultrapassa o tamanho permitido de um dígito por nó da lista e esse **vai um** é um valor que deve ser adicionado na soma do próximo passo (então salvamos esse valor em uma variável)

```
     h           *             h     *              *
x = |1|<->|2|<->|4|<->|0|  y = |6|<->|9|<->|9|  z =  |3|<->|9|  sobra=1
```

A sobra da soma anterior é agora somada aos nós da iteração corrente (indicados por *)

```
     h     *                   *                    *
x = |1|<->|2|<->|4|<->|0|  y = |6|<->|9|<->|9|  z = |9|<->|3|<->|9|  sobra=0
```

Finalmente o número y foi finalizado, mas não o x então apenas o valor atual mais alguma sobra se existir (nesse caso ela é zero) são adicionados em z

```
     *                         *                    *
x = |1|<->|2|<->|4|<->|0|  y = |6|<->|9|<->|9|  z = |1|<->|9|<->|3|<->|9|  sobra=0
```

Para resolver este problema utilize o programa **sum-big.c** (dentro de **arquivos.zip**).

Exercício 13) Utilize o programa **sum-big.c** da questão anterior como parte da solução em um programa para multiplicar dois números utilizando listas duplamente encadedas com ponteiros para cabeça e cauda. Note que a operação de multiplicação, conforme vimos na escola, consiste em multiplicar um dígito de um número por todo o outro número, e ir somando os resultados após sucessivos deslocamentos. Implemente este algoritmo visto na escola como solução. Para resolver este problema utilize o programa **mult-big.c** (dentro de **arquivos.zip**).