Univerdade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Algoritmos e Estruturas de Dados II
2º semestre de 2017
Prof. Fernando, Luiz, Raquel

Exercício de programação 4

Listas circulares duplamente encadeadas

Valor: 5 pontos
Data de entrega: 15/10/2017 (**não será adiado**)

1 Problema

Você foi contratado para desenvolver as funcionalidades de uma *playlist* do **Spotify**. Os contratantes precisam que essa *playlist* tenha as seguintes **operações**:

- Inserir uma música **antes** ou **depois** da música atual;
- Excluir a música atual;
- Ir para a próxima música;
- Voltar à música anterior;
- Limpar a *playlist*.

Além disso, deve ser possível tocar a *playlist* indefinidamente, ou seja, é necesário que se use uma **lista circular duplamente encadeada** para tal implementação.

2 Tipo abstrato de dados

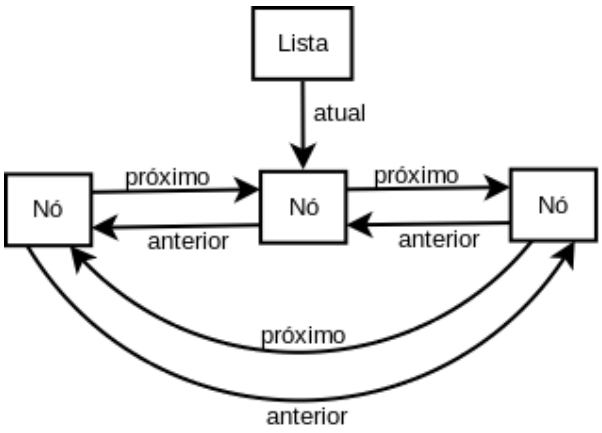

Figura 1: Lista circular duplamente encadeada

Esse tipo de lista, como apresentado na figura 1, possui nós com apontadores para o próximo nó e para o nó anterior. Além disso, precisa-se apontar para o item *inicial* da lista. Dessa forma, a lista pode ser

atravessada indefinidamente para as duas direções. O uso da célula **cabeça** é opcional, o que pode facilitar os casos da lista vazia.

Em *C*, pode-se implementar tal estrutura usando as definições do bloco de código 1.

Bloco de código 1: Tipo abstrato de dados. O tipo *Item* representa a música com seu devido identificador

```
typedef struct TipoItem TipoItem;
typedef struct TipoCelula TipoCelula;
typedef struct TipoLista TipoLista;

struct TipoItem {
  unsigned id;
};

struct TipoCelula {
  TipoCelula* ant;
  TipoCelula* prox;
  TipoItem* item;
};

struct TipoLista {
  unsigned tam;
  TipoCelula* cabeca;
};
```

3 Operações

Nessa seção são reapresentadas as operações a serem implementadas, suas complexidades e alguns exemplos de cada uma. Uma música é representada pelo seu **identificador** (*id*). A lista de comandos é apresentada na tabela 1.

Tabela 1: Lista de comandos

Operação	Custo	Descrição	Código
Inserir após a música atual	O(1)	Insere-se uma música com o *id* informado após a música atual apontada pela lista. **Após a inserção, a música inserida se torna a música atual**.	A<:id>
Inserir antes da música atual	O(1)	Insere-se uma música com o *id* informado antes da música atual apontada pela lista. **Após a inserção, a música inserida se torna a música atual**.	B<:id>
Remover a música atual	O(1)	Remove-se a música atual apontada pela lista. **A música anterior à removida se torna a nova música atual**.	D
Próxima música	O(1)	A próxima música após a atual se torna a nova música atual.	N
Música anterior	O(1)	A música anterior à atual se torna a nova música atual.	P
Limpar a lista	O(N)	Remove todas as músicas da lista.	C

Obs.: Todas as operações devem tratar o caso em que a lista está vazia.

4 Entrada

A entrada será um arquivo de texto onde cada linha representa a criação de uma *playlist* usando os códigos definidos na tabela 1. Por exemplo, a sequência $A1A2A3B4$ produz a lista $1 \leftrightarrow 2 \leftrightarrow \mathbf{4} \leftrightarrow 3 \leftrightarrow 1$, com música atual **4**. Outros exemplos são apresentados abaixo de forma a eliminar qualquer dúvida:

- $B1B2B3$: $\mathbf{3} \leftrightarrow 2 \leftrightarrow 1 \leftrightarrow \mathbf{3}$
- $B1B2B3CA$: $\emptyset$, lista vazia
- $A1A2A3D$: $1 \leftrightarrow \mathbf{2} \leftrightarrow 1$
- $A1A2A3DDD$: $\emptyset$, lista vazia
- $A1A2A3DDDDDDDDDD$: $\emptyset$, lista vazia
- $A1A2A3CCC$: $\emptyset$, lista vazia
- $A1A2A3N$: $\mathbf{1} \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow \mathbf{1}$
- $A1A2A3NNNN$: $\mathbf{1} \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow \mathbf{1}$

Existem outros exemplos no *kit* de implementação que será descrição mais a frente.

5 Saída

Para cada linha do arquivo de entrada, deve-se produzir uma linha no arquivo de saída representando o estado final da *playlist* após a aplicação de todos os comandos daquela linha. A impressão do estado da *playlist* deverá ser da seguinte forma:

Começando pela música atual apontada pela *playlist*, caminha-se para as próximas músicas exibindo-se seus **identificadores** até que se atinja a música atual. Os identificadores devem ser separados por um *espaço* ' '. Exemplos (os espaços nos exemplos estão representados por '_'. '\n' representa uma nova linha):

- $B1B2B3$: $\mathbf{3} \leftrightarrow 2 \leftrightarrow 1 \leftrightarrow \mathbf{3}$:

  `3_2_1_\n`

- $B1B2B3CA$: $\emptyset$, lista vazia :

  `\n`

- $A1A2A3D$: $1 \leftrightarrow \mathbf{2} \leftrightarrow 1$:

  `2_1_\n`

- $A1A2A3DDD$: $\emptyset$, lista vazia :

  `\n`

- $A1A2A3DDDDDDDDDD$: $\emptyset$, lista vazia :

  `\n`

- $A1A2A3CCC$: $\emptyset$, lista vazia :

```
\n
```

- $A1A2A3N$: **1** $\leftrightarrow$ 2 $\leftrightarrow$ 3 $\leftrightarrow$ **1** :

  ```
  1_2_3_\n
  ```

- $A1A2A3NNNN$: **1** $\leftrightarrow$ 2 $\leftrightarrow$ 3 $\leftrightarrow$ **1** :

  ```
  1_2_3_\n
  ```

6 *Kit* de desenvolvimento

O arquivo *kit.zip* possui diversos arquivos para auxiliar na implementação. Ele já define uma estrutura para o projeto. Os arquivos são os seguintes:

- *examples.txt*: 2000 exemplos de entrada gerados de forma aleatória.
- *answers.txt*: A resposta correta de cada um dos 2000 exemplos.
- *generator.py*: Um gerador de exemplos aleatórios. Para esses exemplos você não terá a resposta correta. Para gerá-los use o seguinte comando:

  ```
  make examples > new_examples.txt
  ```

  Obs.: Precisa-se da linguagem **Python** para executá-lo.

- *test.py*: Um programa que confere se a saída do seu programa no arquivo *output.txt* é igual às respostas corretas no arquivo *answers.txt*. Para executar os testes:

  ```
  make test
  ```

  Obs.: Precisa-se da linguagem **Python** para executá-lo.

- *solution/TipoLista.h*: O cabeçalho do *TipoLista*. Aqui está definido o tipo abstrato de dados como apresentado anteriormente.
- *solution/TipoLista.c*: Implementação das operações da lista definidas no cabeçalho.
- *solution/main.c*: O programa principal. Esse deverá ler o arquivo *examples.txt* executar todos os exemplos e gerar como saída o arquivo *output.txt* com formato definido na seção anterior. Para compilar o projeto:

  ```
  make build
  ```

  Para testar o programa gerado com os testes de *examples.txt*:

  ```
  make try
  ```

Todos esses comandos só vão funcionar caso se siga a estrutura proposta pelo *kit* de desenvolvimento e use os nomes de arquivos como os descritos acima. Dê uma olhada no arquivo *Makefile* para descobrir como tudo funciona.

7 Entregáveis

O exercício deve ser implementado na linguagem *C* usando bibliotecas de padrões como *stdio*, *stdlib* e possivelmente *string*. Deve-se entregar todo o código desenvolvido para resolver o problema. Esse código deve ter como arquivo principal o arquivo *main.c*. O *Makefile* deverá ser entregue junto ao código, **principalmente** no caso de alterações do mesmo. Um código bem organizado e bem comentado é um código bem **documentado**.