

# Atividade Prática 02

## “Listas duplamente encadeadas”

Universidade Tecnológica Federal do Paraná (UTFPR), campus Apucarana  
Curso de Engenharia de Computação  
Disciplina de Estrutura de Dados - ED62A - 2ºSemestre 2019  
Prof. Dr. Rafael Gomes Mantovani

### 1 Descrição

Elabore um programa em C que implemente um tipo abstrato de dados para lista duplamente encadeada, e suas operações de manipulação. Uma listas duplamente encadeada é um arranjo de dados onde cada elemento é também um tipo abstrato de nó de lista (**NoLista**) que guarda dois ponteiros, como mostrado na Figura 1:

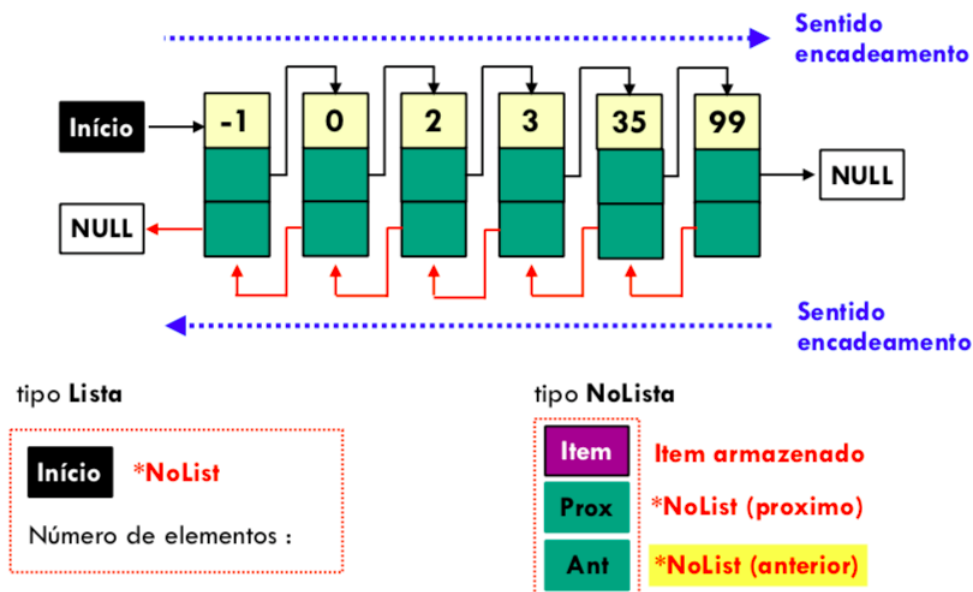


Figura 1: Diagrama representativo de uma lista duplamente encadeada com 6 elementos. No diagrama é possível também ver uma representação gráfica dos tipos abstratos de dados envolvidos em sua codificação.

- **anterior**: ponteiro que aponta para o elemento anterior na lista ordenada;
- **próximo**: ponteiro que aponta para o elemento posterior na lista ordenada.

Use as implementações das estruturas já desenvolvidas em sala para iniciar sua implementação. Além da definição e codificação dos tipos de dados, a estrutura deve implementar as seguintes funções:

Tabela 1: Operações de uma lista duplamente encadeada dinâmica. Os nomes e tipos são apenas sugestões de implementações.

Função
<pre>void iniciaLista(DLista *lista) int tamanho(DLista *lista) bool estaVazia(DLista *lista) bool insereElemento(DLista *lista, int chave) bool pesquisaElemento(DLista *lista, int chave) bool removeChave(DLista *lista, int chave, Objeto *ret) bool removePrimeiro(DLista *lista, Objeto *ret) bool removeUltimo(DLista *lista, Objeto *ret) void imprimeLista(DLista *dupla) void imprimeListaReversa(DLista *dupla) Objeto primeiro(DLista *dupla) Objeto ultimo(DLista *dupla) void destroi(ListaDupla *lista)</pre>

O programa receberá dois arquivos texto como parâmetros:

- **arquivo de entrada:** um arquivo texto contendo números inteiros para serem armazenados na lista. A primeira linha contém um caractere único (char) indicando o modo de impressão da lista no arquivo de saída: **c** - para impressão em ordem crescente, e **d** - para impressão em ordem decrescente. A segunda linha contém um ou mais números que deverão ser armazenados na lista.
- **arquivo de saída:** um arquivo texto onde deverá ser impresso na primeira linha a quantidade total de elementos lidos; e na segunda linha o conteúdo da lista de acordo com o critério selecionado (ordem crescente ou decrescente).

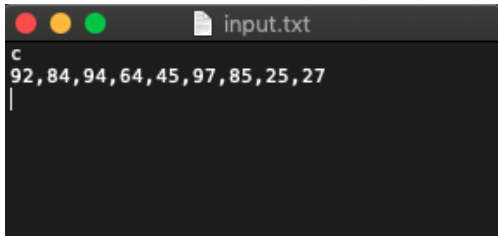
Um exemplo contendo arquivos de entrada e saída é apresentado na Figura 2. **Dica:** Para rodar o programa por linha de comando, manipular os argumentos **argc** e **argv** da função **main**. Para executar o programa por linha de comando, deve-se obedecer o seguinte padrão:

```
[nome do programa] [arquivo de entrada] [arquivo de saída]
```

Por exemplo, se o programa desenvolvido se chamar ‘**at02mantovani.c**’, então o comando de execução será:

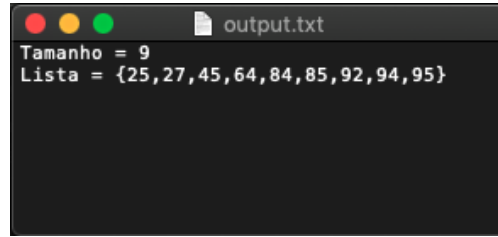
```
> ./at02mantovani input.txt output.txt
```

Lembrem-se que os nomes dos arquivos de entrada e saída podem mudar, e por isso usamos os parâmetros **argc/argv**.

A screenshot of a text editor window titled 'input.txt'. The content shows a variable 'c' followed by a list of numbers: 92,84,94,64,45,97,85,25,27.

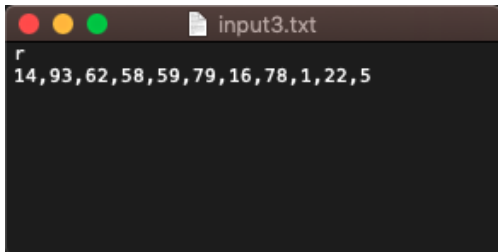
```
c
92,84,94,64,45,97,85,25,27
```

(a) Exemplo de arquivo de entrada sem erro.

A screenshot of a text editor window titled 'output.txt'. The content shows 'Tamanho = 9' and 'Lista = {25,27,45,64,84,85,92,94,95}'.

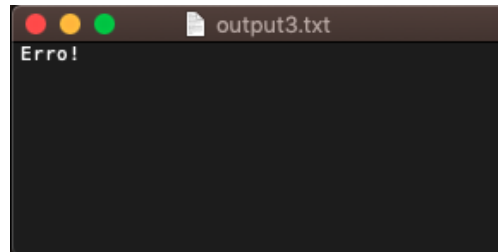
```
Tamanho = 9
Lista = {25,27,45,64,84,85,92,94,95}
```

(b) Exemplo de arquivo de saída sem erro.

A screenshot of a text editor window titled 'input3.txt'. The content shows a variable 'r' followed by a list of numbers: 14,93,62,58,59,79,16,78,1,22,5.

```
r
14,93,62,58,59,79,16,78,1,22,5
```

(c) Exemplo de arquivo de entrada com erro.

A screenshot of a text editor window titled 'output3.txt'. The content shows the word 'Erro!' in red, indicating an error.

```
Erro!
```

(d) Exemplo de arquivo de saída com erro.

Figura 2: Valores de entrada e correspondentes arquivos de saída gerado pelo programa.

## 2 Orientações gerais

- **Data de entrega:** 04/10/2019;
- Implementar também o controle de erros, para lidar com exceções que possam ocorrer. Ou seja, testem os programas para várias situações (de sucesso e de erro);
- Para acompanhamento do desenvolvimento, criar um repositório individual com o código desenvolvido no **github Classroom**, por meio do link: <https://classroom.github.com/a/vHcnbrdS>. Os repositórios serão privados, com acesso apenas do professor e do aluno. Usar esses repositórios para encaminhar dúvidas de implementação.
- Entrega do programa final: via Moodle. O aluno deve submeter o fonte no link da atividade disponibilizado na página da disciplina no Moodle.
- Os códigos desenvolvidos por cada aluno serão também verificados por ferramentas de plágio. Códigos iguais/similares terão nota zero.

## Referências

- [1] Thomas H. Cormen,; Ronald Rivest; Charles E. Leiserson; Clifford Stein. Algoritmos - Teoria e Prática - 3ª Ed. Elsevier - Campus, 2012.
- [2] Nivio Ziviani. Projeto de algoritmos com implementações: em Pascal e C. Pioneira, 1999.
- [3] Adam Drozdek. Estrutura De Dados E Algoritmos Em C++. Cengage, 2010.