

# Listas com descritor (conceitos)

## Descritor

É uma entidade que contém informações chaves de uma lista. Em C++, descritor pode ser desenvolvido como um *struct* (ou *classe*) que contém três atributos básicos:

- **início**: ponteiro do nó que representa o PRIMEIRO dado da lista;
- **fim**: ponteiro do nó que representa o ÚLTIMO dado da lista;
- **tam**: quantidade de dados da lista;

Listas podem servir para representar **tanto FILAS quanto PILHAS**, já que as dinâmicas que as diferem (referentes essencialmente à remoção de dados) estão embutidas nas listas. Portanto, em uma lista, podemos adicionar dados no início e no fim, e TAMBÉM remover dados no início e no fim, sem estar contido em um padrão fechado (como no caso de filas e pilhas).

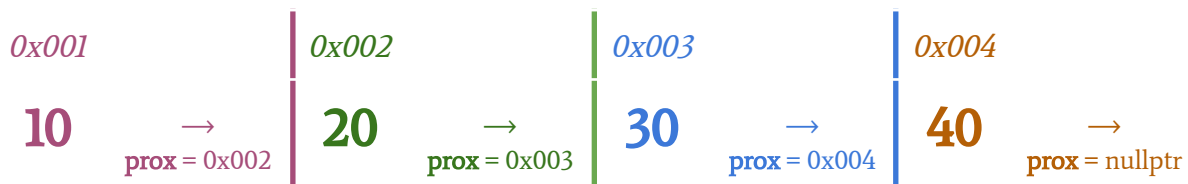
A estrutura dos nós de uma lista são as que definirão de qual tipo consiste uma lista, que são basicamente um dos dois seguintes:

---

## Lista simplesmente encadeada

Cada nó consiste em um *struct* formado por dois atributos:

- **dado**: dado armazenado para este nó;
- **prox**: ponteiro do nó que SUCEDE o atual na lista;



- As divisões e cores acima separam os nós da lista!

Portanto, no último elemento da lista, o ponteiro *prox* é NULO! Se é necessário fazer uma função para percorrer todos os nós de uma lista, **ela só pode ser realizada do início para o fim**, pois os nós SÓ referenciam o seu sucessor.

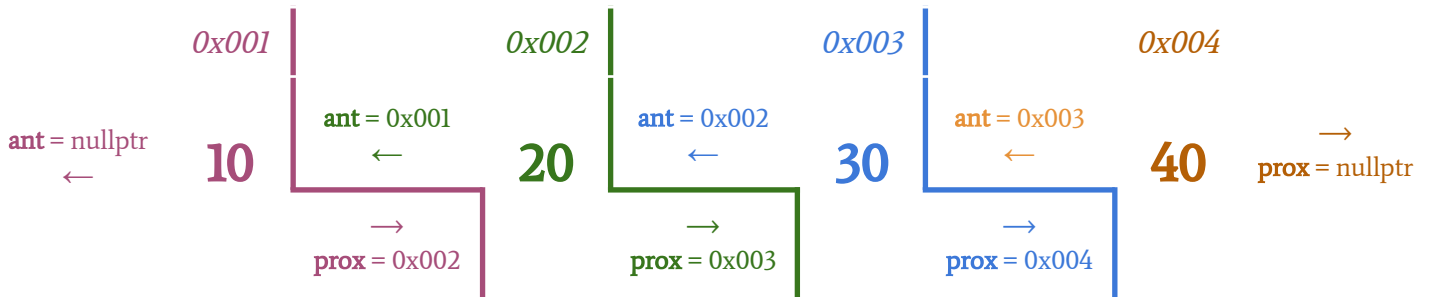
Para percorrer do fim para o início, **uma lista AUXILIAR é obrigatoriamente necessária**, o que não é muito vantajoso em termos de códigos.

---

## Lista duplamente encadeada

Um nó deste tipo de lista **TEM** os atributos *dado* e *prox* do tipo antecessor, mas com um **atributo EXTRA**:

- **ant** ponteiro do nó que ANTECEDE o atual na lista;




- As divisões e cores acima separam os nós da lista!

A vantagem do novo atributo é possibilitar **dois sentidos para percorrer uma lista** exclusivamente pelos nós - ou seja, tanto do **início pro fim** quanto do **fim pro início**. Apesar do atributo extra conferir MAIS informação a ser armazenada na lista, pelo menos em termos de processamento ela pode ser mais vantajosa!

---

No código do descritor com *template*, para criar uma lista SIMPLEMENTE encadeada, usamos a classe **ListaS** (onde os nós são objetos da classe **No**), enquanto que para lista DUPLAMENTE encadeada usamos a classe **ListaD** (onde os nós são objetos da classe **NoD**). Com uso de *templates*, toda vez que criamos uma lista, **DEVEMOS definir o TIPO dos dados contidos na lista!** Confira nos métodos e nas imagens a seguir exemplos de aplicação de cada lista!

## Métodos do descritor

- **#1:** Métodos que possuem **\*\*\*** NECESSITAM de overloading para *structs*. Confira nos exemplos!
- Métodos ou grupos de métodos marcados com  indicam que deve-se **prestar MUITA atenção** nas explicações e nas imagens mostradas de exemplo!

---

## Adição de dados (*void*)

### adInicio(*dado*) e adFim(*dado*)

- Duas funções com intuito de **adicionar um novo dado** em alguma extremidade da lista: a primeira adiciona-o no início da lista (pelo atributo *inicio*) e o segundo no final da lista (pelo atributo *fim*).



### adOrdenado(*dado*)\*\*\*

- **Adiciona um novo dado** em uma posição da lista para mantê-la completamente ordenada (para *strings* e *chars*, mantém em ordem alfabética; para *números*, mantém em ordem crescente).

```
6
7 main() {
8     setlocale(LC_ALL, "Portuguese");
9
10    ListaD<int> L;
11    L.adOrdenado(26);
12    L.adOrdenado(4);
13    L.adOrdenado(2004);
14    L.adOrdenado(-1);
15    L.adOrdenado(10);
16    L.adOrdenado(20);
17
18    cout << "> Primeiro elemento: " << L.inicio->dado;
19    cout << "\n> Último elemento: " << L.fim->dado << "\n";
20    L.mostrarSimples("> Lista ordenada: ");
21
22    cout << endl << endl;
23
24 }
```

```
"C:\Users\PC\Documents\Estrutura de Dados\Exemplos Múltiplos Descritor\exemplo8.exe"
> Primeiro elemento: -1
> Último elemento: 2004
> Lista ordenada: : -1, 4, 10, 20, 26, 2004

Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

Exemplo com lista de inteiros.



CONFIRA o exemplo do **método *adOrdenado*** para *structs* com overloading (com lista de PESSOAS)!!!

---

## Remoção de dados

### removeIni() e removeFim()

- Duas funções com intuito de **remover um nó** em alguma extremidade da lista: a primeira remove o primeiro (do atributo *inicio*) e o segundo o último (do atributo *fim*). Ambos os métodos **retornam o dado que foi removido**!



### removeVariosVal(*dado*)\*\*\*

- Remove TODOS os nós da lista cujo dado corresponde ao **dado do parâmetro**. A **função retorna um *int***, correspondente a QUANTIDADE de nós removidos (pode retornar de 0 - nenhum dado deletado - até *tam* - todos os dados deletados).

```

5  main() {
6      setlocale(LC_ALL, "Portuguese");
7
8      ListaD<int> L;
9      L.adFim(10);
10     L.adFim(15);
11     L.adFim(15);
12     L.adFim(20);
13     L.adFim(30);
14     L.adFim(40);
15     L.adFim(15);
16     L.adFim(50);
17     L.adFim(15);
18     L.adFim(15);
19
20     L.mostrarSimples("> Lista original");
21     cout << "\n\n> Removido " << L.removeVariosVal(15);
22     cout << " '15's!\n\n";
23     L.mostrarSimples("> Nova lista");
24     if(L.removeVariosVal(100) == 0) {
25         cout << "\n\n Ops... Não existe '100' na lista!";
26     }
27
28     cout << endl << endl;
29 }
```

```

> Lista original: 10, 15, 15, 20, 30, 40, 15, 50, 15, 15
> Removido 5 '15's!
> Nova lista: 10, 20, 30, 40, 50
> Ops... Não existe '100' na lista!

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```

*Exemplo de remoção de inteiros usando **removeVariosVal***



### **removeUmVal(*dado*)\*\*\***

- Tenta encontrar a primeira ocorrência de um nó cujo dado corresponde ao **dado do parâmetro** e, uma vez encontrado, REMOVE-O. A função **retorna um ponteiro**: se FOI encontrado, o ponteiro **carrega o dado deletado**; caso NÃO, o ponteiro é **NULO**!

```

5  main() {
6      setlocale(LC_ALL, "Portuguese");
7
8      ListaD<int> L;
9      L.adFim(10);
10     L.adFim(15);
11     L.adFim(15);
12     L.adFim(20);
13     L.adFim(30);
14     L.adFim(40);
15     L.adFim(15);
16     L.adFim(50);
17     L.adFim(15);
18     L.adFim(15);
19
20     L.mostrarSimples("> Lista original");
21     if(L.removeUmVal(15) == NULL) {
22         cout << "\n\n> '15' NÃO foi encontrado!!!\n\n";
23     } else {
24         cout << "\n\n> Removido um '15' da lista com sucesso!\n\n";
25     }
26     L.mostrarSimples("> Nova lista");
27
28     cout << endl << endl;
29 }
```

```

> Lista original: 10, 15, 15, 20, 30, 40, 15, 50, 15, 15
> Removido um '15' da lista com sucesso!
> Nova lista: 10, 15, 20, 30, 40, 15, 50, 15, 15

Process returned 0 (0x0)   execution time : 0.033 s
Press any key to continue.
```

*Mesma lista do exemplo anterior, mas utilizando **removeUmVal**. Notou a diferença?*



**CONFIRA** o exemplo dos **métodos *removeVariosVal* e *removeUmVal*** para **structs** com overloading (com uma lista de CARROS)!!!

# Busca de dados

## existeVal(*dado*)\*\*\*

- Verifica se existe um *dado de parâmetro* em um nó da lista. **Retorna um valor booleano: *true*** se foi encontrado, pelo menos, uma vez, e ***false*** caso contrário.

## existeNo(*dado*)\*\*\*

- Também verifica existência de *dado de parâmetro* em um nó da lista, mas **retorna um ponteiro**: se foi encontrado, **o ponteiro é o NÓ encontrado**; caso contrário, **o ponteiro é NULO!**

## qtdD(*dado*)\*\*\*

- **Retorna um *int***, que é a **quantidade de ocorrências** do *dado de parâmetro* em nós dentro da lista.

```
4
5 main() {
6     setlocale(LC_ALL, "Portuguese");
7
8     ListaD<string> L;
9     L.adFim("Ricardo");
10    L.adFim("Augusto");
11    L.adFim("Karmine");
12    L.adFim("Geremia");
13    L.adFim("Augusto");
14    L.adFim("Karmine");
15
16    L.mostrar("Lista de nomes", "Nomes", 0, 'T');
17    cout << "*****\n";
18    // Existe valor?
19    if(L.existeVal("Ricardo")) {
20        cout << "Existe 'Ricardo' na lista!!!";
21    } if(L.existeVal("Carmin")) {
22        cout << "Existe 'Carmin' na lista!!!";
23    }
24
25    // Existe nó?
26    cout << "\nNó de Augusto: " << L.existeNo("Augusto");
27    cout << "\nNó de Germani: " << L.existeNo("Germani");
28
29    // Quantos dados?
30    cout << "\n'Karmine' aparece " << L.qtdD("Karmine") << " vezes!!";
31
32    cout << endl << endl;
33 }
34
```

```
"C:\Users\PC\Documents\Estrutura de Dados\Exemplos M\Todos Descritor\exemploC.exe"
>> Lista de nomes

** Endereço do início: 0x2482eb0
** Endereço do fim: 0x2482ff0
** Quantidade de elementos: 6

Endereço | Nomes
-----|-----
0x2482eb0 | Ricardo
0x2482ef0 | Augusto
0x2482f30 | Karmine
0x2482f70 | Geremia
0x2482fb0 | Augusto
0x2482ff0 | Karmine
*****
Existe 'Ricardo' na lista!!!
Nó de Augusto: 0x2482ef0
Nó de Germani: 0
'Karmine' aparece 2 vezes!!

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

*Exemplos dos métodos de busca em uma lista de strings. Compreendeu cada linha?*



CONFIRA o exemplo dos **métodos de busca em *struct*** com overloading (usando lista de COMIDAS)!!!



## Operações com listas

## unirLista(*lista*)

- Uma nova lista é criada, correspondente à junção dos dados da lista atual com os dados da **lista do parâmetro**. A nova lista é retornada pelo método!

### intersectLista(**lista**)\*\*\*

- Uma nova lista é criada, onde dados que são COMUNS entre a lista atual e a **lista do parâmetro** são inseridos. O método retorna a nova lista!

```

5  main() {
6      setlocale(LC_ALL, "Portuguese");
7
8      ListaD<int> L1, L2;
9      // Primeira lista
10     L1.adFim(10);
11     L1.adFim(30);
12     L1.adFim(50);
13     L1.adFim(10);
14
15     // Segunda lista
16     L2.adFim(30);
17     L2.adFim(40);
18     L2.adFim(50);
19     L2.adFim(20);
20     L2.adFim(10);
21
22     ListaD<int> Lu1 = L1.unirLista(L2),
23     Lu2 = L2.unirLista(L1),
24     Li = L1.intersectLista(L2);
25
26     Lu1.mostrarSimples("> Lista UNIDA (L1 + L2)");
27     Lu2.mostrarSimples("\n\n> Lista UNIDA (L2 + L1)");
28     Li.mostrarSimples("\n\n> Lista INTERSECTADA");
29
30     cout << endl << endl;

```

```

"C:\Users\PC\Documents\Estrutura de Dados I\Exemplos M\Todos Descritor\exemplo8.exe"
> Lista UNIDA (L1 + L2): 10, 30, 50, 10, 30, 40, 50, 20, 10
> Lista UNIDA (L2 + L1): 30, 40, 50, 20, 10, 10, 30, 50, 10
> Lista INTERSECTADA: 10, 30, 50

Process returned 0 (0x0)   execution time : 0.033 s
Press any key to continue.

```

Exemplos de **operações com duas listas** de inteiros. Nota-se que A **ORDEM É IMPORTANTE!**

## Outros métodos

### mostrar(**nome**, **coluna**, **qtd**, **char**)\*\*\*

- Método mais completo para exibir os dados de uma lista. Primeiro, é mostrado o **nome** da lista, e abaixo, dados são exibidos em formato de tabela, sempre com uma coluna padrão (a do endereço dos nós). Um valor do tipo **char** serve para indicar O QUE será mostrado:
  - **T**: Mostrar TODOS os dados
  - **I**: Mostrar apenas dado do INÍCIO
  - **F**: Mostrar apenas dado do FIM
- Um **parâmetro inteiro** também deve ser inserido, podendo assumir dois valores: **2** se dados mostrados são *structs*; **0** caso contrário!



```

4
5 main() {
6     setlocale(LC_ALL, "Portuguese");
7
8     ListaD<string> Ls;
9     ListaS<int> Li;
10
11     Ls.adFim("Ricardo");
12     Ls.adFim("Augusto");
13     Ls.adFim("Karmine");
14     Ls.adFim("Geremia");
15
16     Li.adInicio(30);
17     Li.adInicio(20);
18     Li.adInicio(10);
19
20     Ls.mostrar("Lista de nomes", "Nome", 0, 'T');
21     cout << "*****" << endl;
22     Ls.mostrar("Primeiro nome", "Nome", 0, 'I');
23     cout << "*****" << endl;
24     Li.mostrar("Lista de inteiros", "Número", 0, 'T');
25     cout << "*****" << endl;
26     Li.mostrar("Último número", "Número", 0, 'F');
27
28     cout << endl << endl;
29 }
30

```

```

"C:\Users\PC\Documents\Estrutura de Dados\Exemplos MÚltos De
>> Lista de nomes
** Endereço do início: 0xe02eb0
** Endereço do fim: 0xe02f70
** Quantidade de elementos: 4

Endereço | Nome
-----|-----
0xe02eb0 | Ricardo
0xe02ef0 | Augusto
0xe02f30 | Karmine
0xe02f70 | Geremia
*****
>> Primeiro nome
Endereço | Nome
-----|-----
0xe02eb0 | Ricardo
*****
>> Lista de inteiros
** Endereço do início: 0xe02d90
** Endereço do fim: 0xe02d70
** Quantidade de elementos: 3

Endereço | Número
-----|-----
0xe02d90 | 10
0xe02cb0 | 20
0xe02d70 | 30
*****
>> Último número
Endereço | Número
-----|-----
0xe02d70 | 30

```

Exemplos das funcionalidades do método para listas de string e número!



O método **mostrar** para **structs** está em **TODOS** os arquivos de exemplos. Confira com muita atenção cada exemplo e como aplicá-los!!

### mostrarDesdeFim(*nome*, *coluna*, *qtd*)\*\*\*

- Método **EXCLUSIVO** para listas **DUPLAMENTE** encadeadas! Pela possibilidade de dois sentidos de “passeio” na lista, é possível um método que mostre desde o **ÚLTIMO** elemento da lista até o **PRIMEIRO**. Tem quase todos os parâmetros da função **mostrar**, com exceção do último.

```

4
5 main() {
6     setlocale(LC_ALL, "Portuguese");
7
8     ListaD<int> L;
9     L.adFim(10);
10    L.adFim(20);
11    L.adFim(30);
12    L.adFim(40);
13    L.adFim(50);
14
15    L.mostrarDesdeFim("Lista de números", "Número", 0);
16    cout << endl << endl;
17 }
18

```

```

"C:\Users\PC\Documents\Estrutura de Dados\Exemplos MÚlt
>> Lista de números
** Endereço do início: 0x632b90
** Endereço do fim: 0x632d50
** Quantidade de elementos: 5

Endereço | Número
-----|-----
0x632d50 | 50
0x632d90 | 40
0x632c70 | 30
0x632bb0 | 20
0x632b90 | 10

```

Exemplo de lista de números sendo mostrada do fim para o início. E se usássemos a função **mostrar** padrão?

### mostrarSimple(*nome*)

- Versão simplificada da função **mostrar** acima. Aqui, o **nome** é sucedido por dois pontos e uma sequência de **TODOS** os dados da lista, separados por vírgulas. Confira sua aplicação em imagens anteriores!

## destruir()

- Deleta TODOS os nós inseridos e *reseta* os parâmetros básicos do descritor (atributos *inicio* e *fim* voltam como **nullptr** e *tam* é zerado).

```
5 main() {
6     setlocale(LC_ALL, "Portuguese");
7
8     ListaD<string> Ls;
9
10    Ls.adFim("Ricardo");
11    Ls.adFim("Augusto");
12    Ls.adFim("Karmine");
13    Ls.adFim("Geremia");
14
15    Ls.mostrarSimples("> Nomes da lista");
16    cout << "\n> Tamanho atual: " << Ls.tam;
17    Ls.destruir();
18    Ls.mostrarSimples("\n\n> Lista destruída");
19    cout << "\n> Tamanho atual: " << Ls.tam;
20
21    cout << endl << endl;
22
23 }
```

```
"C:\Users\PC\Documents\Estrutura de Dados\Exemplos Múltiplos Descritor\exemploC.exe"
> Nomes da lista: Ricardo, Augusto, Karmine, Geremia
> Tamanho atual: 4

> Lista destruída: VAZIA
> Tamanho atual: 0

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

*Exemplo de lista de strings sendo destruída completamente!*

## copiaLista()

- Cria uma nova lista, com novos nós (ou seja, novos endereços sendo alocados), mas com os mesmos dados da original na mesma ordem. **A função retorna a LISTA CÓPIA!!**

```
4
5 main() {
6     setlocale(LC_ALL, "Portuguese");
7
8     ListaD<string> L;
9     L.adFim("Ricardo");
10    L.adFim("Karmine");
11
12    L.mostrar("Lista ORIGINAL", "  Nomes", 0, 'T');
13
14    // CÓPIA DA LISTA
15    cout << endl;
16    ListaD<string> Lcop = L.copiaLista();
17    Lcop.mostrar("Lista COPIADA", "  Nomes", 0, 'T');
18
19    cout << endl << endl;
20
21 }
```

```
"C:\Users\PC\Documents\Estrutura de Dados\Exemplos Múltiplos Descritor\exemploC.exe"
>> Lista ORIGINAL

** Endereço do início: 0x2532ef0
** Endereço do fim: 0x2532f30
** Quantidade de elementos: 2

Endereço | Nomes
-----|-----
0x2532ef0 | Ricardo
0x2532f30 | Karmine

>> Lista COPIADA

** Endereço do início: 0x2532f70
** Endereço do fim: 0x2532fb0
** Quantidade de elementos: 2

Endereço | Nomes
-----|-----
0x2532f70 | Ricardo
0x2532fb0 | Karmine
```

*Exemplo de lista de strings sendo copiada. **Notou a diferença** nos endereços dos nós???*



## ordenar(*funcao*)

- Método que, com uma *função booleana de parâmetro*, permite a ordenação dos **dados**. Dependendo de como a função de parâmetro é montada, podemos ordenar tanto de *baixo para cima* quanto de *cima para baixo*.



```

4
5 bool crescente(string S1, string S2) {
6     return (S1 < S2);
7 }
8
9 main() {
10     setlocale(LC_ALL, "Portuguese");
11
12     ListaD<string> Ls;
13
14     Ls.adFim("Ricardo");
15     Ls.adFim("Augusto");
16     Ls.adFim("Karmine");
17     Ls.adFim("Geremia");
18
19     Ls.mostrarSimples("> Nomes (original)");
20     Ls.ordenar(crescente);
21     Ls.mostrarSimples("\n\n> Nomes (ordenada)");
22
23     cout << endl << endl;
24 }
25

```

```

"C:\Users\PC\Documents\Estrutura de Dados\Exemplos Múltiplos Descritores\exemploC.exe"
> Nomes (original): Ricardo, Augusto, Karmine, Geremia
> Nomes (ordenada): Augusto, Geremia, Karmine, Ricardo

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.

```

*Lista de strings sendo reorganizada em ordem ALFABÉTICA. Nota a função booleana crescente criada no main, utilizada como parâmetro no método ordenar!!!*

```

4
5 bool decrescente(int N1, int N2) {
6     return (N1 > N2);
7 }
8
9 main() {
10     setlocale(LC_ALL, "Portuguese");
11
12     ListaS<int> L;
13
14     L.adFim(10);
15     L.adInicio(20);
16     L.adFim(30);
17     L.adInicio(40);
18     L.adFim(50);
19     L.adFim(60);
20
21     L.mostrarSimples("> Números (original)");
22     L.ordenar(decrescente);
23     L.mostrarSimples("\n\n> Números (ordenada)");
24
25     cout << endl << endl;
26 }
27

```

```

"C:\Users\PC\Documents\Estrutura de Dados\Exemplos Múltiplos Descritores\exemploC.exe"
> Números (original): 40, 20, 10, 30, 50, 60
> Números (ordenada): 60, 50, 40, 30, 20, 10

Process returned 0 (0x0)   execution time : 0.036 s
Press any key to continue.

```

*Lista de números inteiros sendo reorganizada em ordem DECRESCENTE. Notou alguma diferença na função booleana usada para a ordenação?*



**IMPORTANTE** conferir os exemplos dos métodos *copiaLista* e *ordenar* para *structs* com overloading (novamente com lista de PESSOAS)!!!

**Observação IMPORTANTE:** Se métodos de uma lista são usados DENTRO de uma função FORA do main, devemos passar a lista como PONTEIRO para esta função. Portanto, acessamos os métodos usando seta (->) - O MESMO se aplica aos **atributos das listas**, que são *inicio*, *fim* e *tam*. Confira abaixo as diferenças:

- Método de *adFim* acessado **DIRETO no main** (usando pontos)

```

5 main() {
6     setlocale(LC_ALL, "Portuguese");
7
8     ListaD<int> L;
9     L.adFim(10);
10    L.adFim(20);
11    L.adFim(30);
12    L.adFim(40);
13    L.adFim(50);
14
15    L.mostrar("Lista de números", " Número ", 0, 'T');
16    cout << endl << endl;
17 }
18

```

```

"C:\Users\PC\Documents\Estrutura de Dados \Exemplos MÚltodos
>> Lista de números

** Endereço do início: 0x632d10
** Endereço do fim: 0x632c30
** Quantidade de elementos: 5

Endereço | Número
-----|-----
0x632d10 | 10
0x632df0 | 20
0x632db0 | 30
0x632d50 | 40
0x632c30 | 50

```

- Método de *adFim* acessado **DENTRO** de uma função *void* de adicionar (usando seta)

```

5 void adicionar(ListaD<int>* Lp, int dado) {
6     Lp->adFim(dado);
7     cout << "> " << dado << " adicionado com sucesso!!\n";
8 }
9
10 main() {
11     setlocale(LC_ALL, "Portuguese");
12
13     ListaD<int> L;
14
15     adicionar(&L, 10);
16     adicionar(&L, 20);
17     adicionar(&L, 30);
18     adicionar(&L, 40);
19     adicionar(&L, 50);
20
21     cout << endl;
22     L.mostrar("Lista de números", " Número ", 0, 'T');
23     cout << endl << endl;
24 }

```

```

"C:\Users\PC\Documents\Estrutura de Dados \Exemplos MÚltodos
> 10 adicionado com sucesso!!
> 20 adicionado com sucesso!!
> 30 adicionado com sucesso!!
> 40 adicionado com sucesso!!
> 50 adicionado com sucesso!!

>> Lista de números

** Endereço do início: 0x2542dd0
** Endereço do fim: 0x2542e10
** Quantidade de elementos: 5

Endereço | Número
-----|-----
0x2542dd0 | 10
0x2542cd0 | 20
0x2542bd0 | 30
0x2542c10 | 40
0x2542e10 | 50

```

- Exemplo de **nós de lista SIMPLEMENTE encadeada** sendo **acessados DENTRO de uma função void** e aplicadas em um **laço WHILE**. Confira o comentário sobre os TIPOS de dados para as variáveis dos nós!!

```

5 void mostrarEnderecos(ListaS<int>* Lp) {
6
7     int C = 0;
8
9     // Redeção: No<tipo>* --> lista simplesmente encadeada
10    // NoD<tipo>* --> lista duplamente encadeada
11
12    No<int>* N = Lp->inicio; // ACESSANDO NÓ DE LISTA
13    while (N != NULL) {
14        C++;
15        cout << "Endereço #" << C << ": " << N << endl;
16        N = N->prox;
17    }
18 }
19
20 main() {
21     setlocale(LC_ALL, "Portuguese");
22
23     ListaS<int> L;
24     L.adInicio(30);
25     L.adInicio(20);
26     L.adInicio(10);
27
28     mostrarEnderecos(&L);
29
30     cout << endl << endl;
31 }

```

```

"C:\Users\PC\Documents\Estrutura de Dados \Exemplos MÚltodos Descritor\Exemplos Structs\exemploC.exe"
Endereço #1: 0x632d70
Endereço #2: 0x632c50
Endereço #3: 0x632bd0

Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.

```