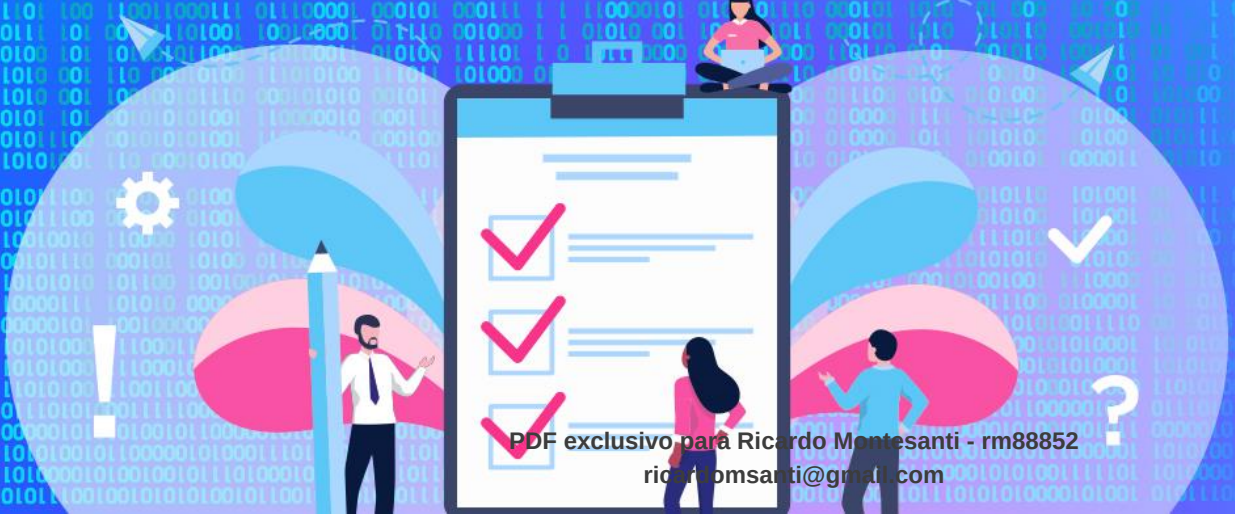
CÓDIGOS DE ALTA PERFORMANCE

LISTAS LINEARES, PILHAS E FILAS

PATRÍCIA MAGNA



LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 LISTAS LINEARES, PILHAS E FILAS

É muito comum termos uma impressora compartilhada com vários usuários, quando mandamos um arquivo para ser impresso pode ser que tenha mais de um usuário fazendo a mesma coisa. Você já pensou em como elaborar um programa que controle o uso da impressora? Como garantir que se você mandou seu arquivo primeiro você seja o primeiro a ter seu arquivo impresso? Como guardar as informações dos usuários que enviaram arquivos para impressão enquanto o seu arquivo era impresso?

Neste capítulo, vamos abordar esse problema e criar tipos de dados a fim de armazenar uma coleção de elementos, e além disso, garantir algum tipo de ordenação, o que chamaremos de listas lineares. Antes, porém, temos que apresentar conceitos e formalidades necessários ao entendimento.

1.1 Representação de informação em sistemas computacionais

Essencialmente, nessa fase será realizado o estudo sobre estruturas de dados, que consiste em aprender e desenvolver algoritmos que manipulam dados, a fim de processá-los e armazená-los em sistemas computacionais digitais (computadores).

É necessário, inicialmente, entender como as informações do mundo real podem ser processadas e armazenadas em computadores digitais.

Para tanto, usaremos como exemplo de aplicação, a previsão de tempo realizada computacionalmente. Em uma primeira etapa, deve-se partir dos cálculos, que podem ser efetuados sem o uso de computadores, fazer a modelagem do problema para gerar algoritmo que possa ser implementado em alguma linguagem de programação (por exemplo, JAVA). Depois, devem ser obtidas medidas do mundo real, tais como pressão, temperatura, umidade do ar, direção de vento predominante etc., em vários locais, para que esses dados sejam usados no processamento a fim de obter informações sobre a probabilidade de chuva nas várias horas do dia.

Assim, dizemos que dados são fornecidos para o processamento e os resultados obtidos do processamento são informações como mostra a figura: Esquema de entrada de dados, processamento e saída de resultados. Claro que

informações geradas por um processamento podem se tornar dados para outro processamento.

Figura 1.1 – Esquema de entrada de dados, processamento e saída de resultados.
Fonte: Elaborado pelo autor (2018).

Quando vamos inserir dados em um computador para o processamento de alguma aplicação, devemos escolher um tipo de dado mais adequado para representá-lo. Por exemplo, a temperatura na aplicação de previsão do tempo não poderia ser do tipo texto, mas sim do tipo número real, para poderem ser realizados cálculos aritméticos sobre esses dados.

Assim, um dos pontos principais na modelagem de uma aplicação para o processamento em um computador é definir como os dados vão ser representados no programa, sempre tendo como objetivo que o processamento seja realizado da melhor forma possível (o que depende dos interesses sobre a aplicação).

O objetivo do estudo de estruturas de dados é fornecer novas formas de tratar dados visando ampliar a gama dos tipos de aplicações que já foram estudadas até este momento do curso.

1.1.1 Tipos de dados

Uma pergunta deve ser feita: como os dados e informações são representados dentro do computador? A resposta é: usando algum tipo de dado definido pela arquitetura do computador, ou melhor, do processador.

Mas, o que é um tipo de dado?

São valores que podem ser assumidos. Operações que podem ser efetuadas.

Por exemplo, tipo de dado inteiro:

- Valores que representam quantidades contáveis de objetos.

- Operações: soma, subtração, multiplicação, divisão e resto da divisão.

Alguns tipos de dados são compreendidos diretamente pelo processador, ou seja, na arquitetura interna do processador estão presentes unidades de aritmética que realizam as operações sobre esses tipos de dados. Esses tipos são ditos **dados primitivos**, os quais normalmente são: *números inteiros com ou sem sinal, reais e caracteres*.

Para poder facilitar o uso de dados por nossos programas, são também definidos os tipos **compostos de dados** ou também conhecidos como **dados estruturados** que são compostos internamente por vários **dados primitivos**, ou não. Como exemplos desses tipos de dados, já estudados, temos: *vetores, matrizes e registros (dados de tipos diferentes são agrupados em um dado do tipo registro).*

1.1.2 Tipos abstratos de dados versus tipos "concretos" de dados

No momento de modelar um problema (elaborar o algoritmo) é apenas necessário definir quais serão os tipos de dados que cada variável deverá ser declarada, mesmo sem saber com qual linguagem o algoritmo será implementado. Não se preocupar com qual tipo de dado será usado no momento da tradução para a linguagem de alto nível escolhida torna mais ampla a escolha, pois se baseia na vida real e não nas limitações da vida digital.

Por exemplo, suponha que se deseje armazenar em uma variável o número de veículos que passam por uma movimentada estrada durante o ano. No algoritmo é escolhido o tipo de dado número inteiro. Mas, o tipo de dado inteiro, quando definido pelo computador, apresenta limitações em relação ao tipo de dado inteiro definido no mundo real, isso por que é necessário ter uma quantidade máxima de dígitos binários (bits) para representar valores. Tendo em vista que trabalhamos com computadores digitais os valores devem todos ser representados na forma binária, assim, por exemplo, o número 20 na base 10 deve ser representado como 10100 ($1x2^4+0x2^3+1x2^2+0x2^1+0x2^0$) na base 2. Se um tipo de dado inteiro pudesse ter apenas 4 bits (dígitos binários) o número 20 não poderia ser representado.

Devem-se distinguir tipos abstratos de dados e tipos concretos de dados.

Os tipos **abstratos de dados** (TAD) especificam as propriedades lógicas e matemáticas de um tipo de dados ou estrutura, tornando-se guias úteis para os programadores. Desta forma, um TAD não precisa levar em consideração como será o tipo de **dado concreto** que será usado no computador. Já os tipos concretos de dados são aqueles que são escolhidos dentro das opções de tipos de dados oferecidos pela arquitetura do computador, ou seja, dentro das limitações que são inerentes do mundo digital.

Na definição de um TAD, não é usual a preocupação com tempo, eficiência de processamento ou limitações de espaço, que são problemas de implementação. Não vamos estudar todas as técnicas e notações de TADs, mas sim o uso de alguns destas em problemas de programação com a finalidade de elaborarmos códigos que usem eficientemente os dados no processamento de aplicações.

1.2 Organizando dados em estruturas de listas lineares

1.2.1 Entendendo o que são listas lineares

No nosso dia a dia, temos a necessidade de agrupar dados que tenham alguma relação e que precisamos lembrar. Normalmente, criamos listas como, por exemplo:

- Lista de compras para irmos ao supermercado.
- Lista de tarefas a serem realizadas no trabalho.
- Lista de convidados para uma festa.

Em desenvolvimento de sistemas de computação também precisamos de listas. Exemplos:

- Lista de arquivos a serem impressos.
- Lista de solicitações de acesso para consulta de servidor de um banco de dados.

Contudo, analisando as aplicações que usam listas em computação podemos observar que não basta criar um conjunto de elementos, mas precisamos definir uma ordem a fim de sermos justos com os usuários que solicitaram algum serviço. Assim:

- Lista de arquivos a serem impressos: se vários usuários solicitam a impressão de arquivos, é justo que o primeiro que solicitou seja o primeiro a ser impresso.
- Lista de solicitações de acesso para consulta de um banco de dados: se vários clientes de um banco solicitam o saldo, o atendimento deve atender o primeiro a pedir.

Portanto, fica claro que devemos criar listas que possamos garantir algum tipo de ordenação. Este tipo de lista é conhecido como lista linear. Então, vamos primeiro defini-la formalmente e, em seguida vamos aprender como criar e utilizar esse tipo de lista.

DEFINIÇÃO: uma lista linear é uma estrutura de dados que além de armazenar vários valores de elementos, impõe que a posição de cada elemento deve respeitar algum tipo de ordem.

Por exemplo, em uma lista de arquivos que estão na fila de impressão de uma impressora compartilhada por vários usuários, deve ser respeitada a ordem cronológica de chegada de cada arquivo. Portanto, uma lista deve não apenas conter as informações de cada arquivo, mas também oferecer uma ordenação, de forma que sempre se saiba quem foi o primeiro a chegar, o segundo e assim por diante.

Usando o formalismo adequado para o estudo de lista linear define-se que é uma estrutura dinâmica caracterizada por uma sequência ordenada de elementos, ordenada no sentido de sua posição relativa**: E_0 , E_1 , E_2 ,.... E_{n-1}**, tal que:

- Existem **n** elementos na sequência.
- **E_0** é o primeiro elemento da sequência.
- **E_{n-1}** é o último elemento da sequência.
- Para todo **i** e **j** entre **0** e **n-1**, se **i < j**, então **E_i** antecede **E_j**.
- Caso **i = j -1, E_i** é o antecessor de **E_j** e **E_j** é o sucessor de **E_i**.

A característica principal de uma lista linear é o sentido de ordem unidirecional dos elementos que a compõem. O critério usado para essa ordenação é bastante genérico, podendo ser definido em função do problema que está sendo modelado.

Dentre as diversas operações que podem ser realizadas com listas, temos:

- Ter acesso a um elemento qualquer da lista (acesso).
- Inserir um elemento em uma posição específica da lista (inserção).
- Remover um elemento em uma posição específica da lista (remoção).
- Combinar duas listas em apenas uma.
- Particionar (dividir) uma lista em duas listas.
- Determinar o total de elementos.

Considerando apenas as operações de acesso, inserção e remoção, podemos definir casos especiais que são, com muita frequência, encontrados na modelagem de problemas a serem resolvidos por computador. Esses casos especiais geram as seguintes listas lineares especiais: pilha e fila. A seguir uma breve descrição de cada uma delas.

1.2.1 Lista Linear Especial: Pilha

Uma pilha (*stack*) é uma lista linear em que as operações de inserção e de remoção são efetuadas apenas em uma extremidade, denominada topo da pilha. Estruturas deste tipo são conhecidas como LIFO (*last in first out* – último a entrar primeiro a sair).

A figura a seguir mostra uma representação de pilha.

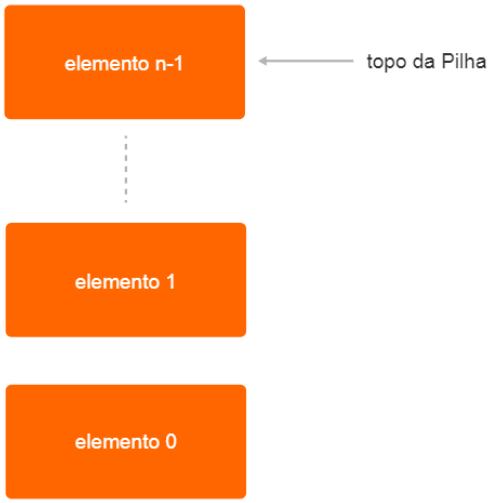


Figura 1.2 – Estrutura de uma lista linear do tipo PILHA.
Fonte: Elaborado pelo autor (2018).

Pilhas são usadas em aplicações onde é necessário que a ordem de saída dos elementos seja inversa da ordem de entrada.

Uma analogia, com um exemplo real, seria uma pilha de livros muito pesados. Cada livro que deve ser empilhado (ser colocado sobre os outros), ou seja, colocado no topo da pilha. E para desempilhar, como são muito pesados, deve ser retirado sempre o livro que estiver no topo da pilha de livros.

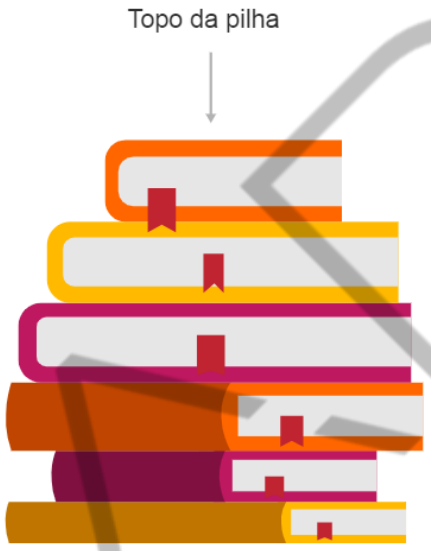


Figura 1.3 – Exemplo de PILHA de livros.
Fonte: Elaborada pelo autor (2018).

1.2.2 Lista linear especial: fila

Uma fila (*queue*) é uma lista linear na qual a operação de inserção é feita em uma extremidade denominada final da fila e remoção é efetuada apenas na outra extremidade denominada início. Estruturas deste tipo são conhecidas como FIFO (*first in first out* – primeiro a entrar primeiro a sair).

A figura: Estrutura de uma lista linear do tipo FILA, abaixo, mostra uma representação de fila.

Figura 1.4 – Estrutura de uma lista linear do tipo FILA.
Fonte: Elaborado pelo autor (2018).

Uma analogia bastante óbvia é a de uma fila de clientes em banco – novos clientes entram no final de fila e o caixa sempre retira a pessoa que está no início.

Figura 1.5 – Exemplo de fila de pessoas em um banco.
Fonte: Adaptado de Google Images (2018)

1.2.3 Como construir listas lineares em programas?

Há duas formas de se implementar listas lineares:

- **Implementação estática:** é usado vetor para armazenar os elementos da lista, por exemplo, armazenar como elementos do vetor os nomes das pessoas que entraram na fila do banco, como mostra a figura a seguir. Por usar vetor a ordenacao é garantida pela posição do elemento dentro do vetor.

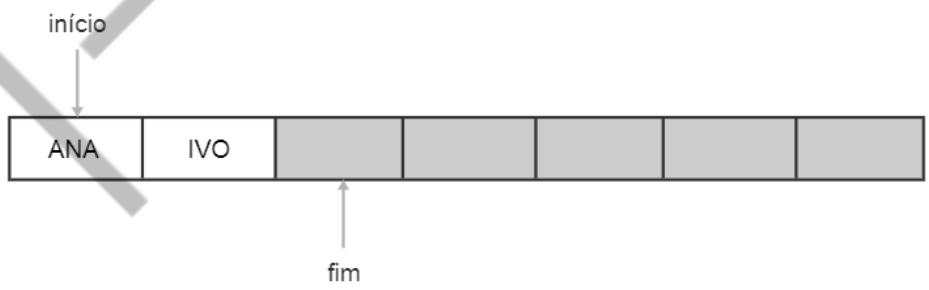


Figura 1.6 – Estrutura de uma FILA usando vetor para implementação estática.
Fonte: Elaborado pelo autor (2018).

- **Implementação dinâmica:** deve-se criar espaço na memória para armazenar cada elemento da lista, conforme a aplicação, isso requer mais um elemento, é chamado de alocação dinâmica, aloca (reserva local) espaço na memória para um dado enquanto o programa está sendo executada (dinamicamente).

Nessa fase, faremos apenas a implementação dinâmica, pois com o uso do vetor para criar listas deve-se restringir o número de elementos, uma vez que na declaração sempre temos que determinar o número máximo de elementos. Além disso, quando se deseja inserir ou remover um elemento em uma posição no meio da lista, são necessárias muitas operações de deslocamento de elementos dentro do vetor, algo que tem custo alto no tempo de execução das aplicações.

Então, vamos seguir para a próxima seção e aprender como criar e tratar listas lineares dinâmicas.

1.4 Lista linear encadeada

Usando o conceito de lista linear que diz ser uma estrutura dinâmica caracterizada por uma sequência ordenada de elementos, quando elementos são alocados na memória principal, o que se pode obter pode ser algo como mostra a figura: Exemplo de ocupação da memória para alocar elementos de uma lista. Pode-se observar que os elementos estão fora de ordem e distribuídos de forma esparsa (espalhada).

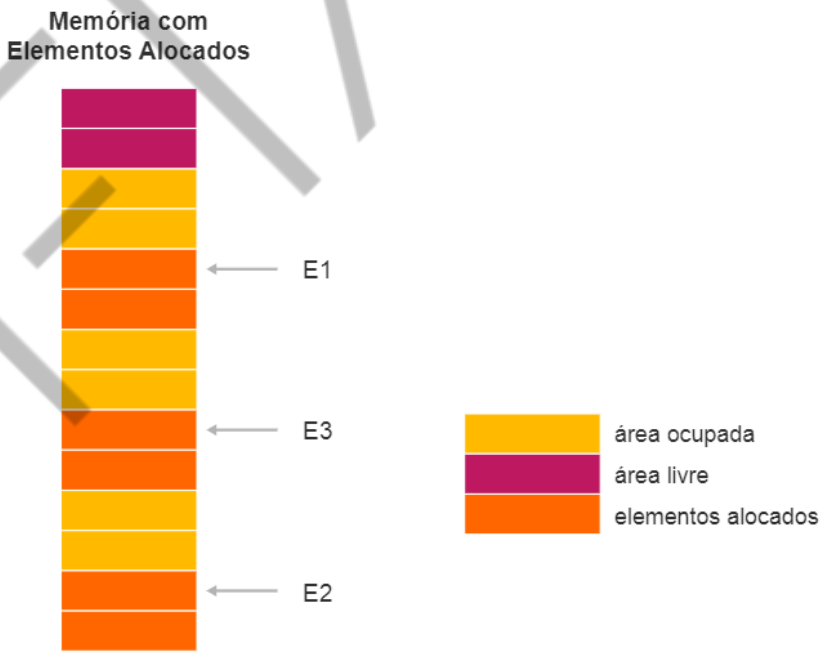


Figura 1.7 – Exemplo de ocupação da memória para alocar elementos de uma lista.
Fonte: Elaborado pelo autor (2018).

Portanto, para que se transforme em uma lista linear dinâmica, é necessário que cada elemento tenha a indicação (ponteiro) de onde se encontra o elemento que

é o seu sucessor. A razão disso é que é preciso manter a ordem em que os elementos foram inseridos e, também, a localização de cada um deles.

Para que isto possa acontecer, cada elemento deve se tornar:

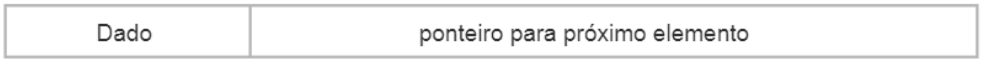


Figura 1.8 – Esquema de um nó que armazena um elemento de uma lista linear encadeada.
Fonte: Elaborado pelo autor (2018).

Como agora cada elemento da lista além do dado sempre terá a informação de localização do elemento sucessor, esse elemento passa a ser chamado de **nó** (*node* em inglês).

Para o exemplo da sequência de alocação de elemento, tem-se:

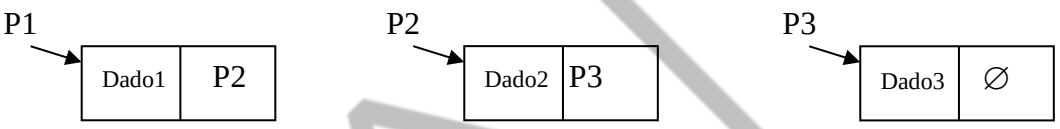


Figura 1.9 – Esquema de alocação de nós de uma lista linear encadeada.
Fonte: Elaborado pelo autor (2018).

Uma forma diferente de representar a lista linear encadeada é feita sabendo que ponteiros são sempre representados como setas que apontam para uma área de memória. Neste caso, o ponteiro para o próximo de cada nó aponta para a área de memória que armazena um nó. A forma mais comum de representar o encadeamento de nós é apresentada na figura: Esquema de encadeamento de nós de uma lista linear encadeada.

Figura 1.10 – Esquema de encadeamento de nós de uma lista linear encadeada.
Fonte: Elaborado pelo autor (2018).

Observe que, o nó que tem dado Dado3 tem como ponteiro prox a indicação que não há nó sucessor, essa indicação em algoritmo representada como NULO e tem representação gráfica como apresentada na figura: Esquema de encadeamento de nós de uma lista linear encadeada.

Com esta nova forma de representar uma lista, como fica a declaração de cada nó?

```
NO
  Inicio
    dado: tipo_do_elemento da lista
    prox: ponteiro para NO
  Fim
```

Observe que o NO tem como campos:

- O dado que será do tipo de dado a ser armazenado na lista.
- O ponteiro **prox** que aponta para o nó sucessor. Assim, o campo **prox** é um ponteiro para outro nó, ou seja, para outro registro no. Esse tipo de declaração é dita recursiva.

Agora vamos aprender como programar listas lineares usando em JAVA.

1.4.1 Lista linear encadeada em java (tipo concreto de dado)

Para mostrar como é feita a declaração de um nó de uma lista encadeada em JAVA, será suposto que os elementos a serem armazenados sejam do tipo inteiro. O nó em JAVA precisa ser do tipo classe NO, pois deve ser composto pelo atributo dado e também pelo atributo prox (ponteiro que aponta para o próximo nó).

```
private static class No {
          public int dado;
          public No prox;
          }
```

Vamos iniciar o estudo de listas lineares com um programa exemplo muito pequeno, apresentado a seguir:

```
  public class Lista_Encadeada_Simples {

  //declaração do nó e dos atributos
     private  static class NO {
           public int dado;
           public NO prox;
           }

  public static void main(String[] args) {
   //inicia lista vazia atribuindo null ao ponteiro lista
           NO lista = null;
           System.out.println("Valor ponteiro lista= " + lista);

   // um nó é alocado e é inserido na lista
           NO novo = new NO();
```

```
            novo.dado=5;
            novo.prox = lista;
            lista = novo;

      System.out.println("Atributos do nó apontado por  lista= " +
lista.dado +" "+lista.prox);
       }
       }
```

Código-fonte 1.1 – Primeiro exemplo de construção de uma lista linear encadeada em JAVA
Fonte: Elaborado pelo autor (2018)

A variável lista declarado do tipo classe NO, quando o instanciamos a classe NO é retornada uma referência ao objeto NO, portanto, lista é considerada um ponteiro para NO. Em seguida, é atribuído à variável lista o valor null (constante 0). Representaremos essa atribuição conforme mostra a figura a seguir.

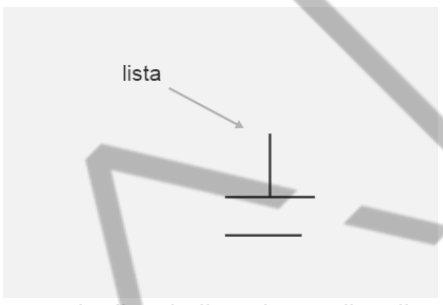


Figura 1.11 – Ponteiro lista indicando que lista linear está vazia.
Fonte: Elaborado pelo autor (2018).

Essa representação será usada quando tivermos uma lista vazia, ou seja, quando ponteiro que deve apontar para lista que não tem nenhum nó referenciado.

Para inserir um nó na lista linear o programa prossegue fazendo a alocação do nó usando o ponteiro novo que aponta para a NO alocado.

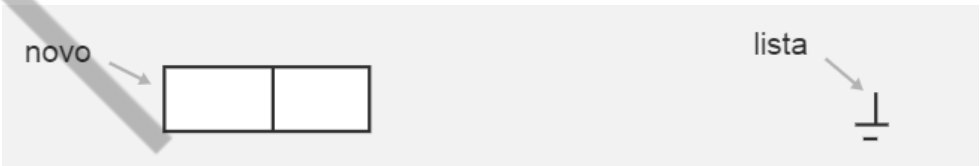


Figura 1.12 – Esquema de alocação de novo nó e ponteiro lista indicando que lista linear está vazia.
Fonte: Elaborado pelo autor (2018).

Realizando a execução do próximo comando, novo.dado = 5, é atribuído ao atributo dado do NO apontado pelo ponteiro novo o valor 5.

Finalmente, com o comando novo.prox = lista, o ponteiro prox recebe o ponteiro lista, quando isso acontece dizemos que prox aponta para o mesmo lugar que o ponteiro lista, que nesse caso aponta para null, representando assim que não há elemento sucessor. A figura: Esquema inserção de valores no novo nó alocado e

ponteiro lista indicando que lista linear está vazia apresenta uma representação desta etapa.

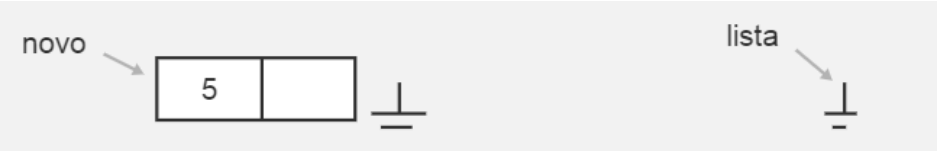


Figura 1.13 – Esquema inserção de valores no novo nó alocado e ponteiro lista indicando que lista linear está vazia.
Fonte: Elaborado pelo autor (2018).

O programa encerra com o ponteiro lista recebendo o ponteiro novo, e como acabamos de ver isso significa que o ponteiro lista deixa de apontar para null e passa apontar para o mesmo nó apontado por novo. Como mostra a figura: Esquema da lista com um nó alocado.

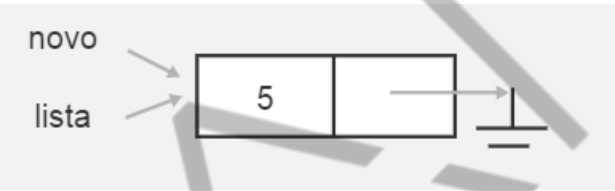


Figura 1.14 – Esquema da lista com um nó alocado.
Fonte: Elaborado pelo autor (2018).

Um exemplo com alocação de 2 nós em uma lista apresentado no código a seguir.

```
      public class Lista_Simples_2_nos {


     private static class NO {
            public int dado;
            public NO prox;
            }

      public static void main(String[] args) {

            NO lista = null;
            System.out.println("Valor ponteiro lista= " + lista);
            for(int i =1; i<=2;i++) {
                  NO novo = new NO();
                  novo.dado= i+4;
                  novo.prox = lista;
                  lista = novo;
            }

            System.out.println("Dado do NO apontado por lista=
"+lista.dado);
            System.out.println("Dado do NO apontado por prox "
+lista.prox.dado);
      }
```

```
    }
```

Código-fonte 1.2 – Exemplo de construção de uma lista linear encadeada com 2 nós em JAVA
Fonte: Elaborado pelo autor (2018)

Com a primeira iteração, a lista idêntica estaria como representada na figura: Esquema da lista com um nó alocado. Na segunda iteração, após criar um novo nó apontado por novo, é atribuído valor 6 como dado do nó, o ponteiro prox do novo nó passa a apontar para o mesmo nó que lista aponta. E, finalmente, lista passa a apontar para o mesmo nó apontado por novo. A figura a seguir, mostra passo a passo o que acontece para a inserção do segundo nó da lista linear encadeada.

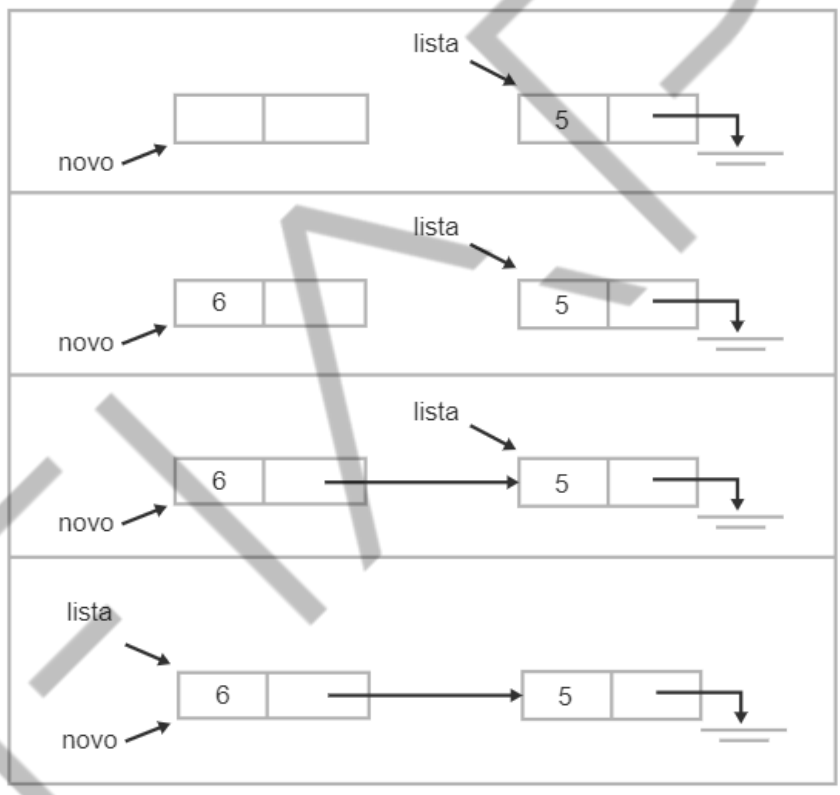


Figura 1.15 – Esquema da lista com segundo nó alocado e inserido.
Fonte: Elaborado pelo autor (2018).

O programa termina apresentando os dados armazenados em cada um dos nós. Primeiro é exibido na tela de saída lista.dado, ou seja, o valor do dado do nó apontado por lista (valor 6). Em seguida, é exibido lista.prox.dado, observe que lista.prox é o ponteiro que aponta para nó sucessor, isto é, é selecionado o nó que tem como dado o valor 5.

Para finalizar nosso primeiro contato com listas lineares encadeadas, vamos pensar em como apresentar os nós de uma lista quando ela possui mais do que 2

elementos. Para fazer isso, devemos entender que o ponteiro que estamos chamando de lista deve sempre apontar para o nó que, a partir dele, temos acesso a todo o encadeamento de nós. Portanto, o ponteiro lista não pode ser movido.

Então como percorrer uma lista inteira, com muitos nós, para apresentar cada um dos seus dados? Simples... basta que usemos um ponteiro auxiliar, que chamaremos aqui de aux. O ponteiro aux deve iniciar apontando para o nó apontado por lista e depois de exibir na tela de saída o dado, o ponteiro aux deve passar a apontar o nó sucessor (aux.prox). O trecho de programa a seguir apresenta a solução com o uso do ponteiro aux.

```
...

aux =lista;
while (aux!=null) {
     System.out.println("Dado do NO apontado por prox " +aux.dado);
      aux = aux.prox;
      }
```

Código-fonte 1.3 – Trecho de Programa para apresentação de todos os nós de uma lista linear encadeada em JAVA
Fonte: Elaborado pelo autor (2018)

Com esse conhecimento básico sobre listas lineares encadeadas, vamos praticar a implementação e a manipulação de listas construindo, também, listas encadeadas especiais: Pilha e Fila. Então, vamos seguir para essa nova etapa, pois exercitar é sempre muito bom e necessário.

EXERCÍCIOS DE FIXAÇÃO

1) Suponha a entrada da seguinte sequência de elementos 1,2,3,4,5 em uma lista linear especial. Qual será a ordem de retirada se a lista especial for uma:

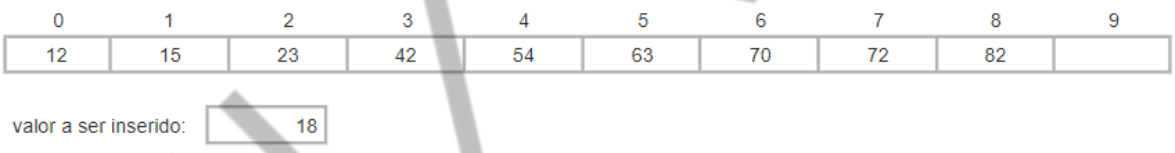
(a) Fila

1,2,3,4,5

(b) Pilha

5,4,3,2,1

2) Suponha um vetor organizado como lista linear com ordenação crescente de seus elementos e um novo valor a ser inserido, como mostra a figura a seguir. Explique o que deve acontecer para a inserção do valor 18 e elabore um pequeno algoritmo para realizar essa inserção.



0	1	2	3	4	5	6	7	8	9
12	15	23	42	54	63	70	72	82	

valor a ser inserido: 18

Resposta:

```
int valor = 18;
int i, j, n=9;

//procura posição para inserir i=0;
while ((valor > vetor[i]) && (i<n))
        i++;
//trecho identifica valor 18 deve ser inserido posição 2
//conta mais um elemento que será inserido
n= n+1;
/*deslocada cada elemento do vetor para uma posição a frente até posição
2*/
for (j=n; j>i; j--)
      vetor[j] = vetor[j-1];
//insere o valor 18 na posição
vetor[i] = valor;
```

3) Modifique o programa para que seja armazenado como dado de cada nó da lista um valor real (double).

```
public class Lista_Simples_2_nos {

      private  static class NO {
              public int dado;
              public NO prox;
              }

      public static void main(String[] args) {

            NO lista = null;
            System.out.println("Valor ponteiro lista= " + lista);
            for(int i =1; i<=2;i++) {
                  NO novo = new NO();
                  novo.dado= i+4;
                  novo.prox = lista;
                  lista = novo;
            }

            System.out.println("Dado do NO apontado por lista=
"+lista.dado);
            System.out.println("Dado do NO apontado por prox "
+lista.prox.dado);
      }
}
```

Resposta:

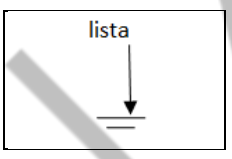
```
public class Lista_Simples_2_nos {

      private  static class NO {
              public double dado;
              public NO prox;
              }

      public static void main(String[] args) {

            NO lista = null;
            System.out.println("Valor ponteiro lista= " + lista);
            for(int i =1; i<=2;i++) {
                  NO novo = new NO();
                  novo.dado= i+4.5;
                  novo.prox = lista;
                  lista = novo;
            }

            System.out.println("Dado do NO apontado por lista=
"+lista.dado);
            System.out.println("Dado do NO apontado por prox "
+lista.prox.dado);
      }
}
```

Desenhe o esquema gráfico da lista gerada pelo programa:

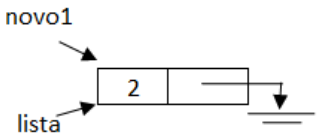
```
public class Lista_Simples_exercicio7 {

	privatestaticclass NO {
		public int dado;
		public NO prox;
		  }

	public static void main(String[] args) {

		NO lista = null;
		System.out.println("Valor ponteiro lista= " + lista);


	NO novo1 = new NO();
	novo1.dado= 2;
	novo1.prox = lista;
	lista = novo1;

	NO novo2 = new NO();
	novo2.dado= 5;
	novo2.prox = null;
	lista.prox = novo2;
		}
	}
}
```

Resposta:

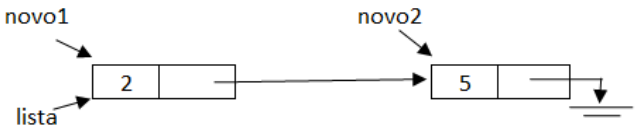
NO lista = **null**;



NO novo1 = **new** NO();
novo1.dado= 2;
novo1.prox = lista;
lista = novo1;



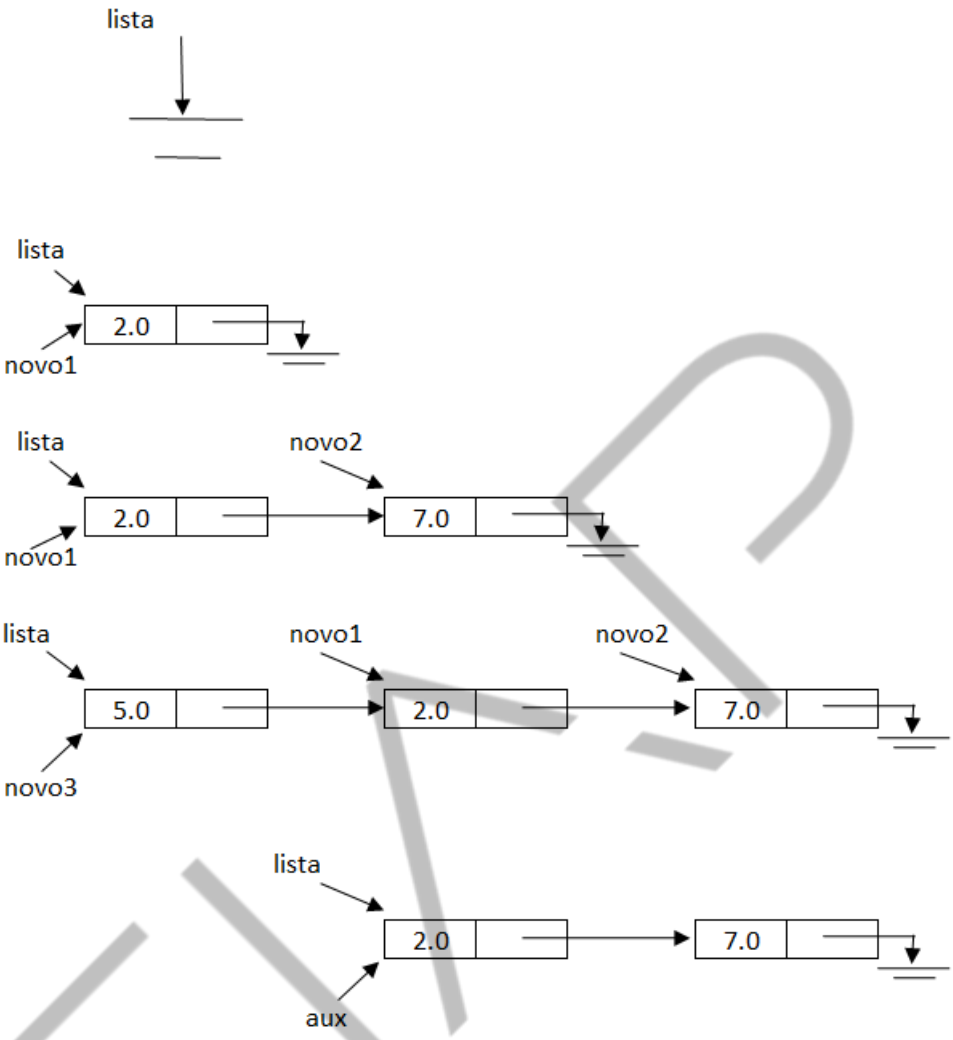
NO novo2 = **new** NO();
novo2.dado= 5;
novo2.prox = null;
lista.prox = novo2;



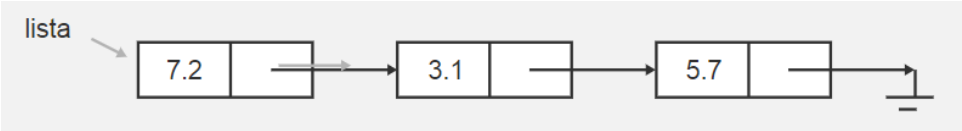
Desenhe o esquema gráfico da lista gerada pelo programa:

```
public class Lista_Simples_exercicio8 {


	private static class NO {
		public double dado;
		public NO prox;
	}
	public static void main(String[] args) {

		NO aux,aux2;
		NO lista = null;
		System.out.println("Valor ponteiro lista= " + lista);

		NO novo1 = new NO();
		novo1.dado= 2;
		novo1.prox = lista;
		lista = novo1;

		NO novo2 = new NO();
		novo2.dado= 7;
		novo2.prox = null;
		lista.prox = novo2;

		NO novo3 = new NO();
		novo3.dado= 5;
		novo3.prox = lista;
		lista= novo3;

		aux = lista;
		lista = aux.prox;
	}
}
```

Resposta:

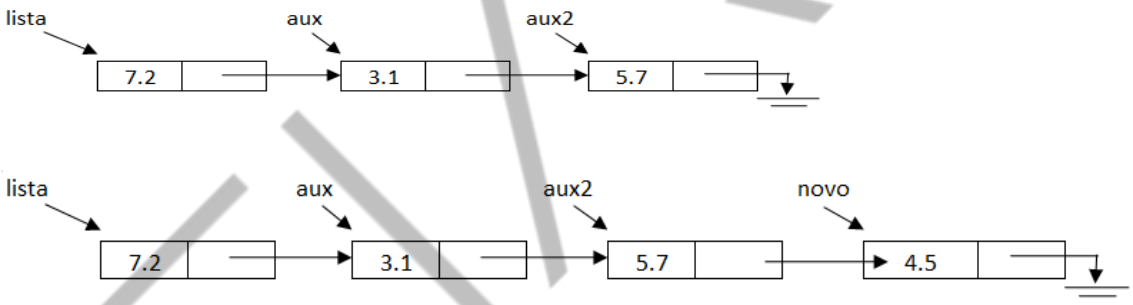


Suponha uma lista linear encadeada apresentada abaixo. Declare a classe NO Sabendo que lista**, aux** e **aux2** são declaradas como NO represente como ficaria a lista realizando as seguintes operações (os itens devem ocorrer na sequência em que são especificados).




```
aux = lista.prox;
aux2 = aux.prox;
lista.prox = aux2;
NO novo = new NO();
novo.dado = 4.5;
novo.prox = null;
aux2.prox = novo;
```

Resposta:



REFERÊNCIAS

ASCÊNCIO, A.F.G; ARAUJO, G.S. **Estruturas de Dados**: algoritmos, análise de complexidade e implementações em JAVA e C/C++. São Paulo: Pearson Prentice Hall, 2010.

PEREIRA, S.L.. **Estruturas de Dados Fundamentais**: conceitos e aplicações. São Paulo: Érica, 1996.

TENEMBAUM, A.M et al. **Estruturas de Dados usando C.** São Paulo: Makron Books, 1995.