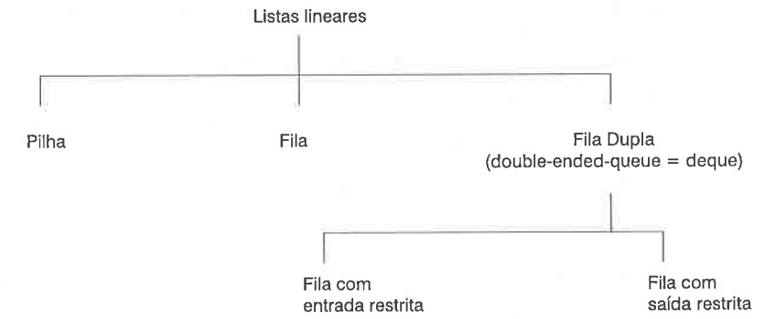
2

Listas lineares

Uma lista é uma forma de agrupar itens com a finalidade de melhorar sua manipulação. No nosso dia-a-dia, é mais do que comum utilizarmos as listas como forma de organizarmos certas tarefas. Alguns exemplos práticos: a lista de itens que devem ser comprados em um supermercado, a lista de convidados para um casamento, a lista de aviões que devem decolar no aeroporto, a lista de processos no estado pronto, de um certo sistema operacional, esperando pelo uso do processador ou, ainda, a lista de serviços de impressões (*spooled file*) aguardando pela liberação de uma impressora física.

É a propriedade seqüencial de uma lista linear, que é a base para sua definição e para o seu uso. Observamos que em uma lista linear existe um início, onde está o primeiro elemento; e um final, onde encontramos o último elemento. A disciplina de acesso, ou seja, a forma pela qual se realizam as operações de inserção e de remoção de elementos nas listas é o que determina algumas classificações das listas lineares. Na Figura 2.1, temos uma possível classificação das listas lineares e ilustramos, de forma geral, como serão apresentados os principais aspectos conceituais sobre essas listas ao longo deste texto.

Figura 2.1
Classificação das listas lineares.



Considerações iniciais

A lista linear é a estrutura que permite representar um conjunto de elementos, de forma a preservar a relação de ordem linear entre eles. Pode-se definir uma lista como um conjunto de n >= elementos (nós), $x_1, x_2, \ldots, x_n$, organizados de tal forma que sua estrutura reflete diretamente as posições relativas dos elementos que compõem a lista. O número de elementos de uma lista é denominado comprimento ou tamanho da lista.

Principal propriedade estrutural das listas

Observando-se a estrutura de uma lista, percebe-se uma grande propriedade estrutural: seus elementos podem ser ordenados de maneira linear de acordo com suas posições na lista.

Supondo n >= 0, onde n representa o número de nós, temos

- x_1: é o primeiro elemento da lista;
- x_n: é o último elemento da lista;
- e para cada i, $1 < i < n$, o elemento x_i é precedido por x_{i-1} e seguido de x_{i+1}.

Quando n = 0, dizemos que a lista é vazia.

Sobre uma lista, podem atuar diversas operações. A seguir, estão algumas das principais operações relacionadas às listas:

a. Criação de uma lista linear vazia;
b. Acesso ao i-ésimo elemento de uma lista para examiná-lo ou para alterá-lo;
c. Inserção de um novo elemento antes (ou depois) do i-ésimo elemento de uma lista linear;
d. Remoção do i-ésimo elemento de uma lista linear;
e. Cópia de uma lista linear;
f. Combinação (mesclagem) de duas ou mais listas lineares em uma única lista linear;
g. Particionamento (quebra) de uma lista linear em duas ou mais listas lineares;
h. Determinação do tamanho de uma lista linear;
i. Ordenação dos elementos de uma lista linear;
j. Localização em uma lista linear de um elemento com uma determinada informação;
l. Exclusão de uma lista linear;
m. Outras...

Considerações sobre a implementação de listas lineares

De uma forma geral, em uma mesma implementação, ou seja, no mesmo programa, não é utilizado o conjunto completo das operações possíveis sobre uma lista linear.

Um motivo para que isso seja quase uma regra é a dificuldade de manter uma boa eficiência para todas as operações simultaneamente. Por exemplo, se uma implementação tem uma boa performance na inserção ou na exclusão de um elemento, pode ser pouco provável que essa mesma implementação consiga também obter um bom desempenho na operação de busca de qualquer elemento da lista.

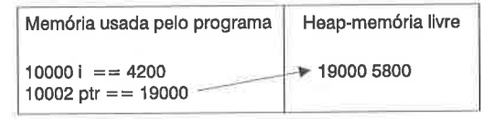
Considerações sobre a alocação de memória

Se considerarmos o ponto de vista do programador, a alocação de memória para executar um programa pode ser realizada de duas formas: estática ou dinâmica.

Na alocação de memória estática, o espaço de memória ocupado pelas variáveis é determinado no momento da compilação. Se o espaço de memória é determinado durante a execução do programa, a alocação de memória é realizada de forma dinâmica. É nesse caso que surgem as chamadas variáveis anônimas, pois não conhecemos seu nome. Sabemos apenas seu endereço de memória. Mais adiante, podemos encontrar um pequeno programa C que ilustra os conceitos de alocação estática *versus* dinâmica (Figura 2.2).

Para entendermos a ilustração da Figura 2.2, temos de supor que o tamanho do tipo inteiro é de dois *bytes*. Observe que, inicialmente, o conteúdo do endereço da variável 'i' é indefinido e, posteriormente, armazena o valor de 4.200. O mesmo acontece com a variável ponteiro 'ptr' que, posteriormente, armazena o valor do ponteiro do endereço (19.000) da área heap que contém o valor de 5.800. Lembramos você de que os endereços são fictícios.

Figura 2.2
Exemplo de alocação dinâmica.




```
main ( )
{
   int *ptr,i;
   ptr=(int *) malloc(sizeof(int));
   *ptr = 5800;
   i = 4200;
}
```

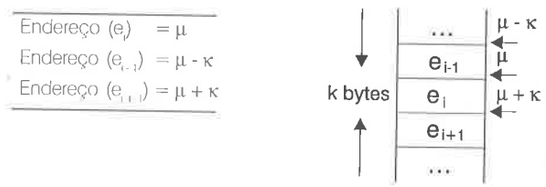
Formas de agrupar e acessar elementos em uma lista linear

Podemos apresentar duas formas para agrupar e acessar os elementos em uma lista linear: seqüencial e encadeada.

A forma seqüencial é considerada a possibilidade mais natural para se colocar elementos no interior de uma lista. Assume-se simplesmente que a colocação dos elementos de uma lista linear ocorre em células com memórias consecutivas, uma após a outra.

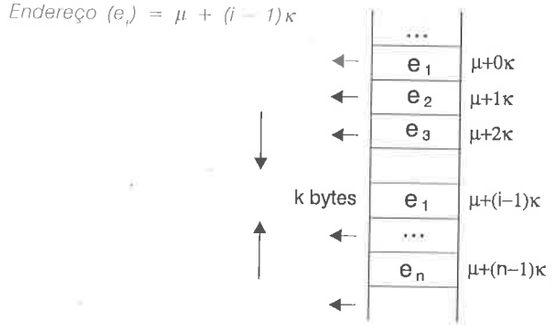
Para discutir as vantagens e as desvantagens do agrupamento dos elementos em uma lista linear de forma seqüencial, considere que cada célula de memória tem um endereço único μ e um tamanho de k *bytes*.

Figura 2.3
Agrupamento seqüencial.



Observando-se a Figura 2.3, pode-se conseguir uma generalização de uma equação para o cálculo exato do endereço de uma certa célula de memória. Considerando-se o elemento e_i, presente em uma célula de endereço μ, temos:

Figura 2.4
Representação do acesso direto no agrupamento seqüencial.



Uma vantagem do agrupamento seqüencial que podemos ressaltar é a facilidade para calcular o endereço de memória de um certo elemento de índice i. Para isso, basta saber o endereço inicial μ da área alocada pela lista linear, o tamanho de k *bytes* de cada célula de memória e o índice i do elemento que se deseja obter o endereço inicial de memória (Figura 2.4).

Já uma desvantagem que aparece em um agrupamento seqüencial de elementos de uma lista linear, surge no esforço do momento de inserir ou remover elementos no meio da lista. Isso ocorre porque é necessário movimentar os elementos para liberar (inserção) ou diminuir (remoção) o espaço entre eles.

A forma encadeada é a segunda possibilidade para agrupar os elementos em uma lista linear. Nesse caso, em vez de manter os elementos em células consecutivas, pode-se utilizar quaisquer células (não necessariamente consecutivas) para guardar os elementos da lista. Esse esquema implica que, para preservar a relação de ordem linear da lista, cada elemento armazena sua informação e também o endereço da próxima célula de memória válida.

Com o encadeamento, os elementos são armazenados em blocos de memória denominados nós. Cada nó possui dois campos: um para armazenar os dados e outro para o próximo endereço.

No esquema de agrupamento encadeado, dois endereços são muito importantes:

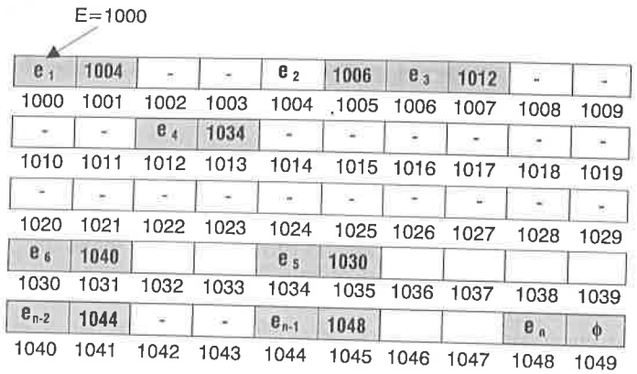
E = é o endereço do primeiro elemento;

ϕ = é o endereço do elemento fictício para o qual aponta o último elemento da lista (ϕ = nulo = null).

O endereço inicial ***E*** do primeiro elemento é essencial para a localização do ponto de início da lista linear com alocação encadeada. Sem esse endereço inicial ***E***, é impossível localizarmos o início de uma lista linear. Já o endereço ϕ, é importante porque marca o final da lista linear.

A ilustração na Figura 2.5 mostra o esquema de funcionamento de uma lista linear, considerando-se o agrupamento encadeado. Observe que, na representação abstrata da memória do computador, foram assumidos endereços fictícios de memória. Foi pressuposto que cada nó tem um tamanho de dois *bytes* e possui dois campos: um para a informação propriamente dita e outro para conter o endereço do próximo nó. Podemos notar que a ilustração da alocação encadeada sugere que os nós não estão necessariamente dispostos em células vizinhas. Na ilustração da Figura 2.5, está evidenciado a existência de um endereço ***E*** que aponta para o primeiro nó acessível da lista linear, e o conteúdo do campo de endereço do último nó da lista que tem valor nulo ϕ.

Figura 2.5
Representação do agrupamento encadeado.

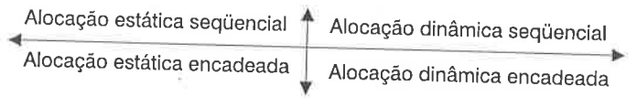


A grande vantagem do encadeamento aparece no momento da inserção ou da remoção de um elemento em qualquer ponto da lista linear. Isso acontece porque não é necessário movimentar nenhum elemento. Basta fazer a devida atualização na parte do nó correspondente ao endereço apontado para o próximo elemento.

A desvantagem aparece quando é necessário manipular um elemento específico da lista linear. Diferentemente do agrupamento seqüencial, no qual é possível até utilizar uma fórmula para calcular o endereço inicial de um elemento específico, com o encadeamento esse cálculo é impossível. Isso acontece porque só temos o endereço do primeiro elemento da lista linear. Por exemplo, se for necessário acessar o último elemento da lista linear, será necessário percorrer todos os elementos anteriores.

A seguir, exibimos um quadro que ilustra as diversas combinações entre alocações estática *versus* dinâmica e agrupamentos seqüencial *versus* encadeado.

Figura 2.6
Combinações de alocações estáticas *versus* dinâmicas e agrupamentos seqüenciais *versus* encadeados.



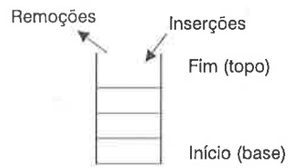
Listas lineares com disciplinas de acesso

Disciplinas de acesso referem-se à forma como são acessados os elementos de uma lista linear. Verificam se as inserções, as remoções e os acessos de consulta são realizados no primeiro ou no último elemento das listas. Elas podem ser classificadas em pilhas, filas ou, ainda, em filas duplas.

Pilhas (*'stacks'*)

É uma lista linear na qual todos os acessos (inserção, remoção ou consulta) são realizados em uma só extremidade, denominada TOPO.

Figura 2.7
Estrutura da pilha.



Equivalência de ações sobre as pilhas		
Inserir	⇔	Empilhar
Remover	⇔	Desempilhar

Analisando-se a possibilidade de inserir e remover elementos em uma pilha (Figura 2.7), percebemos claramente um padrão de comportamento que indica que o último elemento a ser inserido (empilhado), é o primeiro elemento a ser removido (desempilhado). Esse é o motivo das pilhas serem também referenciadas como uma estrutura *LIFO* (*'Last In, First Out'*).

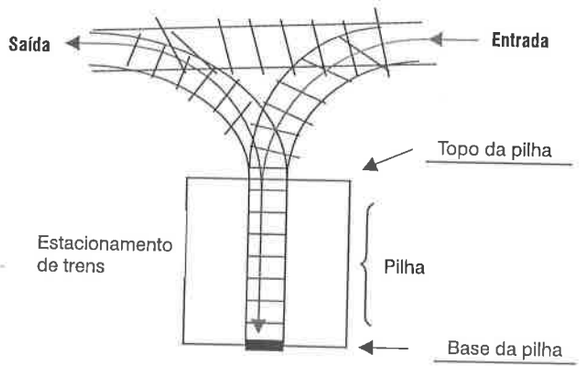
Algumas vezes, um porta-guardanapo é usado para ilustrar o funcionamento do mecanismo de uma pilha. Essa analogia deve ser considerada com cuidado, porque neste caso pode existir uma mola que ajuda na retirada ou na colocação de um papel no porta-guardanapo. Isso pode causar a impressão de que é a base da pilha que se movimenta, quando, na realidade, é o topo que faz a movimentação.

Por sua vez, se considerarmos uma pilha de pratos, a analogia ajusta-se de maneira mais adequada. Isso acontece porque uma pilha de pratos aumenta ou diminui por um único lado: seu topo. Nessa analogia, percebemos que é o topo que se movimenta, não a base.

Uma analogia muito interessante foi proposta por *E. W. Dijkstra* e usa um estacionamento de trens para facilitar o estudo e a compreensão da estrutura de uma pilha. A idéia do estacionamento de trens é a possibilidade da realização de permutações das composições de trens. As pilhas são utilizadas de forma semelhante, ou seja, são usadas nas situações de reversão da ordem de entrada de dados.

Na Figura 2.8, vemos uma ilustração do estacionamento de trens e percebemos que estes só podem entrar ou sair desse estacionamento apenas por um lado. Esse lado funciona como o topo de uma pilha. Isso significa que toda vez que um trem está entrando ou saindo, está sendo realizado o empilhamento ou o desempilhamento de elementos em uma pilha.

Figura 2.8
Analogia da pilha.



Vamos realizar um exercício simples para visualizarmos como essa analogia pode ser útil na compreensão do funcionamento de uma pilha. Suponha que existem três trens na entrada, numerados na seqüência de 1, 2 e 3, e nessa respectiva ordem. Os trens podem sair do estacionamento na ordem 2, 1 e 3? E na ordem 3, 1 e 2? E na ordem 2, 3 e 1?

Operações primitivas que manipulam as pilhas

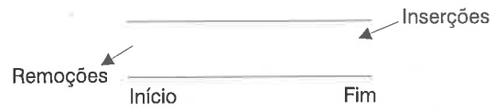
A seguir, estão algumas operações primitivas e seus respectivos significados quando aplicadas sobre as pilhas:

a. InicPilha (s)	Faz a pilha *s* ficar vazia.
b. PilhaVazia (s)	Retorna V, se a pilha *s* está vazia.
c. PilhaCheia (s)	Retorna V, se a pilha *s* está cheia.
d. Empilha (s,x) ou push (s,x)	Insere o elemento x no topo da pilha *s*.
e. Desempilha (s) ou pop (s)	Remove o elemento do topo da pilha *s*, Retornando-o como valor da função.
f. ElemTopo (s)	Acessa um elemento do topo da pilha *s*, sem, contudo, removê-lo.

Fila ('queues')

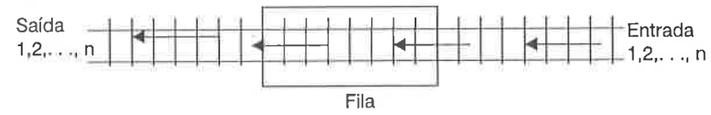
É uma lista linear na qual todas as inserções são realizadas em uma das extremidades (fim da fila). E, além disso, todas as remoções e os acessos são realizados em outra extremidade (início da fila). Esse é o motivo pelo qual as filas são denominadas *FIFO* (*First-In First-Out*) (Figura 2.9).

Figura 2.9
Estrutura da fila.



Também existe uma analogia com estacionamento de trens, proposta por *E. W. Dikjstra*, para ilustrar o funcionamento de uma fila.

Figura 2.10
Analogia da fila.



Observando-se a analogia (Figura 2.10), percebemos a ocorrência de uma disciplina na forma que os trens entram no estacionamento. Se um trem entra pelo lado denominado entrada, ele sairá somente pelo lado denominado saída, caracterizando, assim, a disciplina de acesso em uma fila.

Como exercício, podemos supor que existam *n* trens na entrada e na ordem 1,2, . . ., n. A única seqüência possível na saída do estacionamento é 1,2, . . ., n.

Operações primitivas que manipulam as filas

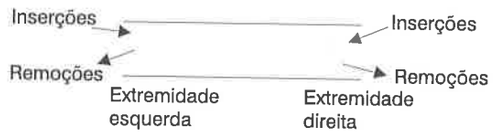
Também, como ocorre nas pilhas, existem operações primitivas que podem atuar sobre as filas. Logo abaixo, estão algumas delas:

a. InicFila(q)	Faz a fila q ficar vazia.
b. FilaVazia(q)	Retorna V, se a fila q estiver vazia.
c. FilaCheia(q)	Retorna V, se a fila q estiver cheia.
d. InsFila(q, x)	Insere o elemento x no final da fila q.
e. RemFila(q)	Remove o último elemento da fila, retornando o conteúdo do elemento como o valor da função.

Fila dupla (*'deque'*)

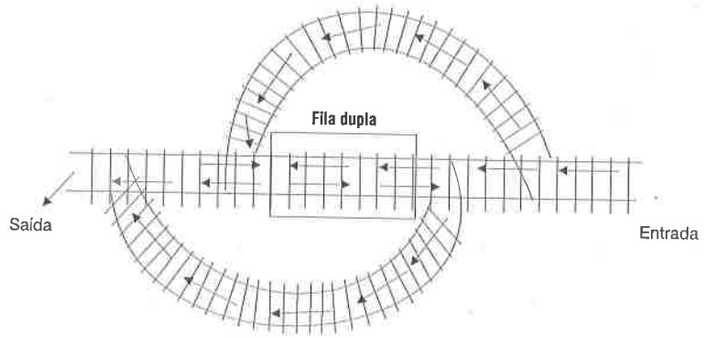
É uma lista linear na qual todas as inserções, as remoções e os acessos são realizados nos extremos direito ou esquerdo da lista linear (Figura 2.11).

Figura 2.11
Estrutura da fila dupla.



A analogia com um estacionamento de trens (Figura 2.12) continua ainda sendo útil para compreendermos o funcionamento do mecanismo de uma fila dupla.

Figura 2.12
Analogia da fila dupla.

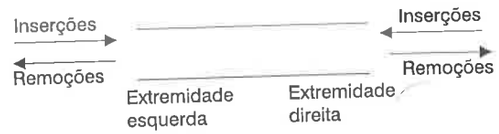


Fila dupla com saída ou entrada restrita

Dependendo da prioridade necessária para um determinado elemento, a fila dupla pode ser classificada como fila de saída ou de entrada restrita.

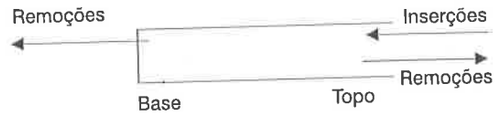
Uma fila dupla com saída restrita ocorre quando queremos dar prioridade a um determinado elemento, e, para isso, permitimos que, no momento da inserção, o elemento seja colocado no início da fila dupla. A ilustração a seguir mostra como ocorrem as inserções e as remoções em uma fila dupla com saída restrita (Figura 2.13).

Figura 2.13
Estrutura da fila dupla com saída restrita.



No caso de uma pilha, se ocorrer um estouro de sua capacidade (*overflow*), no momento da inserção de um novo elemento, podemos permitir que nesta situação o elemento da base seja removido (já que ele é o elemento que está há mais tempo na pilha). Essa possibilidade de quebrarmos a regra básica de uma pilha, que diz que a inserção ou a remoção só pode ocorrer pelo topo da pilha, dá origem a uma estrutura denominada fila dupla com entrada restrita (Figura 2.14).

Figura 2.14
Estrutura da fila dupla com entrada restrita.



EXERCÍCIOS DE APROFUNDAMENTO

1. Desenhe a evolução do conteúdo de uma pilha, considerando, para isso, a execução das duas seqüências de instruções:

Seqüência 1	Seqüência 2
Push (s, 'a')	Push (s, 'a')
Push (s, 'b')	Push (s, 'b')
Push (s, 'c')	Push (s, 'c')
Pop (s)	x←ElemTopo (s)
Pop (s)	Pop (s)
Pop (s)	y←ElemTopo (s)
	Pop (s)
	Push (s, x)
	Push (s, y)
	Pop (s)

2. Uma pilha pode ser encarada — hipoteticamente — como um estacionamento de trens, conforme já mostrado anteriormente. Suponha a existência de trens numerados 1, 2, . . ., n do lado direito da Figura 2.6. Deseja-se permutar os trens que deixam a estação pelo lado esquerdo. Um trem que tenha entrado no estacionamento pode ser deixado nesse local ou pode sair pelo lado esquerdo, contudo jamais pode retornar pelo lado direito do estacionamento. Por exemplo, se n = 3 e temos os trens 1, 2 e 3 no lado direito, então o trem 1 pode entrar no estacionamento. Poderíamos, então, enviar o trem 2 para o estacionamento. Em seguida, o trem 3 pode ser enviado diretamente para o lado esquerdo, e, logo depois, o trem 2 sai para o lado esquerdo e, finalmente, o trem 1. Obteríamos uma nova ordem, como 3, 2, 1. Dessa forma, solicita-se o seguinte:

   - Para n = 3, encontre todas as permutações possíveis de serem obtidas.
   - Idem para n = 4.
   - **(Desafio)** Tente achar uma fórmula genérica que possa ser utilizada para encontrar o número de permutações que podem ser obtidas considerando um certo valor *n*.

3. Discorra sobre as principais características das pilhas, filas, filas duplas, filas duplas com entrada restrita e filas duplas de saída restrita.

4. Qual a razão da implementação de listas na forma seqüencial ser considerada mais eficiente na pesquisa de um elemento arbitrário?

5. Comente sobre as vantagens e as desvantagens em utilizar os chamados agrupamentos encadeados e seqüenciais?

6. Qual o relacionamento das alocações estáticas e dinâmicas com os agrupamentos encadeados e seqüenciais?

7. Considere trens numerados de 1 a 5. Indique duas permutações que não podem ser observadas como saída de um estacionamento de trens na forma de fila dupla.

8. Suponha um estacionamento de trens na forma de pilha e responda às seguintes questões:

   a. Imagine a existência de seis trens na entrada do estacionamento numerados de 1 até 6. É possível obter uma saída na ordem 3, 2, 5, 6, 4 e 1 ? E na ordem 1, 5, 4, 6, 2 e 3? Justifique sua resposta.

   b. Construa um algoritmo que tenha como entrada uma permutação do tipo p_1, p_2, p_3,. . ., p_n de 1, 2, 3,... n trens, indicando se é ou não possível os trens saírem na permutação informada. Além disso, o algoritmo deve indicar a movimentação exata dos trens (quem saiu e quem entrou no estacionamento).

   c. Verifique se a afirmação abaixo é verdadeira (justifique sua resposta):

   p_1, p_2,... p_n pode ser considerado uma permutação conseguida de 1, 2,..., n, por meio do uso de uma pilha, se — e somente se — não existem índices i, j e k que satisfazem a seguinte relação : $i < j < k$ e $p_j < p_k < p_i$.

9. Suponha a possibilidade de manobrar trens na forma de uma fila dupla e que existem quatro trens na entrada. Responda cada questão abaixo e justifique sua resposta:

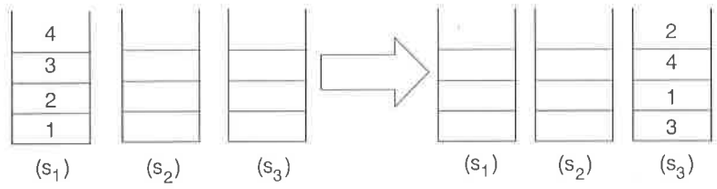
   a. Relacione quais permutações podem ser obtidas de uma fila dupla? E se considerarmos a existência de cinco trens na entrada. Quais permutações não podem ser obtidas da fila dupla?

   b. Identifique uma permutação possível de ser obtida a partir da fila dupla com entrada restrita, mas que não seja possível de ser obtida em uma fila dupla com saída restrita.

   c. Identifique pelo menos uma permutação que podemos obter de uma fila dupla com saída restrita, mas que não seja possível conseguirmos em uma fila dupla com entrada restrita.

   d. Identifique pelo menos uma permutação que não seja possível conseguirmos a partir de uma fila dupla com entrada restrita, tampouco de uma filha dupla com saída restrita.

10. Considerando a ilustração a seguir, mostre a seqüência de operações *Push* e *Pop* que devem ser realizadas sobre as pilhas s_1, s_2 e s_3 para que, partindo do estado inicial, possamos chegar ao estado final.



11. Considere uma *string* formada apenas pelos caracteres '+' e '–' que representam respectivamente as operações *Push* e *Pop*. Suponha que *MAX* é o número máximo de elemento que a pilha *s* pode conter. É solicitada a elaboração de um algoritmo que, dados *string* e *MAX*, indique se a *string* de operações será possível de ser realizada sem causar *underflow* ou *overflow*.

   Underflow = É observado na tentativa de remover elementos de uma pilha vazia.

   Overflow = É observado na tentativa de inserir elementos em uma pilha cheia.

12. Suponha a existência de uma linguagem hipotética na qual não existam vetores, mas que apresente uma pilha como um tipo de dados primitivo. Seria possível implementar os vetores a partir do tipo de dados pilha? Se sim, demonstre como fazê-lo.

13. Mostre a situação da pilha *s*, inicialmente vazia, após a execução de cada uma das operações mostradas a seguir:

1 push (s, '1')

2 push (s, '2')

3 push (s, '3')

4 push (s, elemTopo (s))

5 push (s,pop (s))

6 pop (s)

7 push (s, '4')

8 pop (s)