

Unidade 6

Listas Encadeadas: Conceito, Representação e Algoritmos

Entendeu ou quer que eu desenhe?




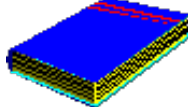
Nossos Objetivos nesta Unidade:

- Entender o conceito de listas encadeadas, e conhecer sua representação usual; entender a diferença entre alocação seqüencial e alocação encadeada de memória, no contexto do armazenamento de conjuntos de elementos;
- Desenvolver a habilidade para implementar Pilhas e Filas, através do conceito de listas encadeadas.

Aplicação para Contextualização: Automação de Biblioteca

Vamos voltar à aplicação que já começamos a discutir na Unidade 5: automação de biblioteca:

- Todos os livros devem ser cadastrados;
- O sistema deve informar se um determinado livro está ou não disponível nas estantes;
- Caso o livro não estiver disponível, o usuário poderá aguardar pela liberação do livro se cadastrando em uma fila de espera;
- Quando o livro for devolvido e liberado, o primeiro da fila deve ser contatado para vir buscá-lo.

Fila de Espera para o Livro	Livros do Acervo	Livro Disponível?
	 Trigonometria	não
	 Química Inorgânica	não
		sim

Fila vazia!



Estruturas de Dados

Tabela 6.1 Filas de espera por livros em um sistema de automação de biblioteca

Você já definiu a funcionalidade da fila (Tabela 5.2 – entra na fila, sai da fila, etc.), e pensou em implementar a fila como um vetor circular, conforme estudamos na Unidade 5. Agora você está em uma reunião de projeto, e está conhecendo mais um pouco sobre a biblioteca:

- A biblioteca conta com 120.000 livros;
- Um máximo de 1.000 pessoas ficam a espera por um dos livros da biblioteca;
- Até 30 pessoas ficam a espera de um mesmo livro.

Você começa a pensar na implementação das filas de espera pelos livros. Como são 120.000 livros, e para cada livro precisamos ter uma fila de espera, teremos 120.000 filas de espera. Como até 30 pessoas ficam na fila de espera por um único livro, cada fila de espera precisa caber pelo menos 30 pessoas. Em cada uma das 120.000 filas, é preciso reservar espaço para 30 pessoas. Ou seja, 120.000 filas, multiplicado por 30 pessoas: isso resulta em espaço reservado para 3.600.000 pessoas. Isso é incrível, pois sabemos que um máximo de 1.000 pessoas ficam a espera por um dos livros da biblioteca. Será que não temos uma alternativa mais interessante para implementar essas filas de espera?

Caminho Alternativo: Compartilhar Espaço!

Você pensa em um caminho alternativo: se sabemos que um máximo de 1.000 pessoas ficam a espera por um dos livros na biblioteca, por que não reservar espaço para 1.000 pessoas (ao invés de espaço para 3.600.000), e então compartilhar esse espaço entre as 120.000 filas de espera?

Alternativa 1	Alternativa 2
<ul style="list-style-type: none">• Reservar espaço para 120.000 filas (uma para cada livro), com capacidade para 30 pessoas;• 120.000 vetores de tamanho {30};• Espaço reservado para 3.600.000 pessoas;• Muito espaço reservado não é utilizado.	<ul style="list-style-type: none">• Alocar espaço para 1.000 pessoas na fila;• Todas as 120.000 filas compartilham esse espaço reservado para 1.000 pessoas;• Problema: como 120.000 filas podem compartilhar a memória reservada para essas 1.000 pessoas?

Tabela 6.2 Alternativas para a implementação das filas de espera

Como Várias Estruturas Podem Compartilhar um Espaço de Memória?

A alternativa 2 parece interessante, mas como podemos compartilhar um mesmo espaço de memória entre diversas (possivelmente 120.000) estruturas de armazenamento?

Vamos fazer um estudo. Para simplificar, ao invés de um espaço para 1.000 pessoas, vamos

reservar um espaço para 5 pessoas. Vamos usar para isso um vetor de tamanho 5. Será o nosso “banco de memória”, a ser compartilhado. E ao invés de compartilhar esse espaço entre 120.000 estruturas de armazenamento, vamos compartilhar esse espaço entre apenas 3 estruturas de armazenamento do tipo Pilha: Pilha X, Pilha Y e Pilha Z.

Veja na Tabela 6.3, na primeira linha, o banco de memória inicialmente vazio. A segunda linha traz a operação Push(X, A), que significa que estamos empilhando (push) o elemento A na pilha X. Note ainda na segunda linha que a posição 1 do banco de memória não está mais disponível. Ela está reservada agora para a pilha X. Note também que o elemento A está no topo da pilha X.

Banco de Memória	Pilha X	Pilha Y	Pilha Z	Operação
<div> <div>1</div> <div>2</div> <div>3</div> <div>4</div> <div>5</div> </div>				
<div> <div>2</div> <div>3</div> <div>4</div> <div>5</div> </div>	<div> <div>a</div> <div>1</div> </div>			push(x, a)
<div> <div>3</div> <div>4</div> <div>5</div> </div>	<div> <div>a</div> <div>1</div> </div>	<div> <div>b</div> <div>2</div> </div>		push(y, b)
<div> <div>4</div> <div>5</div> </div>	<div> <div>a</div> <div>1</div> <div>c</div> <div>3</div> </div>	<div> <div>b</div> <div>2</div> </div>		push(x, c)
<div> <div>5</div> </div>	<div> <div>a</div> <div>1</div> <div>c</div> <div>3</div> </div>	<div> <div>b</div> <div>2</div> </div>	<div> <div>d</div> <div>4</div> </div>	push(z, d)
<div> <div>2</div> <div>5</div> </div>	<div> <div>a</div> <div>1</div> <div>c</div> <div>3</div> </div>		<div> <div>d</div> <div>4</div> </div>	pop(y)
<div> <div>5</div> </div>	<div> <div>a</div> <div>1</div> <div>c</div> <div>3</div> </div>		<div> <div>d</div> <div>4</div> <div>e</div> <div>2</div> </div>	push(z, e)
<div>visão do banco de memória</div> <div>X = ocupado</div> <div> <div>5</div> <div>X 4</div> <div>X 3</div> <div>X 2</div> <div>X 1</div> </div>	<div>visão pilha x</div> <div> <div>5</div> <div>4</div> <div>c 3</div> <div>2</div> <div>a 1</div> </div>	<div>visão pilha y</div> <div> <div>5</div> <div>4</div> <div>3</div> <div>2</div> <div>1</div> </div>	<div>visão pilha z</div> <div> <div>5</div> <div>d 4</div> <div>3</div> <div>e 2</div> <div>1</div> </div>	

Tabela 6.3 Duas pilhas compartilhando um banco de memória

Na terceira linha temos a operação Push(Y, B), que empilha o elemento B na pilha Y. Na quarta linha temos Push(X, C). A pilha X passa a ter 2 elementos, A e C, e C está no topo da pilha. As operações push e pop vão acontecendo e, ao final, na última linha da tabela, temos o banco de memória com 4 posições ocupadas: 2 pela pilha X, 2 pela pilha Z. A pilha Y está

vazia.

Problemas a Resolver: Seqüência dos Elementos em uma Pilha

Mas ainda temos algumas questões a responder: como saber qual é o elemento do topo em cada uma das pilhas? Além disso, como poderemos definir a seqüência dos elementos? Na alocação seqüencial, que utilizamos nas Unidades 3 e 5, a seqüência dos elementos da pilha (Unidade 3) e fila (Unidade 5) era definida implicitamente. Se o primeiro elemento estava na posição 1 do vetor, implicitamente sabíamos que o segundo elemento estaria na posição 2 do vetor, e assim por diante. Mas agora os elementos de cada pilha não estão armazenados em posições seqüenciais do vetor. Os elementos da pilha X estão ocupando as posições 1 e 3. A posição 2 está sendo ocupada pela pilha Z. Como saberemos que na pilha X o elemento C, que está no topo, está na posição 1 do vetor, e o segundo elemento está na posição 3 do vetor?

Definição de Alocação Encadeada

Precisamos diferenciar, nesse momento, a alocação seqüência de memória, da alocação encadeada de memória. Na alocação seqüencial (que utilizamos nas unidades 3 e 5), os elementos eram armazenados em posições adjacentes do vetor, ou seja, em posições adjacentes de memória. A seqüência dos elementos é física, real, e implícita.

Na alocação encadeada de memória os elementos de uma estrutura de armazenamento não estão necessariamente em posições adjacentes de memória. E a seqüência não é definida implicitamente. Precisamos definir a seqüência explicitamente. No exemplo da Tabela 6.3, precisamos indicar, explicitamente, que o primeiro elemento da pilha X está na posição 3, e o segundo elemento está na posição 1.

Para indicar explicitamente qual é o primeiro elemento, ou o elemento que está no topo da pilha, usaremos um apontador, ou ponteiro. Também utilizaremos um ponteiro para indicar qual é o próximo elemento. Ou seja, para indicar que depois do elemento que está armazenado na posição 3, vem o elemento que está na posição 1. Veja a figura 6.1.

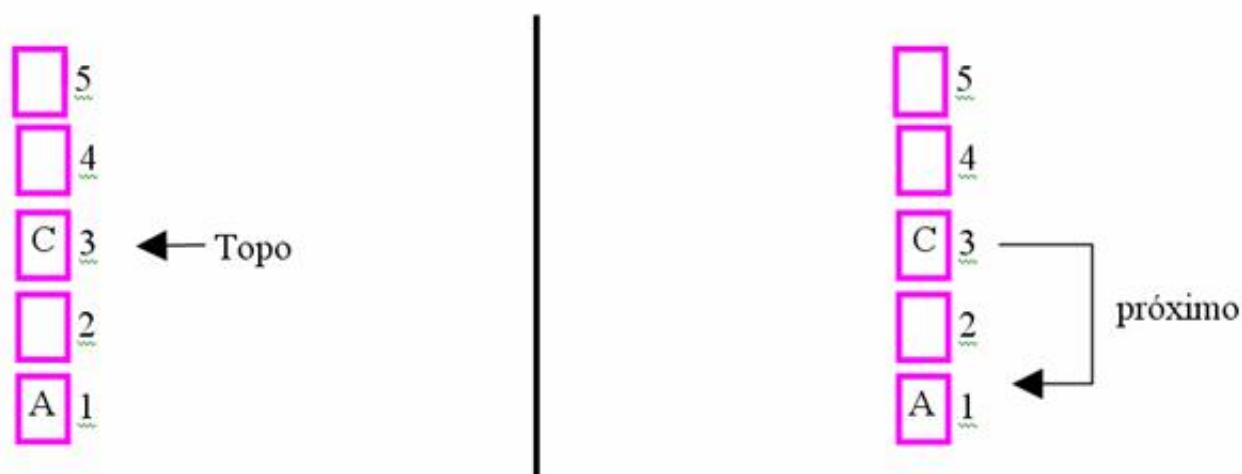


Figura 6.1 Indicação explícita do primeiro e do próximo

Definição e Representação de Listas Encadeadas

Definição: Uma lista encadeada L , com n blocos de memória B_1, B_2, \dots, B_n é definida pelas seguintes características:

- Cada bloco de memória B_i , ou cada “nó” da lista, tem dois campos: informação a ser armazenada, e a indicação do próximo elemento da lista. Na figura 6.1, a informação é representada pelos caracteres A, B, C e D. A indicação do próximo elemento é indicada pela seta. Um ponteiro para o próximo elemento;
- Os blocos de memória não estão necessariamente em seqüência física, na memória;
- O acesso aos elementos da lista ocorre através de um ponteiro para o início da lista (o primeiro elemento); Na figura 6.2, o início da lista é indicado por L ;
- O acesso aos demais elementos ocorre sempre a partir do primeiro elemento (L), a seguir através da indicação de quem é o próximo na seqüência. Não é possível ter acesso direto ao elemento C, no bloco B_3 , por exemplo. Para acessar o elemento C, no bloco B_3 , é preciso acessar o primeiro elemento através de L , depois ir para o próximo elemento através do ponteiro, e depois para o próximo novamente;
- O último nó da lista aponta para um endereço inválido, chamado NIL ou ainda NULL. É essa a indicação de que o bloco de memória B_n guarda o último elemento do conjunto;
- A lista vazia (sem elementos) é representada pelo apontador de início da lista (L) apontando para NULL.

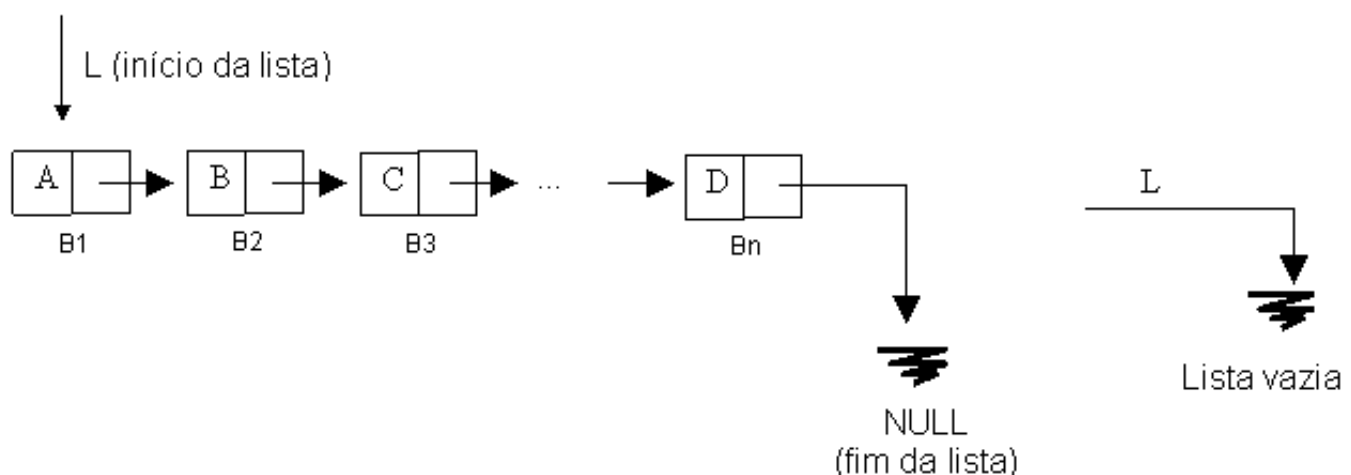


Figura 6.2 Representação de listas encadeadas

Notação para Algoritmos Sobre Listas Encadeadas

Considere a definição do tipo Node como:

```
Tipo Node = Registro
    Info : Char;
    Next : Ponteiro
```

O tipo Node define o tipo dos blocos de memória (B1, B2... Bn) representados na Figura 6.2, e detalhados na Figura 6.3. Cada bloco de memória, ou cada “nó” (ou Node, em inglês), possui 2 campos: campo Info, que armazena a informação, e o campo next, que armazena a indicação do próximo elemento no conjunto. Ou seja, esse “next” encadeia os elementos do conjunto. Daí o nome, listas encadeadas, ou alocação encadeada de memória.

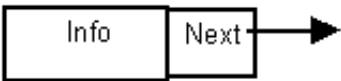


Figura 6.3 Detalhe de um nó, com os campos info e next

Considere também as variáveis:

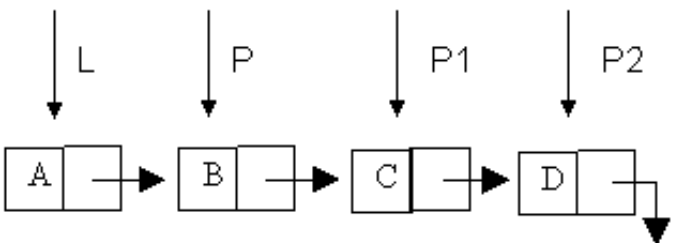
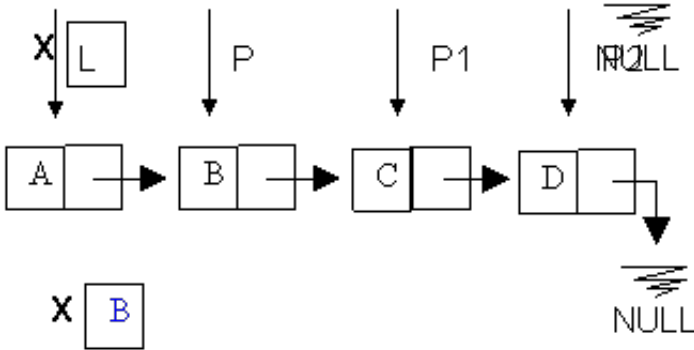
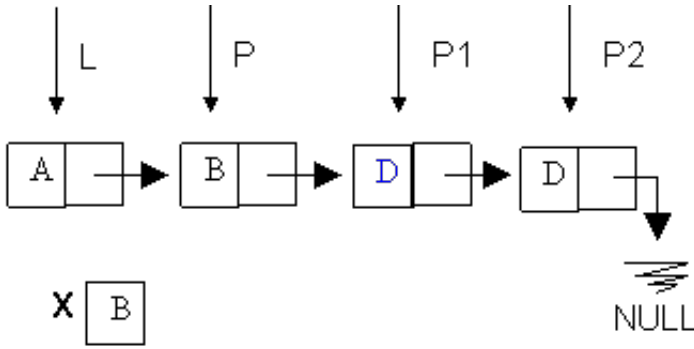
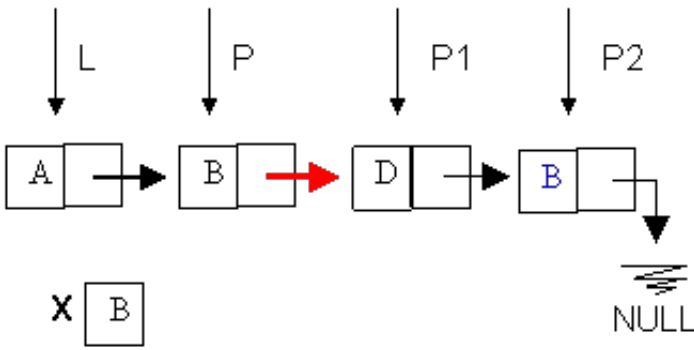
- P, P1, e P2 variáveis do tipo Ponteiro (ou seja, do mesmo tipo do campo Next);
- X uma variável do tipo char (ou seja, do mesmo tipo do campo Info).

Considere válidas as seguintes operações sobre nós e ponteiros:

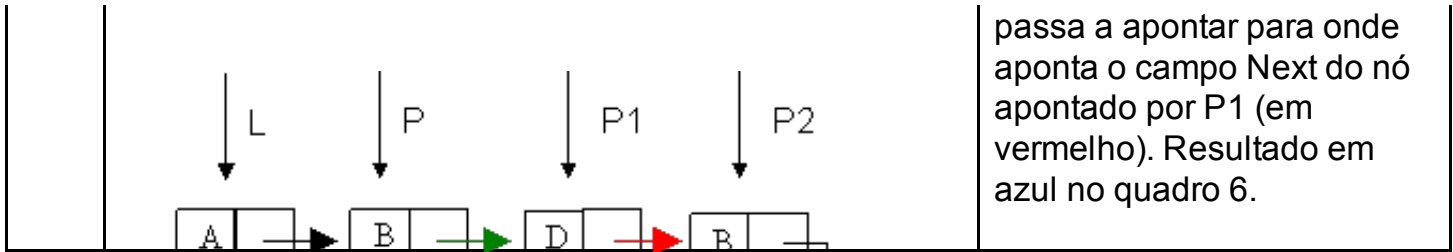
- P = GetNode { aloca um bloco de memória tipo Node e retorna o endereço em P }
- Freenode(P) { libera o bloco de memória apontado por P }
- Info(P) = X { o campo Info do nó apontado por P recebe o valor de X }
- X = Info(P) { X recebe o valor do campo Info no nó apontado por P }
- P1 = P2 { o ponteiro P1 passa a apontar para onde aponta P2 }
- P1 = Next(P) { P1 passa a apontar para onde aponta o campo Next do nó apontado por P }
- Next (P) = P1 { o campo Next do nó apontado por P passa a apontar para onde aponta P1 }

Exercício 6.1: Exercitando a Notação para Manipulação de Listas Encadeadas

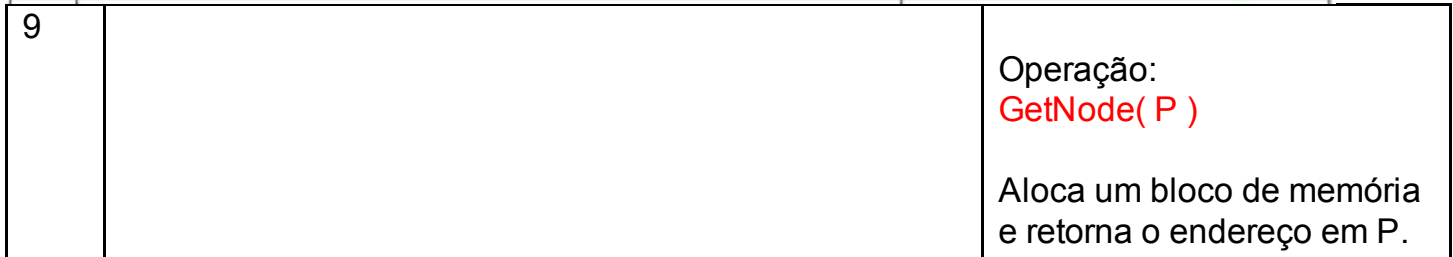
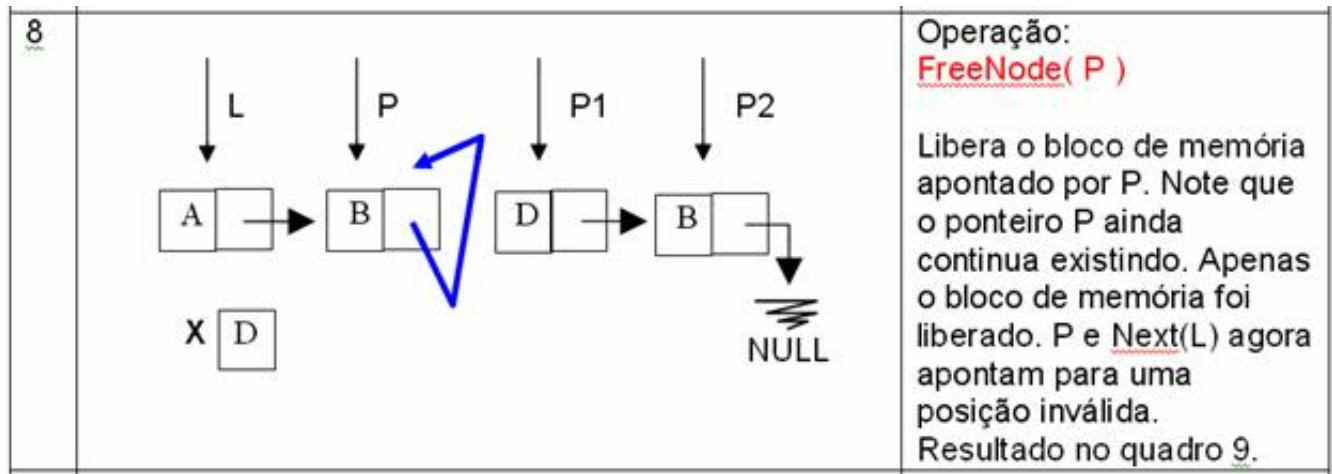
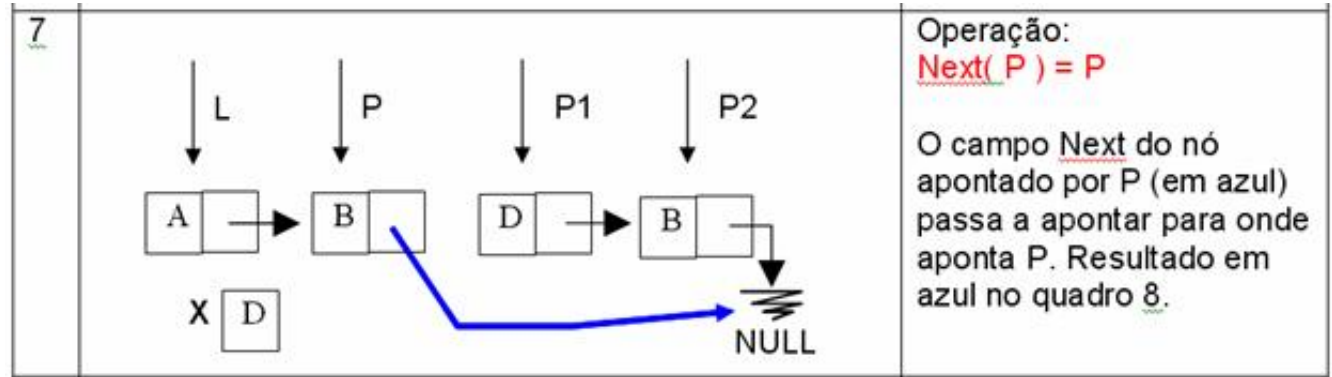
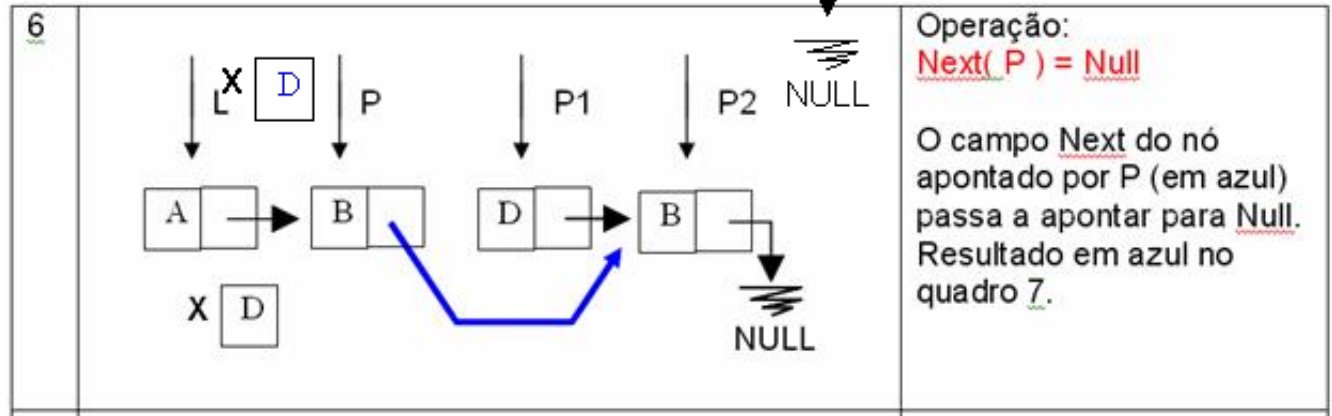
1	Situação inicial. Aplicamos a operação: X = Info(P)
---	-----------------------------------------------------------------

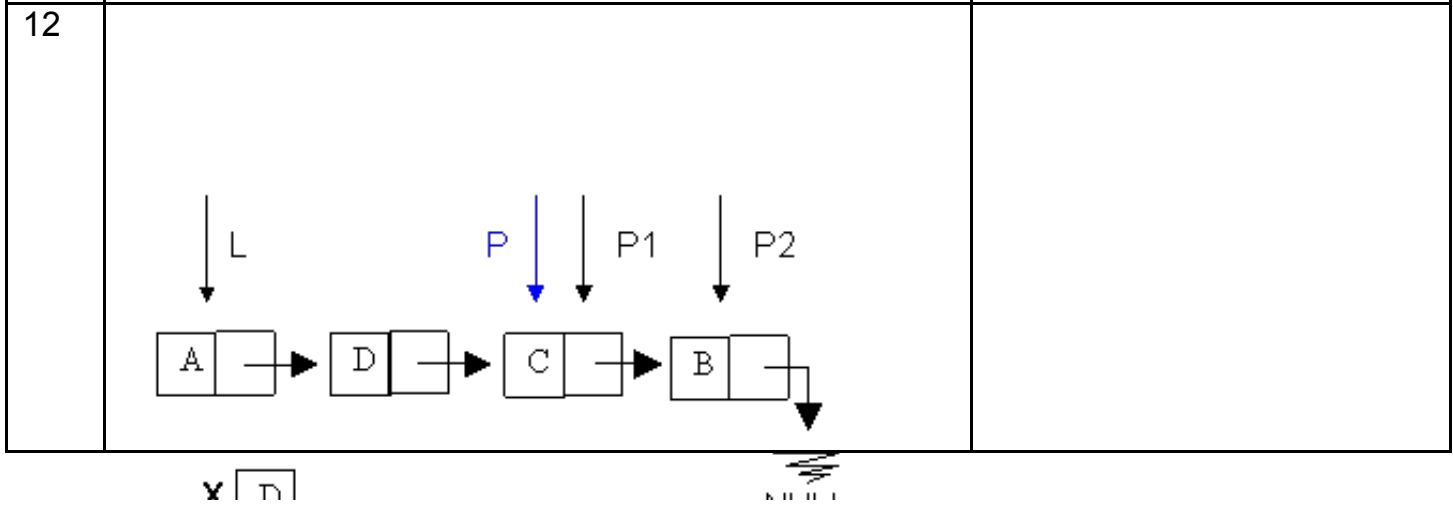
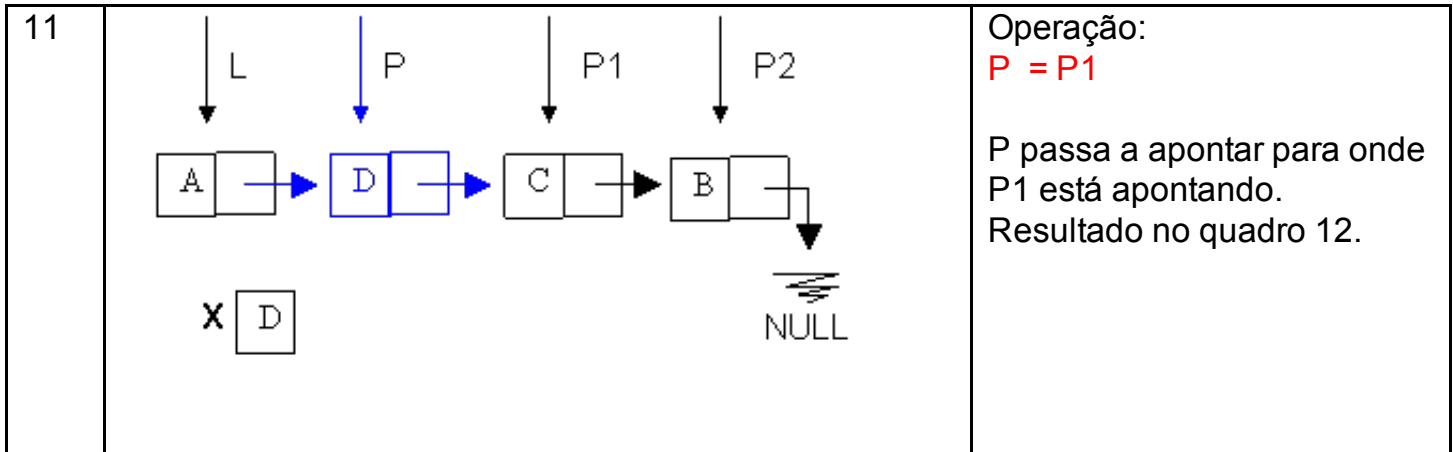
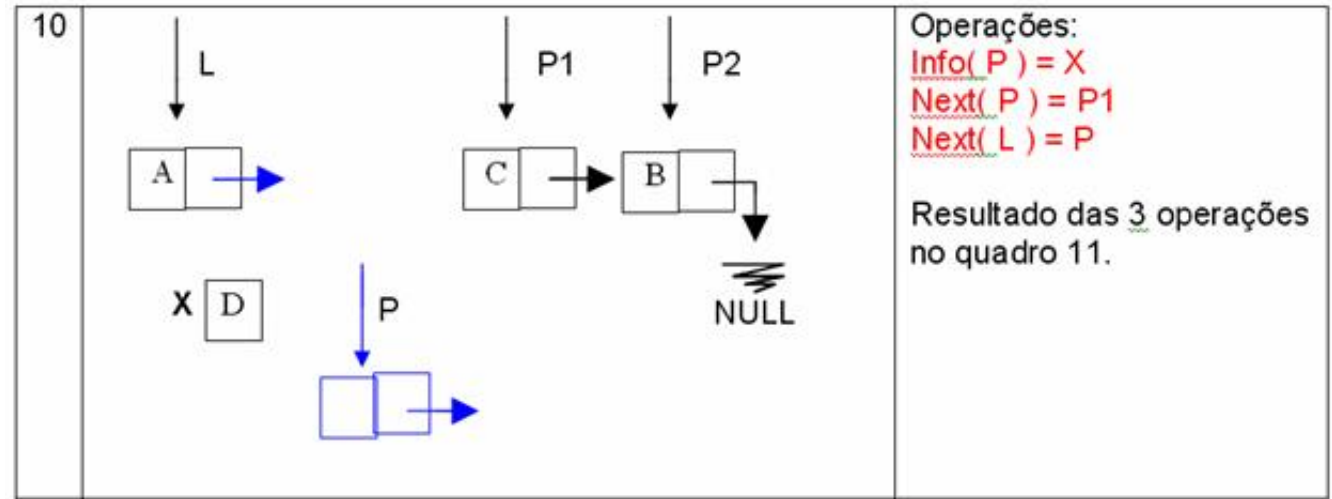
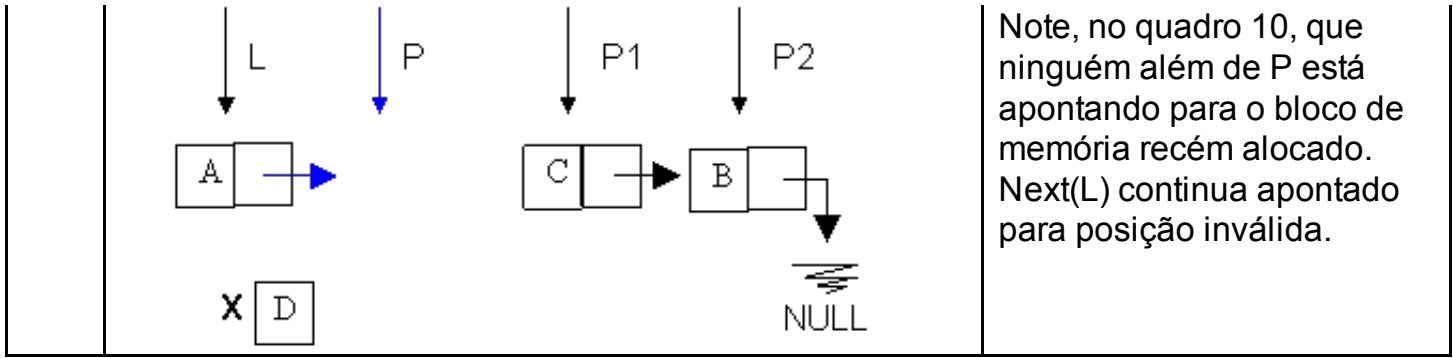
		<p>Isso significa que a variável X passará a ter o valor do campo Info do Nó apontado pelo Ponteiro P. Ou seja, X recebe o valor 'B'. Veja no quadro 2, em azul.</p>
2		<p>Operação: Info(P1) = Info(P2)</p> <p>O campo Info do nó apontado por P1 recebe o valor do campo Info do Nó apontado por P2. Veja o resultado no quadro 3, em azul.</p>
3		<p>Operação: Info(P2) = X</p> <p>O campo Info do nó apontado por P2 recebe o valor da variável X. Veja o resultado no quadro 4, em azul.</p>
4		<p>Operação: X = Info(Next(P))</p> <p>A variável X recebe o valor do campo Info do nó apontado pelo ponteiro Next(P), destacado em vermelho. Resultado em azul no quadro 5.</p>

5		<p>Operação: Next(P) = Next(P1)</p> <p>O campo Next do nó apontado por P (em verde)</p>
---	--	-----------------------------------------------------------------------------------------------------



passa a apontar para onde aponta o campo Next do nó apontado por P1 (em vermelho). Resultado em azul no quadro 6.





Simulação 6.1 Notação para Listas Encadeadas

Exercite sua compreensão da notação sobre listas encadeadas interagindo com a [Simulação 6.1: Notação para Listas Encadeadas](#). **Atenção: clique com o mouse para avançar a apresentação.**

Implementação de Pilha Através do Conceito de Listas Encadeadas

A Figura 6.4 apresenta diagramas contendo a representação de uma pilha, através do conceito de listas encadeadas. Na primeira linha, temos uma pilha vazia. O ponteiro Topo está apontando para Null. Na segunda linha o ponteiro Topo está apontando para o primeiro e único elemento da pilha, que contém o valor A no campo de informação. Na terceira linha, a pilha possui 2 elementos, sendo que B é o elemento do topo.

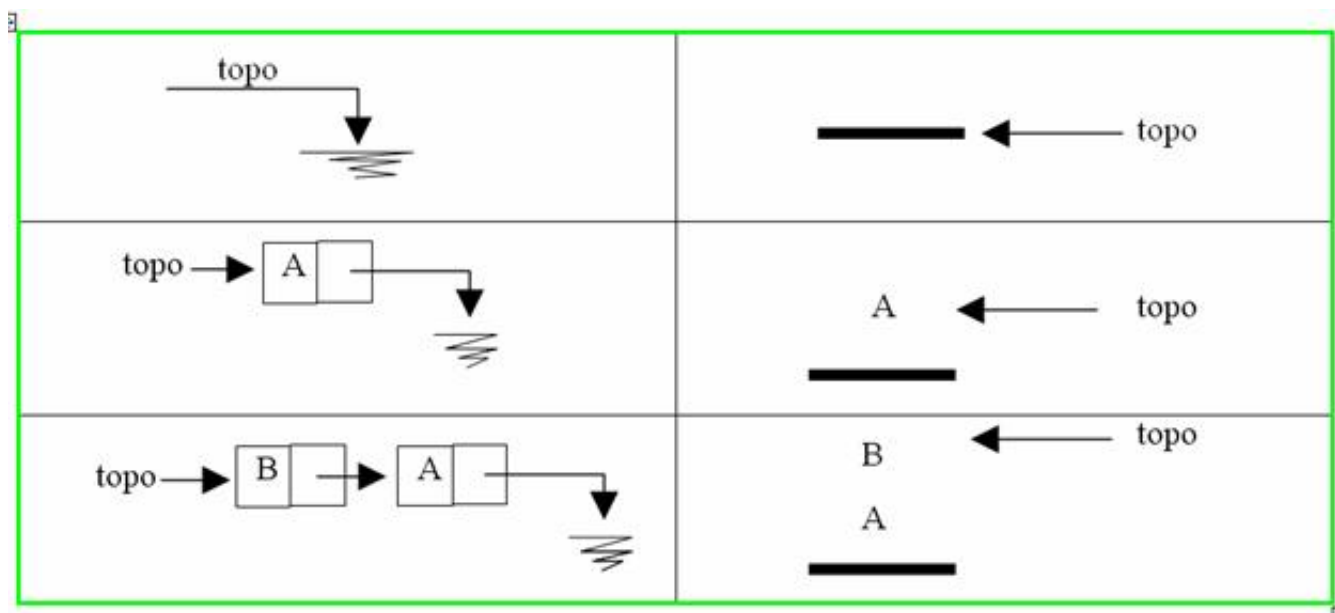


Figura 6.4 Representação de uma Pilha Como uma Lista Encadeada

Algoritmos para as Operações para Manipulação de Pilhas

Vamos implementar as operações sobre pilhas definidas na Unidade 3, agora utilizando o conceito de listas encadeadas:

- **Empilha(QualPilha, QualElemento, DeuCerto?):** Empilha o elemento passado como parâmetro QualElemento, na pilha passada no parâmetro QualPilha. O parâmetro DeuCerto? indica se a operação foi bem sucedida ou não.
- **Desempilha(QualPilha, QualElemento, DeuCerto?):** Desempilha (retira o elemento do topo) da pilha passada no parâmetro QualPilha, retornando o valor do elemento que foi desempilhado no parâmetro QualElemento. O parâmetro DeuCerto? indica se a operação foi bem sucedida ou não.
- **Vazia?(QualPilha):** Verifica se a pilha passada como parâmetro (QualPilha) está ou não vazia (vazia = sem nenhum elemento).
- **Cria(QualPilha):** Cria uma pilha, iniciando sua situação como vazia.

Exercício 6.2: Operações Cria e Vazia

Cria (parâmetro por referência Topo do tipo Pilha)

/* Cria uma pilha, iniciando sua situação como vazia. */

Topo = Null

Boolean **Vazia**(parâmetro por referência Topo do tipo Pilha)

/* Verifica se a pilha passada como parâmetro (P) está ou não vazia (vazia = sem nenhum elemento). */

se Topo == Null

então Retorne Verdadeiro

senão Retorne Falso

Na operação Cria devemos Apontar o ponteiro Topo para Null. Isso indicará que a pilha está vazia. Veja na Figura 6.4 – 1ª linha. A operação Vazia simplesmente irá testar se o topo da pilha está apontando para Null. Se estiver em Null, a pilha estará vazia. Se não estiver em Null, a pilha não estará vazia.

Simulação 6.2: Operações Cria e Vazia

Veja uma simulação da execução e uso das operações Cria e Vazia, interagindo com a [Simulação 6.2: Operações Cria e Vazia](#). **Atenção: clique com o mouse para avançar a apresentação.**

Exercício 6.3 Operação Empilha

Empilha(parâmetro por referência Topo do tipo Pilha, parâmetro Elemento do tipo Char, parâmetro por referência DeuCerto? do tipo Boolean)

/* Empilha o elemento Elemento, passado como parâmetro, na pilha Topo também passada como parâmetro. O parâmetro DeuCerto? deve indicar se a operação foi bem sucedida ou não. */

Variável auxiliar P tipo Ponteiro

P = GetNode

Info(P) = Elemento

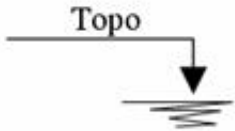
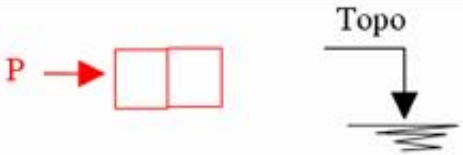
Next(P) = Topo

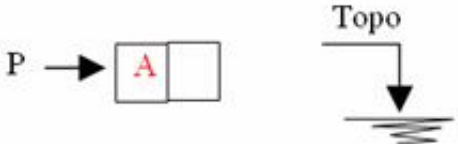
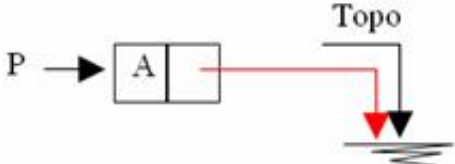
Topo = P

DeuCerto = Verdadeiro

// pelo menos por enquanto, vamos considerar que essa operação sempre dará certo.

O primeiro passo, na operação empilha, é alocar memória para guardar o elemento que será inserido na pilha. Ou seja, reservar uma caixinha, ou um nó, para inserir o elemento. Isso fazemos com a operação `GetNode`. Depois inserimos o valor de X no campo `Info` do nó alocado (apontado por P), e por último ligamos o nó recém alocada na lista. Para isso, apontamos o `Next(P)` para o `Topo`, e apontamos o `Topo` para P. Veja na Figura 6.5 uma execução passo a passo da operação Empilha.

	<p>Situação Inicial. A pilha está vazia.</p>
	<p><code>P = GetNode</code></p> <p>Aloca um bloco de memória. P está apontando para ele.</p>

	<p><code>Info(P) = Elemento</code></p> <p>Coloca a informação no campo <code>Info</code> do nó apontado por P.</p>
	<p><code>Next(P) = Topo</code></p> <p>Aponta o campo <code>Next(P)</code> para <code>Topo</code>. Ou seja, nesse caso, para <code>Null</code>.</p>



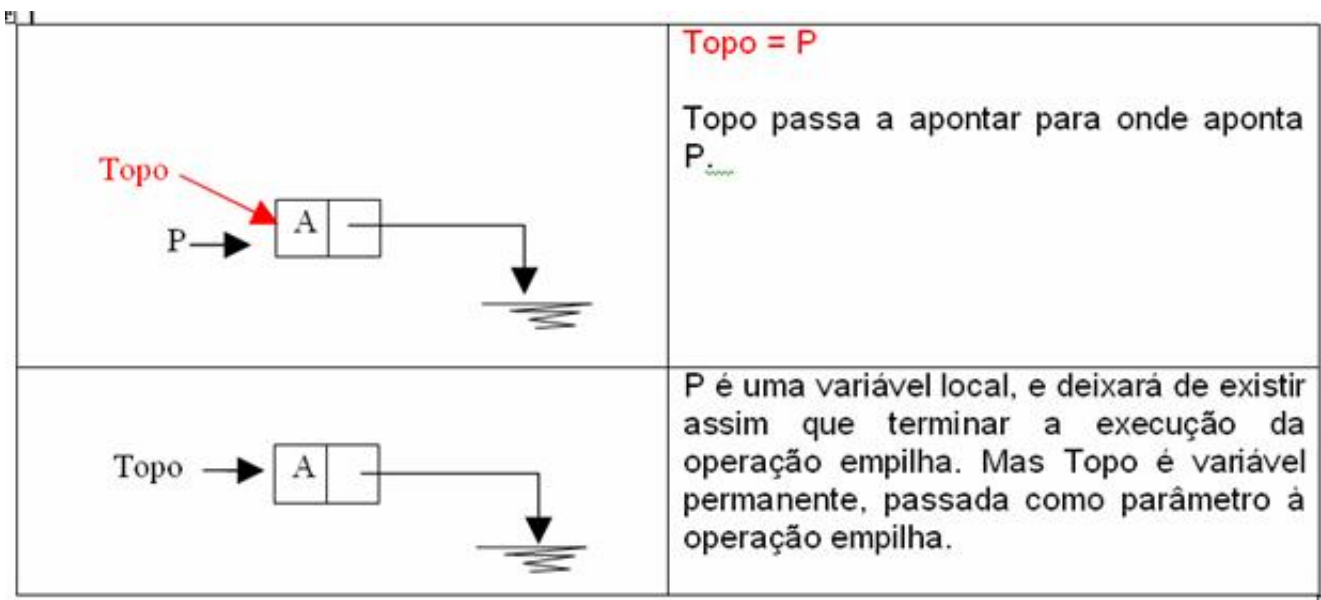


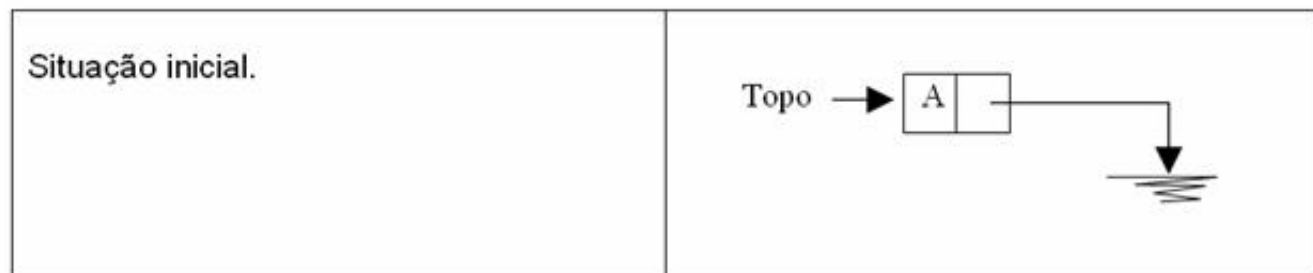
Figura 6.5 Execução da operação empilha

Simulação 6.3: Operação Empilha

Veja uma simulação da execução e uso da operação Empilha, interagindo com a [Simulação 6.3: Operação Empilha](#). **Atenção: clique com o mouse para avançar a apresentação.**

Exercício 6.4 Segunda Execução da Operação Empilha

Considerando a situação inicial a pilha com 1 elemento, conforme a figura, execute passo a passo o algoritmo da operação empilha, fazendo o desenho correspondente ao comando. Esse tipo de exercício é muito importante para desenvolver a habilidade para a elaboração de algoritmos sobre listas encadeadas.



<p>P = GetNode</p>	
--------------------	--

<p>Info(P) = Elemento</p> <p>Considere que o valor da variável Elemento é 'B'.</p>	
<p>Next(P) = Topo</p>	
<p>Topo = P</p>	
<p>Situação final, sem a variável temporária P.</p>	

Dica Importante

A frase “entendeu ou quer que eu desenhe?” faz muito sentido no contexto da disciplina estruturas de dados. É muito mais fácil entender olhando o desenho do que está acontecendo. A partir de agora, sempre que for realizar um algoritmo sobre listas encadeadas, faça o desenho passo a passo.

Simulação 6.4: Segunda Execução da Operação Empilha

Veja uma simulação da segunda execução da operação Empilha, interagindo com a [Simulação](#)

6.4: Segunda Execução da Operação Empilha. **Atenção:** clique com o mouse para avançar a apresentação.

No exercício 6.5 (operação desempilha), tente elaborar e testar o algoritmo assim: fazendo o desenho passo a passo. Você encontrará uma solução (apenas o algoritmo) no final da Unidade, mas não olhe a solução. Procure realmente fazer, e depois olhar a solução. É assim que se aprende. Não esqueça de testar (e desenhar) para várias situações. Pelo menos para uma pilha com vários elementos, uma pilha com 1 único elemento, e para uma pilha vazia.

Exercício 6.5 Desempilha

Desempilha(parâmetro por referência Topo do tipo Pilha, parâmetro por referência Elemento do tipo Char, parâmetro por referência DeuCerto? do tipo Boolean)

/* Desempilha (retira o elemento do topo), retornando o valor do elemento que foi desempilhado no parâmetro Elemento. O parâmetro DeuCerto? deve indicar se a operação foi bem sucedida ou não. A operação só não será bem sucedida se tentarmos retirar um elemento de uma pilha vazia. */

Implementação de Fila Através do Conceito de Listas Encadeadas

Assim como fizemos com uma pilha, vamos primeiramente representar (desenhar) uma fila, através das convenções e conceitos relacionados a listas encadeadas – Figura 6.5.

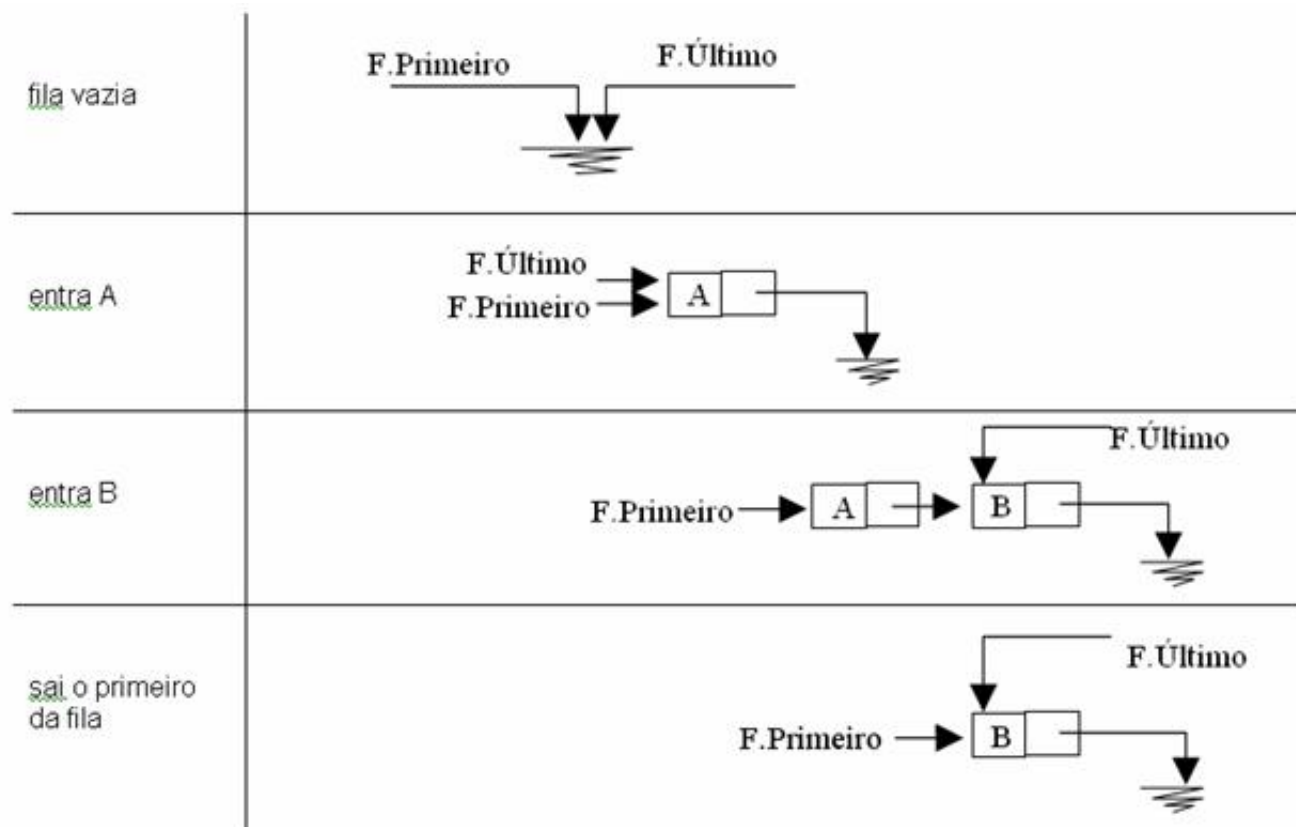


Figura 6.6 Representação de uma Fila Como uma Lista Encadeada

Note que agora não temos um único ponteiro definitivo para a lista encadeada (como na pilha, que tínhamos um único ponteiro, chamado *Topo*). Temos agora dois ponteiros: *Primeiro*, e *Último*, que deverão apontar, respectivamente, para o primeiro e para o último elemento da fila. O tipo *Fila* pode ser considerado, então, um registro, contendo 2 campos: *primeiro*, e *último*, ambos do tipo ponteiro. A referência a eles fica sendo, então, *F.Primeiro*, e *F.Último*.

Exercício 6.6 Operações Sobre Filas

Vamos implementar as operações sobre filas definidas na Unidade 5, agora utilizando o conceito de listas encadeadas:

- **Entra(QualFila, QualElemento, DeuCerto?):** Insere o elemento passado como parâmetro *QualElemento*, na fila passada no parâmetro *QualFila*. O elemento deve ser inserido no final da fila, ou seja, passando a ser o último elemento da fila. O parâmetro *DeuCerto?* indica se a operação foi bem sucedida ou não.
- **Sai(QualFila, QualElemento, DeuCerto?):** Retira o primeiro elemento da fila passada no parâmetro *QualFila*, retornando o valor do elemento que foi retirado no parâmetro *QualElemento*. O parâmetro *DeuCerto?* indica se a operação foi bem sucedida ou não.
- **Vazia?(QualFila):** Verifica se a fila passada como parâmetro (*QualFila*) está ou não vazia (vazia = sem nenhum elemento).
- **Cheia?(QualFila):** Verifica se a fila passada como parâmetro (*QualFila*) está ou não cheia (cheia = situação em que é impossível inserir mais elementos na fila). Nessa implementação, considere (por enquanto) que a fila nunca ficará cheia. Ou seja, por

enquanto, retorne sempre falso nessa operação.

- **Cria(QualFila)**: Cria uma fila, iniciando sua situação como vazia.

Não se esqueça de fazer os desenhos, passo a passo. Fazendo os desenhos, terão melhores chances de acertar a solução. Você encontrará uma possível solução no final da Unidade. Mas não olhe a resposta antes de fazer.

Revisão: Comparação Entre Alocação Seqüencial e Alocação Encadeada

Nas unidades 3 e 5 implementamos uma pilha e uma fila usando alocação seqüencial. Agora, na Unidade 6, utilizamos alocação encadeada. Veja uma comparação de características, na Tabela 6.4.

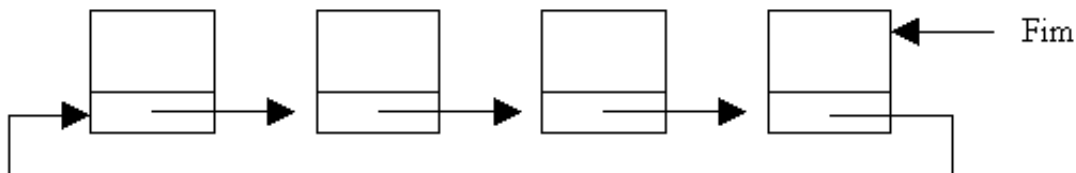
Alocação Seqüencial	Alocação Encadeada
<ul style="list-style-type: none">· Elementos são alocados em seqüência;· Seqüência física;· Seqüência implicitamente indicada.	<ul style="list-style-type: none">· Elementos não estão necessariamente em posições de memória adjacentes;· Seqüência lógica, virtual ou explicitamente indicada.

Tabela 6.4 Comparação entre alocação seqüencial e encadeada

Lembram-se do problema inicialmente colocado: como 120.000 estruturas de armazenamento podem compartilhar um “banco de memória” de 1.000 posições? A alocação encadeada nos permite isso: compartilhar memória.

Exercícios de Fixação

1. [Qual a diferença entre alocação seqüencial e alocação encadeada?](#)
2. Na sua opinião, [quais são as vantagens](#) de se utilizar alocação encadeada para um conjunto de elementos? Quais as possíveis desvantagens?
3. Faça diagramas ilustrando uma [pilha encadeada](#) e explique sucintamente o funcionamento
4. Faça diagramas ilustrando uma [fila encadeada](#) e explique sucintamente o funcionamento
5. Em uma LISTA CIRCULAR ENCADEADA, o último nó aponta para o primeiro (e não para NULL). Dessa forma, se queremos implementar uma fila, basta um ponteiro para o FIM, pois o COMEÇO será o seu próximo. Implemente uma fila circular, com as operação: cria, vazia?, entra, e sai.



6. Idem ao exercício anterior, mas para uma pilha.

Solução para o Exercício 6.5

Desempilha(parâmetro por referência Topo do tipo Pilha, parâmetro por referência Elemento do tipo Char, parâmetro por referência DeuCerto? do tipo Boolean)

```
/* Desempilha (retira o elemento do topo), retornando o valor do elemento que foi desempilhado
no parâmetro Elemento. O parâmetro DeuCerto? deve indicar se a operação foi bem sucedida
ou não. A operação só não será bem sucedida se tentarmos retirar um elemento de uma pilha
vazia. */
```

Variável auxiliar P tipo Ponteiro

Se Vazia(Topo) == Verdadeiro

Então DeuCerto = Falso

Senão DeuCerto = Verdadeiro

Elemento = Info(Topo)

P = Topo

Topo = Next(Topo)

Freenode(P)

Solução para o Exercício 6.6

Tipo Fila = Registro

Primeiro, Último: Ponteiro

Cria(F)

F.Primeiro = Null

F.Último = Null

Boolean Vazia(F)

se F.Primeiro = Null { ou F.Último }

então resposta = true

senão resposta = false

Entra(F, X)

Variável auxiliar P tipo ponteiro

P = GetNode

Info(P) = X

Next(p) = Null

Se F.Último = Null { se a fila está vazia...}

Então F.Primeiro = p { ..o nó que entra será o último e também o primeiro }

Senão Next(F.Último) = P { ..o nó que entra será o último, e o próximo do que }

F.Último = P { ... era o último }

Sai(F retorna X)

Variável auxiliar Aux tipo ponteiro

Se vazia(F)

Então não faz nada

Senão X = Info(F.Primeiro)

Aux = F.Primeiro

F.Primeiro = Next(F.Primeiro)

Se F.Primeiro = Null

Então F.Último = Null

FreeNode (Aux)

{ guarda o Primeiro }

{ se era o único elemento... }

{ .. o Primeiro E o Último vão para Null }