Listas com política de acesso: Pilhas, Filas e Deques

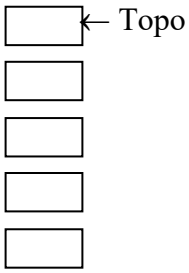
Formas restritas de listas onde elementos só podem ser inseridos e removidos em posições pré-determinadas.

Pilhas: inserções e remoções sempre no mesmo lado
LIFO (*last in first out*)
ou, os elementos saem da pilha na ordem reversa de chegada
Filas: inserções no fim e remoções no início
FIFO (*first in first out*)
ou, os elementos saem da fila na ordem de chegada

Pilhas

A referência para inserções e remoções de elementos é o Topo da pilha.

Operações: Empilhar (*push*): inserir elemento no topo
Desempilhar (*pop*): retirar elemento do topo



- menos flexível
- mais eficiente e fácil de implementar
- certas aplicações exigem pilhas

T.A.D. Pilha:

TAD Pilha {
Dados: topo
Operações:
novaPilha ()
empilha(E: elemento)
desempilha(S: elemento)
vazia(S: lógico)
}

Pilhas baseadas em vetores:

Simplificação de listas baseadas em vetores.

topo = primeira posição livre (ou, última ocupada)

Pilha vazia:

topo = 1

1	2	3	4	5	6

Uma pilha com três elementos:

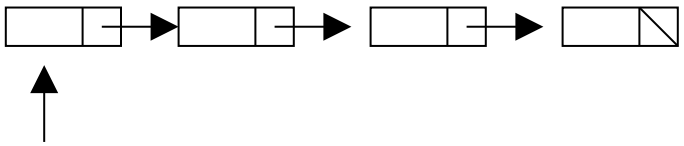
topo = 4

1	2	3	4	5	6
43	67	98			

Exercício: implementar os procedimentos de empilhar e desempilhar para uma pilha cujos elementos têm tamanho variável (ocupam um número variável de espaços na pilha).

Pilhas implementadas como listas encadeadas:

Simplificação de listas encadeadas.



topo

Não é necessário nodo cabeça.

Exercício: propor implementações para as operações de empilhar e desempilhar considerando pilhas com encadeamento simples.

Comparando as implementações:

Única base para comparações: espaço necessário

- vetor: tamanho fixo no início.

- encadeamento: espaço para apontadores

Exercício: escreva e implemente procedimentos empilhar e desempilhar para 2 pilhas implementadas em um único vetor.

Exercício: Escreva uma operação para esvaziar uma pilha implementada por encadeamento simples.

Aplicações de pilhas

1. **Controle de chamada de subrotinas:**

Aplicação mais comum para pilhas. Toda a informação necessária para o funcionamento de uma subrotina é colocada em uma pilha: empilhamento do registro de ativação.

- cada chamada de subrotina adiciona um registro à pilha
- cada retorno remove um registro.

2. **Recursão:**

O processo de recursão também é controlado com o uso de pilhas de registros de ativação (como na chamada de subrotinas).

- pode ser ineficiente porque cada chamada implica na criação e empilhamento de um registro.
- algoritmos recursivos podem ser simulados com o uso de pilhas explícitas.

3. **Interpretador de expressões aritméticas**

Esse interpretador aplica-se a expressões completamente parentizadas, e considera apenas os operadores **+, -** e *****, e os operandos **números**.

A partir de uma expressão já digitada corretamente e completamente parentizada, o interpretador faz a leitura caracter a caracter (sendo por caracter, os números considerados são formados por apenas um dígito, e inteiros), sendo seu funcionamento definido da seguinte forma:

- componentes: números inteiros, parênteses e operadores +, - e *****
- o que utilizar: duas pilhas, uma para os operandos e outra para os operadores
- o que fazer a partir do reconhecimento de:
  - **(**: não fazer nada
  - **número**: coloca-o na pilha de operandos
  - **operador (+, - ou *)**: coloca-o na pilha de operadores
  - **)**: calcula sub-expressão, com os dois últimos números colocados na pilha de operandos e o último sinal de operação colocado na pilha de operadores, colocando o resultado na pilha de operadores.

Exemplo:

expressão	pilha de operandos	pilha de operadores
(3*(4+5))		
(3*(4+5))	3	
(3*(4+5))	3	*
(3*(4+5))	3	*
(3*(4+5))	3 - 4	*
(3*(4+5))	3 - 4	* +
(3*(4+5))	3 - 4 - 5	* +
(3*(4+5))	3 - 9	*
(3*(4+5))	27	

Filas

Também uma forma restrita de lista: elementos só podem ser inseridos no final e removidos do início da lista.

T.A.D. Fila:

```
TAD Fila {
        elemento
Operações:    novaFila ()
              enfileira(E: elemento)
              desenfileira(S: elemento)
              vazia(S: lógico)
}
```

Implementação de fila baseada em vetores:

Implementação ineficiente se usarmos sempre os "n" primeiros espaços para uma fila de "n" elementos.

Implementação mais eficiente: não movemos elementos, a fila move-se circularmente no vetor: quando atingimos o final do vetor a fila move-se do índice mais alto para o mais baixo.

Ex: considerando a fila implementada no vetor abaixo:

Situação de fila vazia:

Início = 1 Fim = 0

1	2	3	4	5	6

inserimos os elementos (12,43,56,78,90) nesta seqüência.

Início = 1 Fim = 5

1	2	3	4	5	6
12	43	56	78	90	

Retiramos agora dois elementos do início da fila: 12 e 43 (nesta seqüência).

Início = 3 Fim = 5

1	2	3	4	5	6
		56	78	90	

Inserimos o valor 45 na fila

Início = 3 Fim = 6

1	2	3	4	5	6
		56	78	90	45

Inserimos o valor 82 na fila

Início = 3 Fim = 7 ? como há espaços livres no início do vetor podemos utilizá-los, portanto o último espaço ocupado passa a ser o de índice 1.

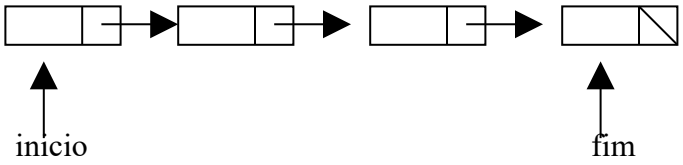
Fim = 1

1	2	3	4	5	6
82		56	78	90	45

Exercício: Propor implementações para as operações de inserir e remover elementos de filas implementadas com vetores.

Implementação de fila baseada em listas encadeadas:

Adaptação de listas encadeadas.



Não é necessário nodo cabeça.

Situação inicial:

início ← nulo
fim ← nulo

Aplicações:

Em Sistemas Operacionais: filas de processos para serem executados (uma fila de impressão é um exemplo de um processo).

Exercício: Considere a implementação de uma fila usando uma lista com encadeamento simples. Escreva uma operação que inverte as posições na fila.

Deques (ou dequeues)

Também uma forma restrita de lista ("Double-Ended Queue"): elementos só podem ser acessados, inseridos e retirados no início e/ou no final da lista. Pode-se estabelecer restrições:

- inserções só podem ser realizadas em uma das extremidades e remoções nas duas (**deque com entrada restrita**), ou
- inserções podem ser realizadas nas duas extremidades e remoções apenas em uma (**deque com saída restrita**).

T.A.D. Deque:

```
TAD Deque{
Dados: ladoA, ladoB
Operações:    novaDeque ()
              incluiInicio(E: elemento)
              incluiFinal(E: elemento)
              excluiInicio(S: elemento)
              excluiFinal(S: elemento)
              vazia(S: lógico)
}

TAD DequeEntradaRestrita{
Dados: ladoA, ladoB
Operações:    novaDequeEntradaRestrita ()
              inclui(E: elemento)
              excluiInicio(S: elemento)
              excluiFinal(S: elemento)
              vazia(S: lógico)
}

TAD DequeSaidaRestrita{
Dados: ladoA, ladoB
Operações:    novaDequeSaida Restrita ()
              incluiInicio(E: elemento)
              incluiFinal(E: elemento)
              exclui(S: elemento)
              vazia(S: lógico)
}
```