Lição 2

Stack

Objetivos

Ao final desta lição, o estudante será capaz de:

- Explicar os conceitos básicos e operações em *stack* ADT
- Implementar uma *stack* ADT usando representação seqüencial e de ligação
- Discutir aplicações de *stack*: Os problemas de reconhecimento de padrões e conversões do tipo **infix** para **postfix**
- Explicar como múltiplas *stack* podem ser armazenadas utilizando *array* de uma dimensão
- Realocação de memória durante um transbordamento (estouro) de um *array* com múltiplas *stack* utilizando algoritmos **unit-shift policy** e **Garwick's**

Introdução

- “O último a entrar é o primeiro a sair” (LIFO)
- As operações são sempre no topo da *stack* e não temos acesso aos outros elementos
- Operações básicas: *push* e *pop*

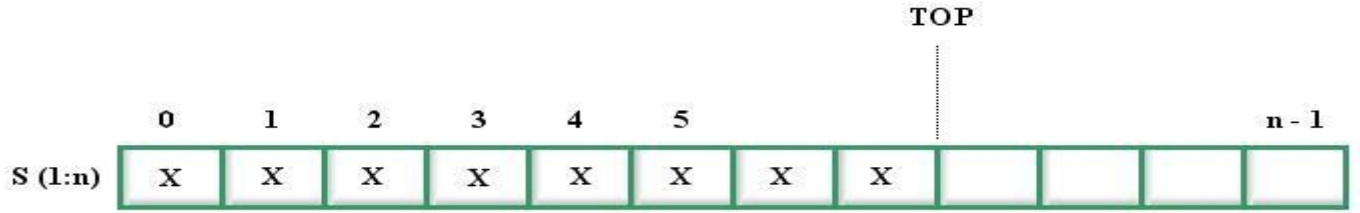
Operações

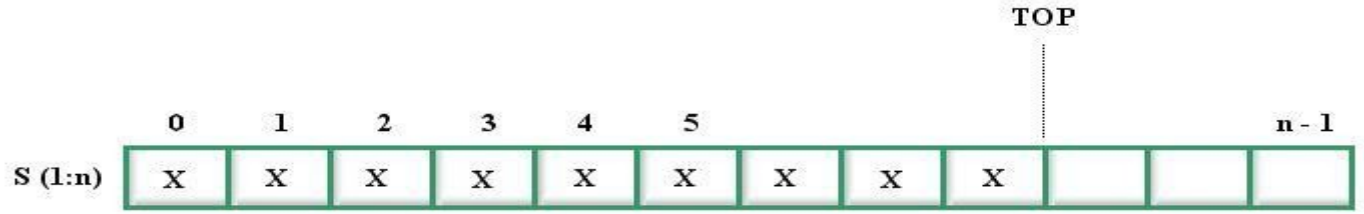
- Representação: seqüencial ou encadeada

- Operações comuns as *stacks*:
  - Verificar o tamanho
  - Verificar se está vazia
  - Obter o elemento do topo sem retirá-lo
  - push - Inserir um novo elemento na *stack*
  - pop - Retirar um elemento do topo da *stack*

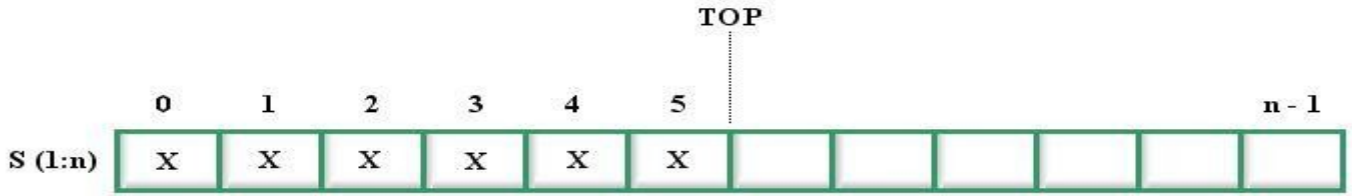
Operações

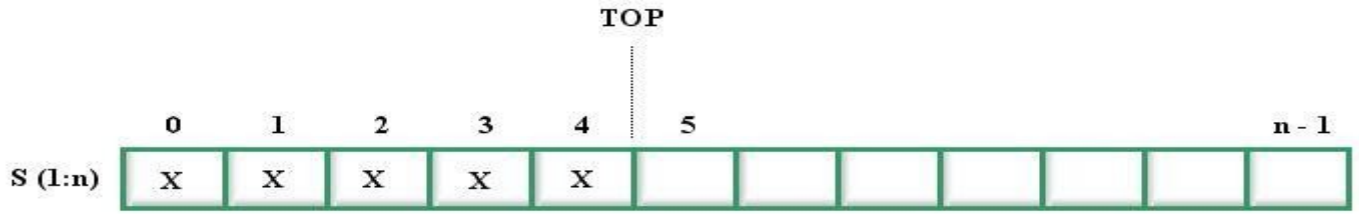
- *push*





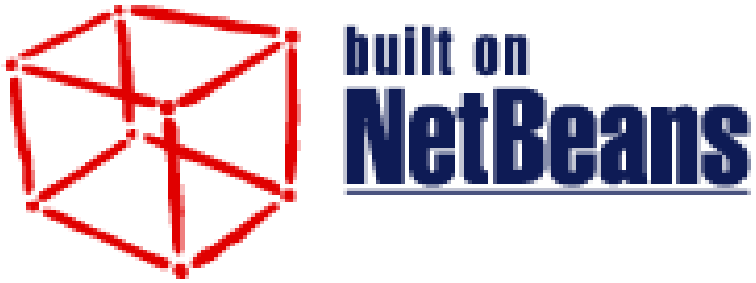
- *pop*





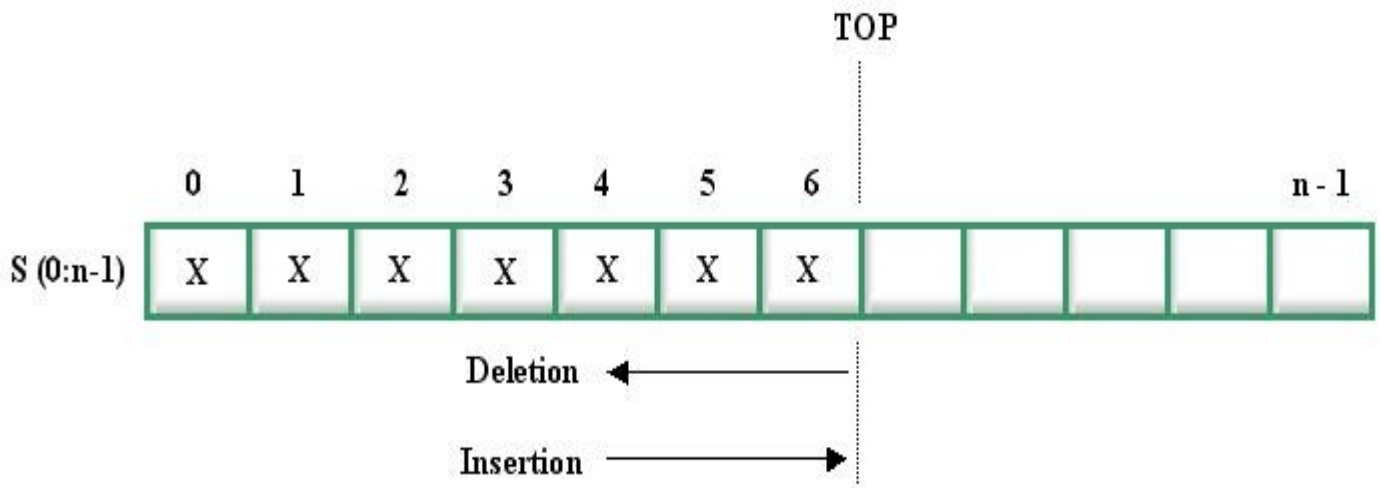
Operações

- Passaremos agora para o NetBeans



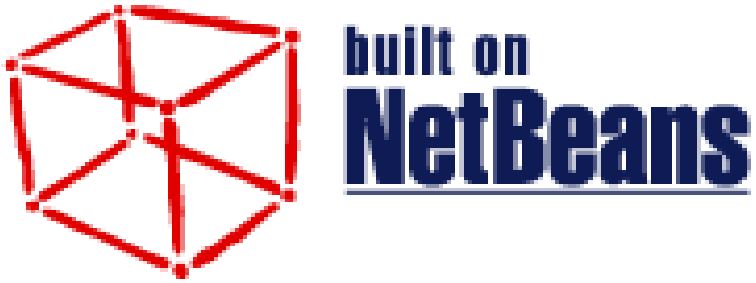
Implementações: Representação seqüencial

- Usando *arrays*
- *Stack* vazia se top=-1 e cheia se top=n-1
- Retirar elemento de uma *stack* vazia causa *underflow*
- Inserir elemento em uma *stack* cheia causa *overflow*



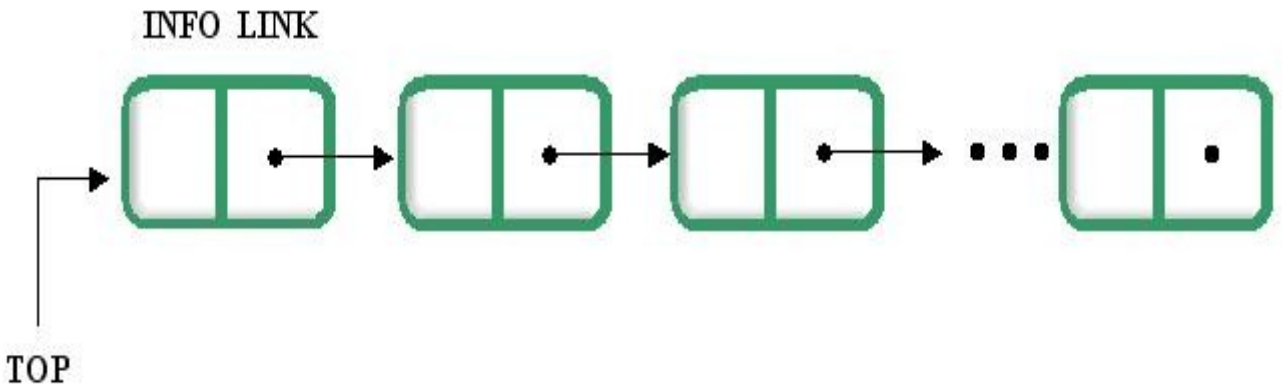
Implementações: Representação seqüencial

- Passaremos agora para o NetBeans



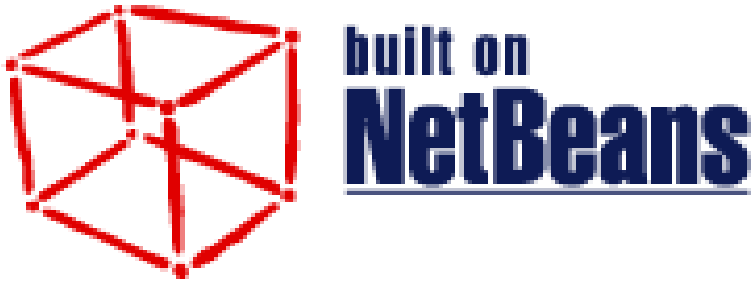
Implementações: Representação encadeada

- Listas encadeadas de *nodes* de *stack* podem ser utilizadas



Implementações: Representação encadeada

- Passaremos agora para o NetBeans



Aplicação: problema de reconhecimento de padrões

- Dado o conjunto L = { wcw^R | w $\subset$ { a, b }+ }
  1. Pegue o próximo caractere a ou b da *string* de entrada e insira na pilha; repita até o símbolo c ser encontrado
  2. Pegue o próximo caractere a ou b da *string* de entrada, abra a *stack* e compare. Se os dois símbolos batem, continue, caso contrário, pare – a string não está em L

  Estado adicional quando a *string* não está em L:

  1. O fim da *string* foi atingido mas o c não foi encontrado
  2. O fim da *string* foi atingido mas a *stack* não está vazia
  3. A *stack* está vazia mas o fim da *string* não foi atingido ainda

Aplicação: problema de reconhecimento de padrões

Entrada	*Ação*	*Stack*
abbabcbabba	------	(bottom) --> (top)
abbabcbabba	Push a	a
bbabcbabba	Push b	ab
babcbabba	Push b	abb
abcbabba	Push a	abba
bcbabba	Push b	abbab
cbabba	Discard c	abbab
babba	Pop, compare b and b --> ok	abba
abba	Pop, compare a and a --> ok	abb
bba	Pop, compare b and b --> ok	ab
ba	Pop, compare b and b --> ok	a
a	Pop, compare a and a --> ok	-
-	**Success**	

Aplicação: Infix para Postfix

- Forma **infix** – operando-operador-operando
- Forma **postfix** – operando-operando-operador
- Propriedades:
  - Grau do operador
  - Rank de um operando
  - Se z = x | y é uma string, então x é o topo de z. x é o próprio topo se y não é uma string nula

Aplicação: Infix para Postfix

- Teorema: uma expressão postfix é bem moldada se o rank de todos os topos são maiores ou igual a 1 e o rank da expressão é 1

Operador	*Prioridade*	*Propriedade*	*Exemplo*
^	3	Associação a direita	a^b^c = a^(b^c)
* /	2	Associação a esquerda	a*b*c = (a*b)*c
+ -	1	Associação a esquerda	a+b+c = (a+b)+c

Aplicação: Infix para Postfix

- Regras para converter infix para postfix:

1. A ordem dos operandos nas duas formas são as mesmas se os parênteses estiverem ou não presentes na expressão infix.
2. Se a expressão infix não contém parênteses, então a ordem dos operadores na expressão postfix está de acordo com sua prioridade.
3. Se a expressão infix contém sub-expressões em parênteses, a regra 2 se aplica do mesmo modo para as sub-expressões.

Aplicação: Infix para Postfix

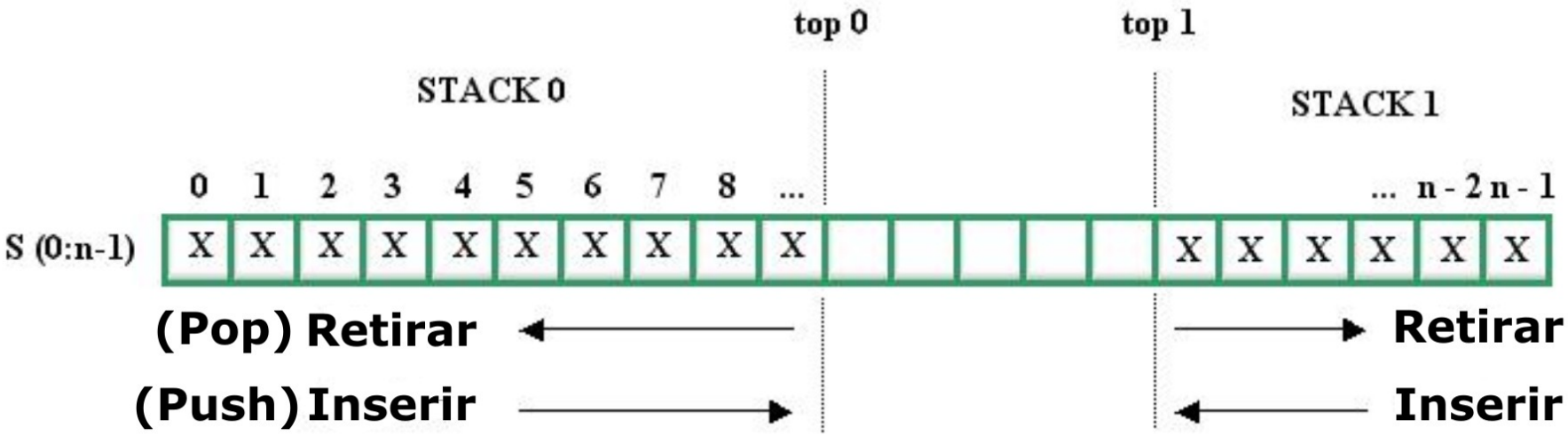
- Números prioritários:

Token, x	*icp(x)*	*isp(x)*	*Rank*
Operando	0	-	1
+ -	1	2	-1
* /	3	4	-1
^	6	5	-1
(	7	0	-

Aplicação: Infix para Postfix

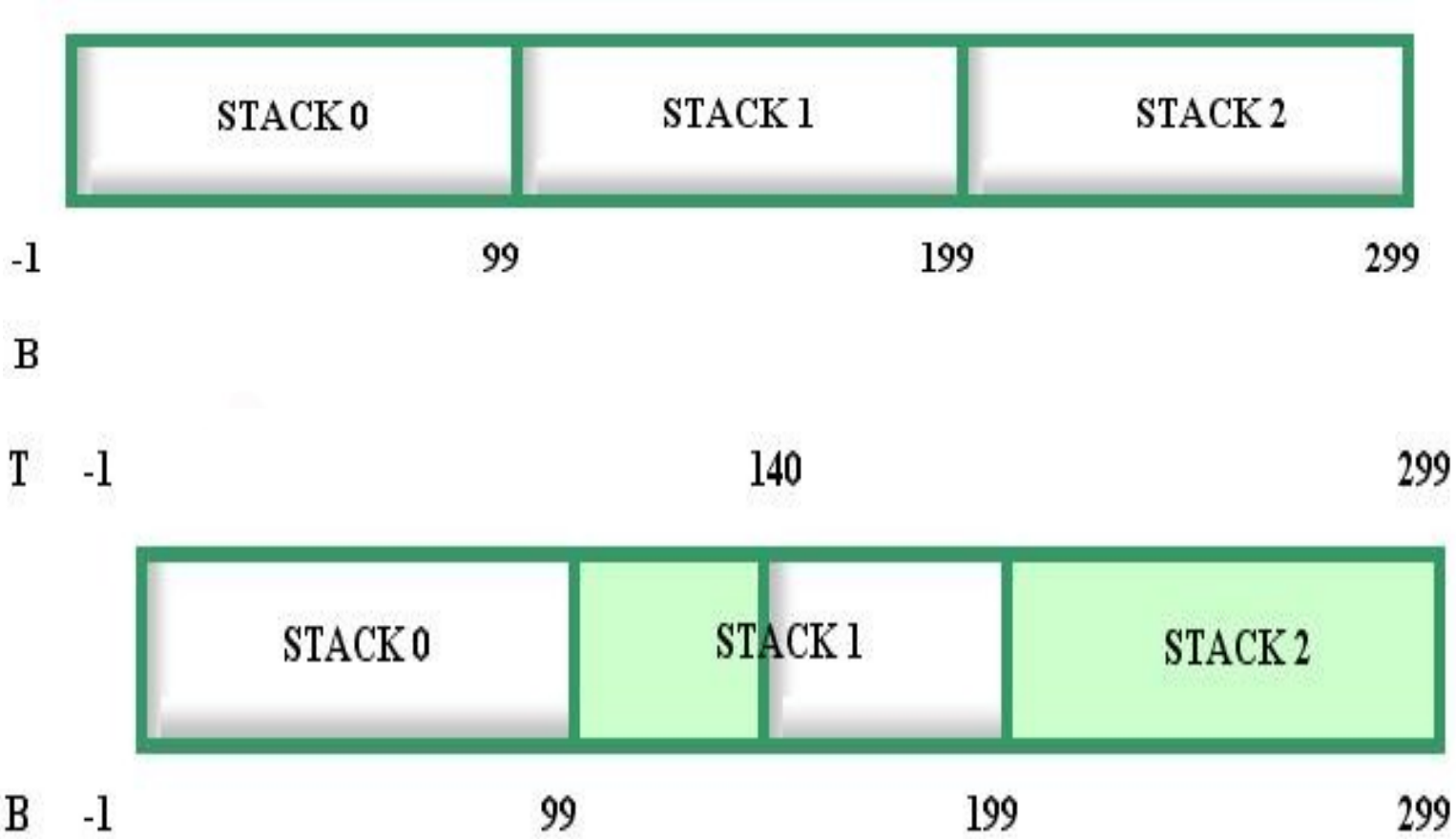
1) Pega o próximo símbolo (Token) x
2) Se x é operando então sai x
3) Se x é (, então insere x na *stack*
4) Se x é), então retira elementos da *stack* até que "(" seja encontrado, mais uma vez apagar o "(", Se topo = 0, o algoritmo termina
5) Se x é um operador então enquanto icp(x) < isp(stack(top)), sai elementos da *stack*; caso contrário; se icp(x) > isp(stack(top)), então insere x na pilha
6) Retorna ao passo 1

Tópico avançado: múltiplas *stacks*



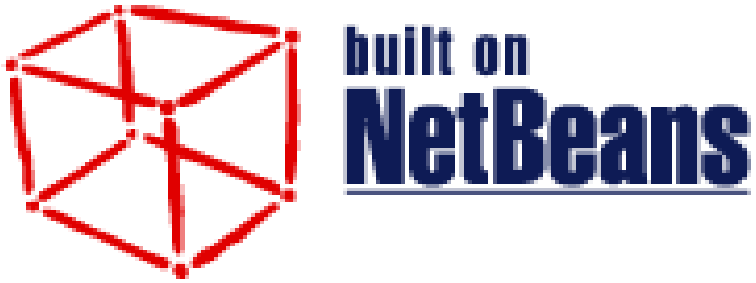
Tópico avançado: múltiplas *stacks*

- Múltiplas *stacks* em um *array* unidimensional



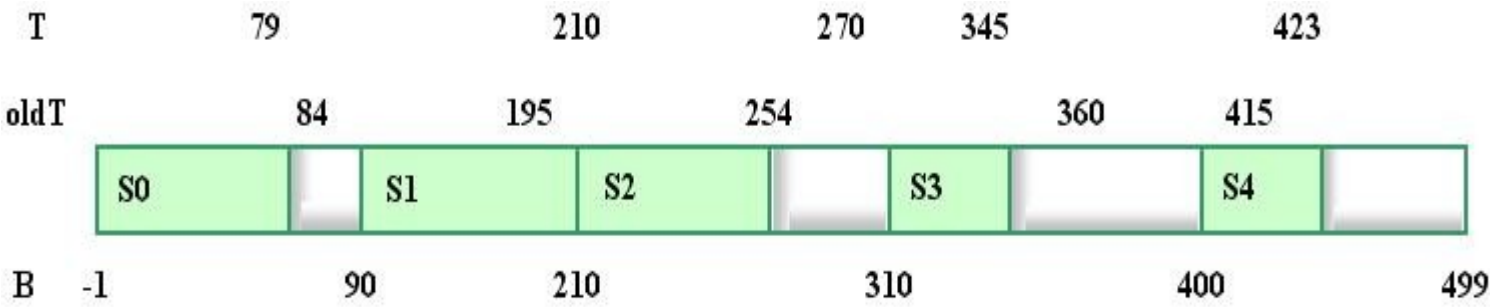
Tópico avançado: múltiplas *stacks*

- Passaremos agora para o NetBeans



Tópico avançado: múltiplas *stacks*

Realocação de memória numa pilha sobrecarregada
Algoritmo de Garwick (Implementação de Knuth):



Stacks Antes da re-alocação

Stacks depois da Re-alocação

Sumário

- Operações
- Implementações
  - Representação seqüencial
  - Representação encadeada
- Aplicação
  - Problema do reconhecimento de padrões
  - *Infix* to *Postfix*
- Tópico avançado
  - Múltiplas *stacks*

Parceiros

- **Os seguintes parceiros tornaram JEDI™ possível em Língua Portuguesa:**