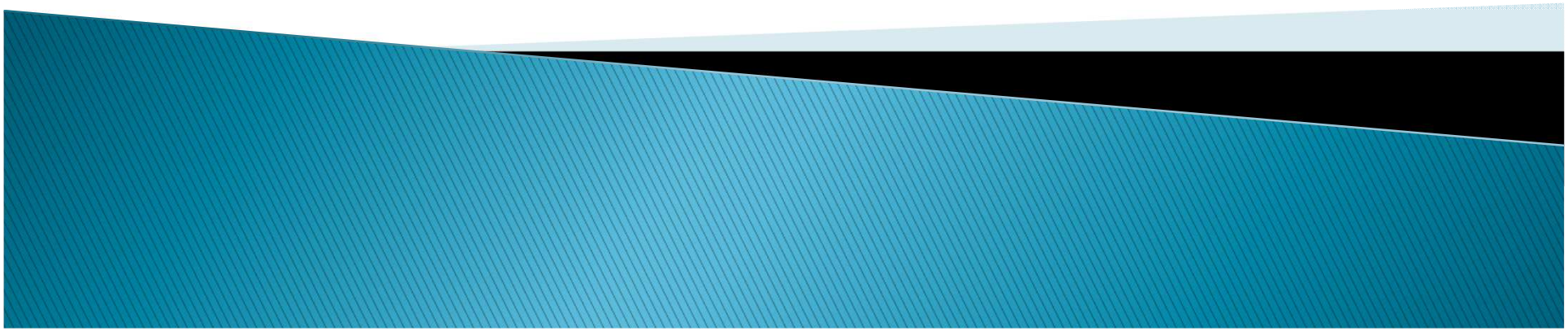
Estrutura de Dados

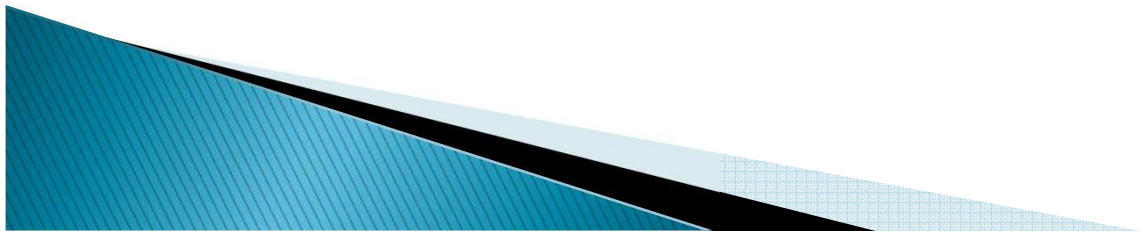
Pilhas

Prof. Jesus José de Oliveira Neto



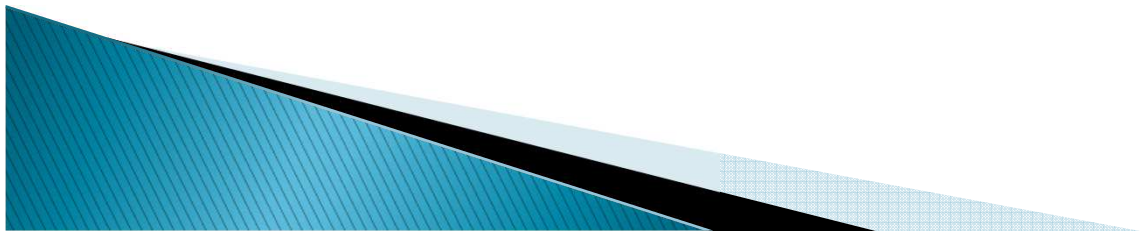
Pilhas

- É uma das estruturas de dados mais simples
- A idéia fundamental da pilha é que todo o acesso a seus elementos é feito através do seu topo.
- Assim, quando um elemento novo é introduzido na pilha, passa a ser o elemento do topo, e o único elemento que pode ser removido da pilha é o do topo.



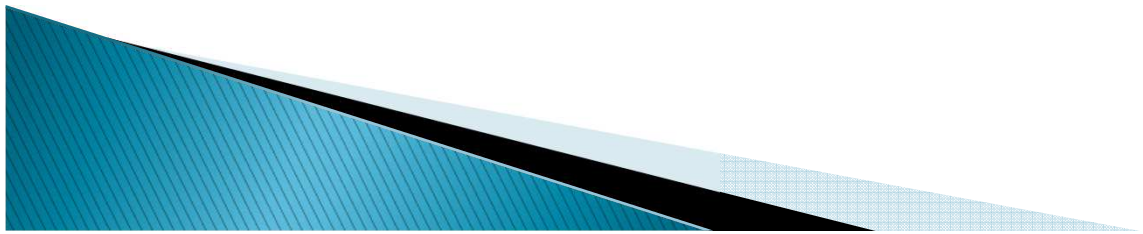
Pilhas
:: Aplicações

- Verificação de parênteses.
- Retirada de vagões de um trem.
- Retirada de mercadorias em um caminhão de entregas.



Pilhas

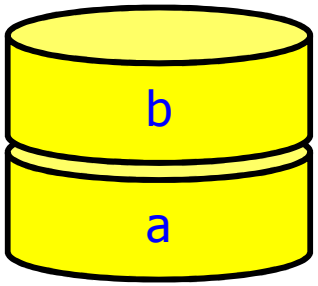
- Os elementos da pilha são retirados na ordem inversa à ordem em que foram introduzidos: o primeiro que sai é o último que entrou (LIFO – last in, first out).
- Existem duas operações básicas que devem ser implementadas numa estrutura de pilha:
  - operação para empilhar (push) um novo elemento, inserindo-o no topo,
  - operação para desempilhar (pop) um elemento, removendo-o do topo

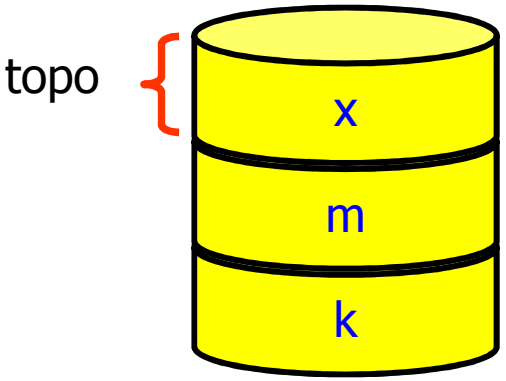


Pilhas

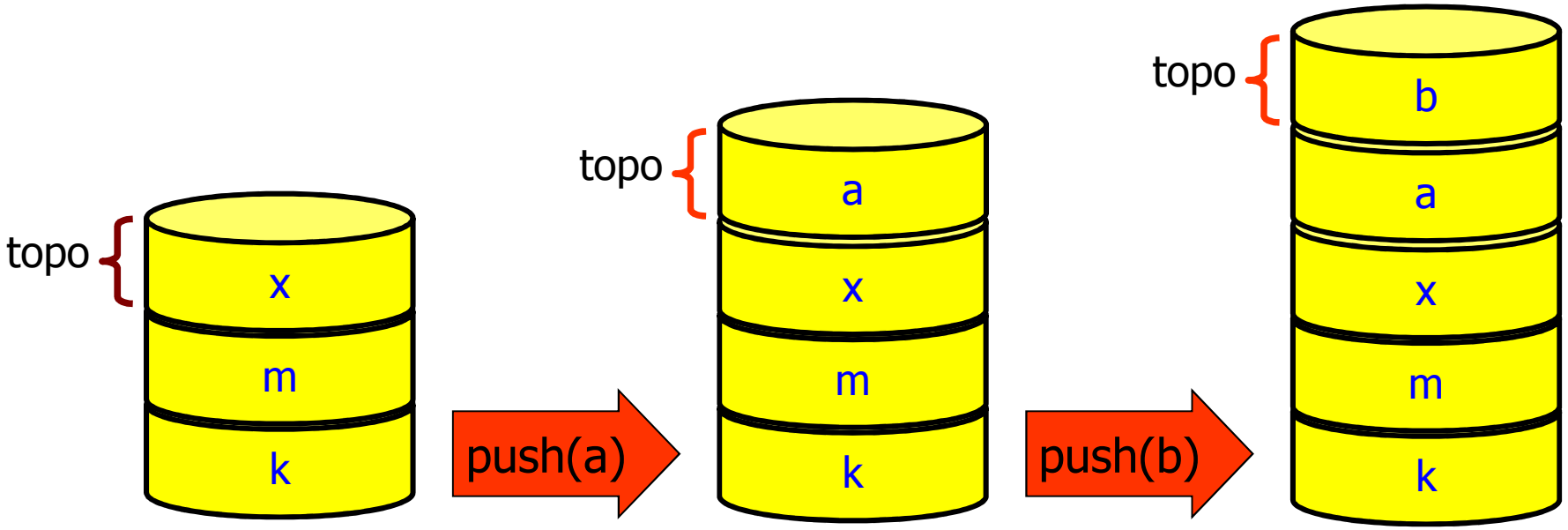
:: Push

push(a)
push(b)



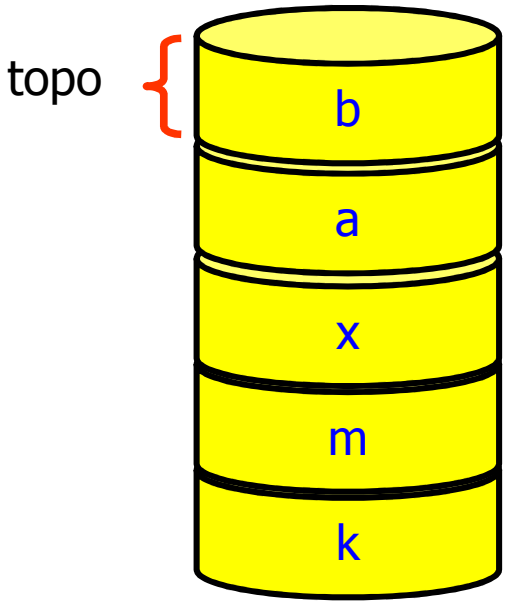


Pilhas
:: Push

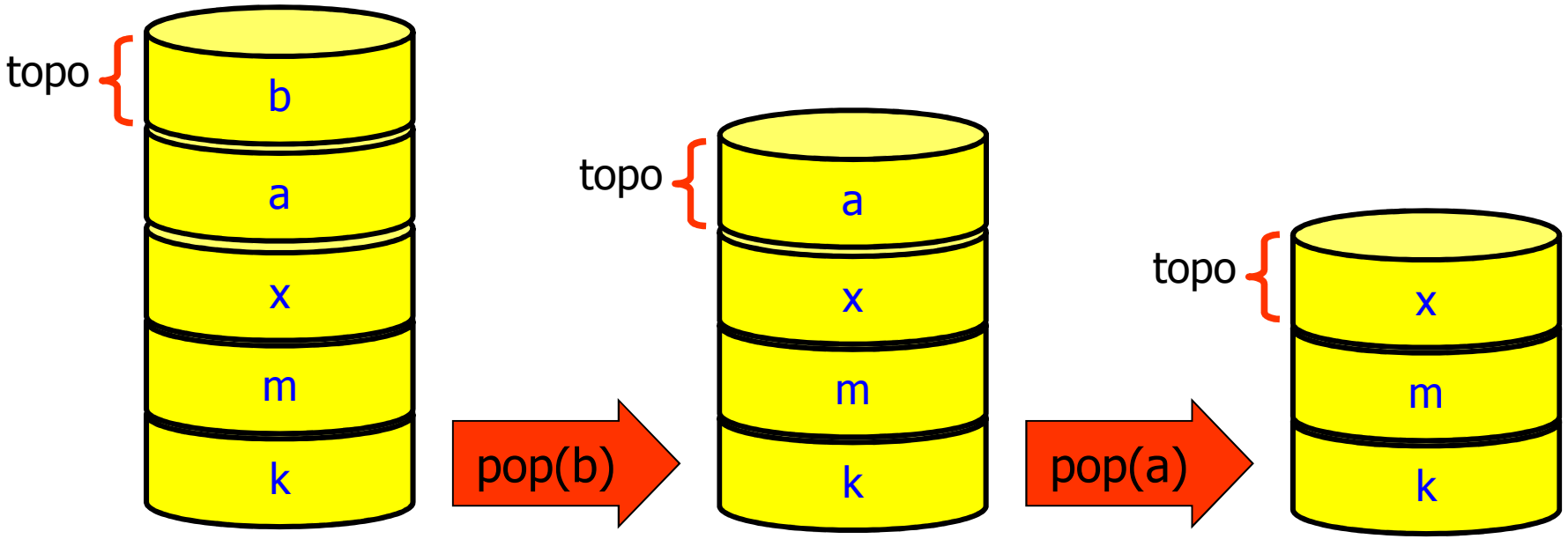


Pilhas
:: Pop

pop(b)
pop(a)

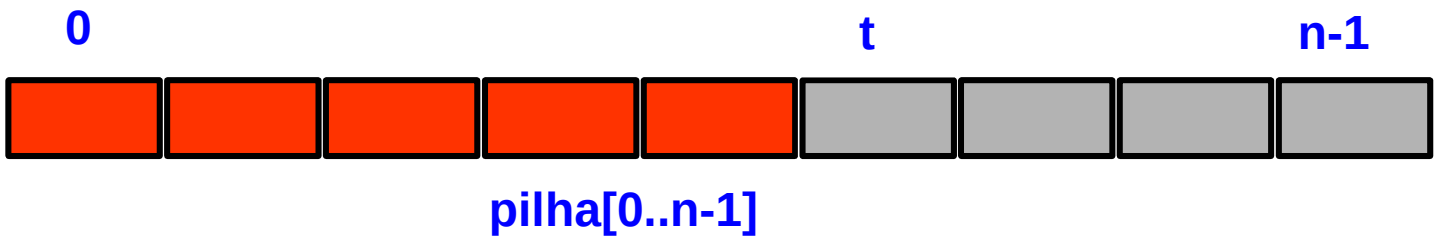


Pilhas
:: Pop



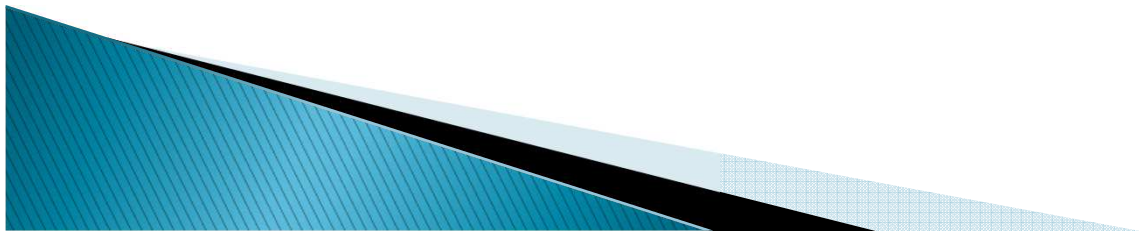
Pilhas
:: Implementação de pilha com vetor

- Supondo a pilha está armazenada em um vetor pilha[0..n-1].
- A parte do vetor ocupada pela pilha será:



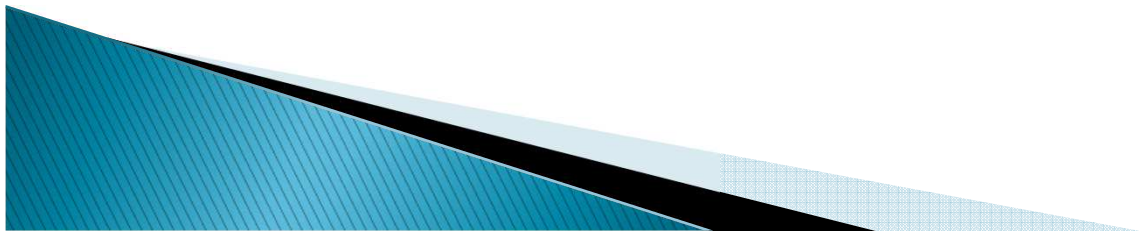
Pilhas
:: Implementação de pilha com vetor

```
public class Pilha {

  static final int MAX = 50;

  //Elementos adicionados
  static int tamanho;
  static double vet[] = new double[MAX];
  ...
}
```



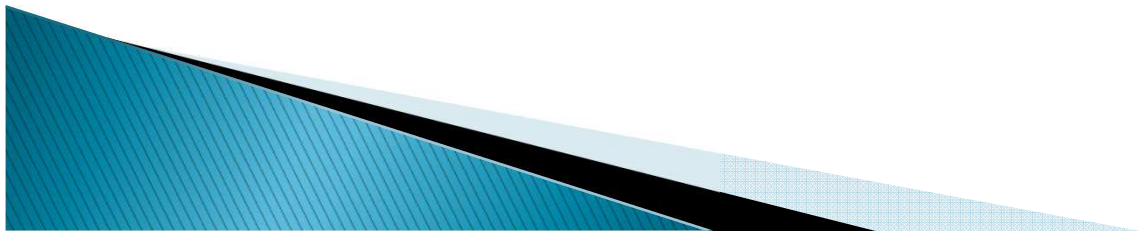
Pilhas
:: Inserir o elemento do topo – push()

```
static void push(double v)
{
  if(pilhaEstaCheia()){
    System.out.println("Pilha Cheia.");
    System.exit(1); /*aborta programa*/
  }
  /* insere elemento na próxima posição livre */
  vet[tamanho] = v;
  tamanho++;
}
```



Pilhas
:: Remover o elemento do topo – pop()

```
static double pop()
{
  double v;
  if (pilhaEstaVazia()) {
    System.out.println("Pilha vazia.");
    System.exit(1); /*aborta programa*/
  }
  /*retira elemento do topo*/
  v = vet[tamanho-1];
  tamanho--;
  return v;
}
```

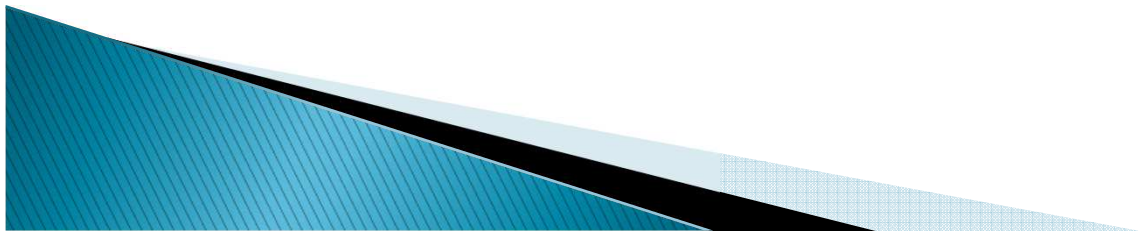


Pilhas
:: Imprime estrutura de pilha

```
static void imprime() {

  int i;

  for (i=tamanho-1; i>=0; i--)
  {
    System.out.println(vet[i]);
  }
}
```

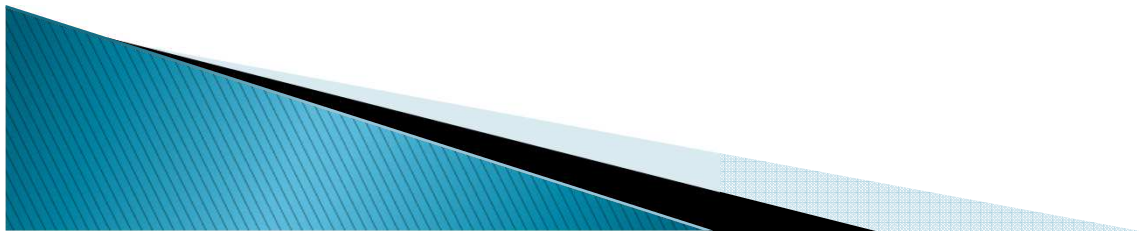
Pilhas
:: Verificar se a pilha está vazia

```
static boolean pilhaEstaVazia()
{
  return tamanho == 0;
}
```



Pilhas
:: Verificar se a pilha está cheia

```
static boolean pilhaEstaCheia()
{
  return tamanho == MAX;
}
```

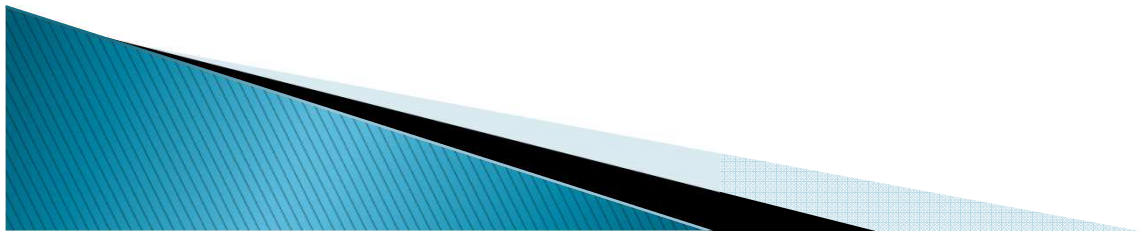


Pilhas
:: Teste

```
public static void main(String[] args) {

  push(p,20.0f);
  push(p,20.8f);
  push(p,20.3f);
  push(p,44.5f);
  push(p,33.3f);
  push(p,20.9f);
  imprime();
  System.out.println("Retirado: "+ pop());
  System.out.println("Retirado: "+ pop());
  System.out.println("Configuracao da fila:“);
  imprime();

}
```

Implementação de pilha dinâmica

- Quando o número máximo de elementos que serão armazenados na pilha não é conhecido, devemos implementar a pilha usando uma estrutura de dados dinâmica, no caso, empregando uma lista encadeada.
- Os elementos são armazenados na lista e a pilha pode ser representada simplesmente por um ponteiro para o primeiro nó da lista.

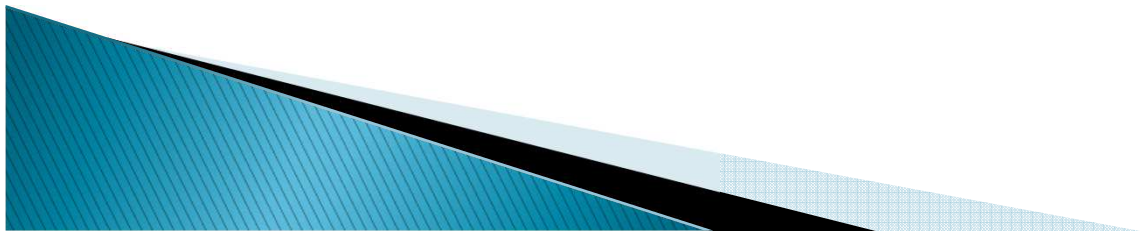


Pilhas
:: Implementação de pilha dinâmica

```
public class NoPilha {

   double info;
   NoPilha proximo;
   ...
}

public class PilhaDinamica
{
   NoPilha topo;
   ...
}
```

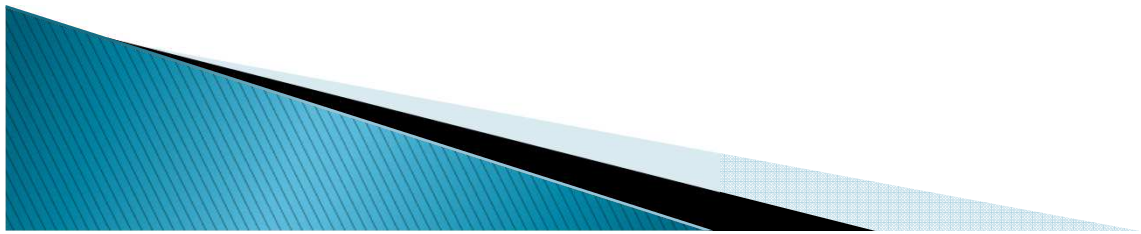
Pilhas Dinâmicas
:: Operações básicas

- Criar uma estrutura de pilha;
- Inserir um elemento no topo (push);
- Remover o elemento do topo (pop);
- Verificar se a pilha está vazia;
- Liberar a estrutura de pilha



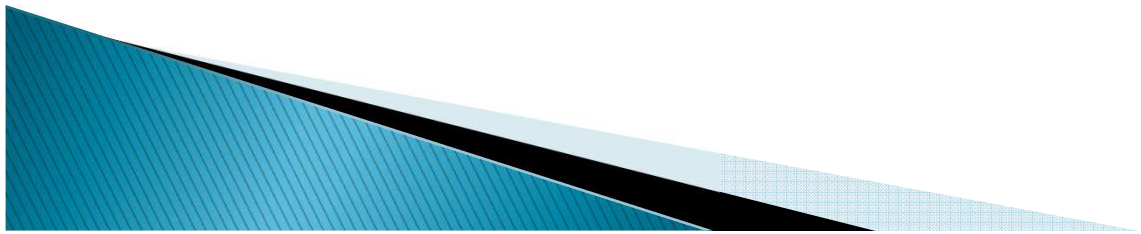
Pilhas Dinâmicas
:: Criar uma estrutura de pilha

```
public PilhaDinamica()
{
  this.topo = null;
}
```



Pilhas Dinâmicas
:: Inserir o elemento do topo – push()

```
public void push(double valor)
{
   NoPilha aux = new NoPilha(valor);
   aux.proximo = topo;
   topo = aux;
}
```

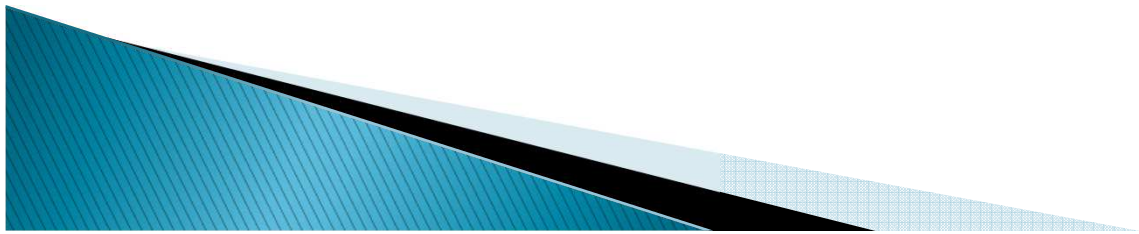


Pilhas Dinâmicas
:: Remover o elemento do topo – pop()

```
public double pop()
{
  double valor;
  NoPilha aux;
  if (pilhaEstaVazia())
  {
    System.out.println("Pilha vazia.");
    System.exit(1); /*aborta o programa*/
  }
  valor = topo.info;
  aux = topo;
  topo = aux.proximo;
  return valor;
}
```

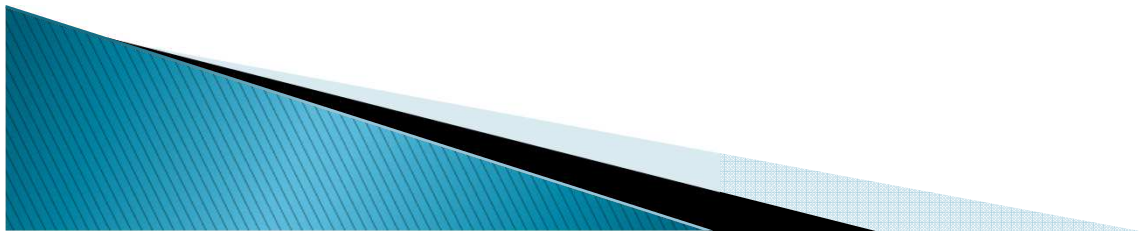
Pilhas Dinâmicas
:: Verificar se a pilha está vazia

```
public boolean pilhaEstaVazia()
{
  return (topo == null);
}
```



Pilhas Dinâmicas
:: Imprime estrutura de pilha

```
public void imprime() {

  for(NoPilha q=topo;q!=null;q=q.proximo)
  {
    System.out.println(q.info);
  }

}
```



Pilhas Dinâmicas
:: Teste

```
public static void main(String[] args) {
  PilhaDinamica p = new PilhaDinamica();
  p.push(20.0);
  p.push(20.8);
  p.push(20.3);
  p.push(44.5);
  p.push(33.3);
  p.push(20.9);
  p.imprime();
  System.out.println("Retirado: "+ p.pop());
  System.out.println("Retirado: "+ p.pop());
  System.out.println("Configuracao da pilha:\n");
  p.imprime();

  // libera memória
  p = null;
}
```

Exemplo de aplicação do uso da Pilha

- Método para verificar expressões matemáticas:
  - Considerando cadeias de caracteres com expressões matemáticas que podem conter termos entre parênteses, colchetes ou chaves, ou seja, entre os caracteres '(' e ')', ou '[' e ']', ou '{' e '}';
  - O método retorna **true**, se os parênteses, colchetes e chaves de uma expressão aritmética são abertos e fechados corretamente, ou **false** caso contrário;
  - Para a expressão "2*{3+4*(2+5*[2+3])}" o método deve retornar **true**;
  - Para a expressão "2*(3+4+{5*[2+3}])" o método deve retornar **false** ;

Exemplo de aplicação do uso da Pilha

- A estratégia para resolver esse problema é percorrer a expressão da esquerda para a direita:
  - Se encontra '(', '[' ou '{', empilha;
  - Se encontra ')', ']' ou '}', desempilha e verifica o elemento no topo da pilha, que deve ser o caractere correspondente;
  - Ao final, a pilha deve estar vazia.

- Protótipo:

```
public boolean verifica(String exp)
```

Exemplo de aplicação do uso da Pilha (1)

```
public char fecho(char c)
{
          if(c == '}') return '{';

          if(c == ']') return '[';

          if(c == ')') return '(';

          return ' ';
}

… // continua
```

Exemplo de aplicação do uso da Pilha (2)

```
…
public boolean verifica(String exp) {
          char caractere;

          for(int i = 0; i < exp.length(); i++)
          {
                if(exp.charAt(i) == '{' ||
                   exp.charAt(i) == '[' ||
                   exp.charAt(i) == '(')
                {
                      caractere = exp.charAt(i);

                      push(caractere);
                }
                … // continua
```

Exemplo de aplicação do uso da Pilha (3)

```
            … // continua
            else if(exp.charAt(i) == '}' ||
                    exp.charAt(i) == ']' ||
                    exp.charAt(i) == ')')

            {
                if(pilhaEstaVazia())
                        return false;

                caractere = pop();

                if(caractere!=fecho(exp.charAt(i)) )
                        return false;
            }
        }

        if(!pilhaEstaVazia()) return false;
            return true;
    }
```

Aplicações do uso da Pilha

- Avaliação de expressões
  - Expressão: composta de operadores, operandos e delimitadores (parênteses).
  - Problema existente na avaliação: determinar em que ordem as operações devem ser realizadas, ou seja, determinar a prioridade dos operadores.
  - Notação convencional: infixa (operador entre operandos).
    - Ex.: (A+B)
  - Outras notações: pós-fixa, pré-fixa.

Aplicações do uso da Pilha

- Os prefixos "pre", "pos" e "in" referem-se à posição relativa do operador em relação aos dois operandos:
  - Na notação prefixa, o operador precede os dois operandos
  - Na notação posfixa, o operador é introduzido depois dos dois operandos e,
  - Na notação infixa, o operador aparece entre os dois operandos

Aplicações do uso da Pilha

Infixa	Pós-fixa	Pré-fixa
a	a	a
a+b	ab+	+ab
a+b*c	abc*+	+a*bc
(a+b)*c	ab+c*	*+abc

Aplicações do uso da Pilha

- Vantagens das notações *pós-fixa, pré-fixa*
  - não precisam de parênteses
  - não existe preocupação com prioridade de operandos

- Vantagens da notação *pós-fixa*
  - facilidade de avaliação: a expressão pode ser avaliada da esquerda para direita empilhando operandos, desempilhando operandos quando um operador é encontrado, calculando expressão dependendo do operador e também empilhando o resultado na pilha.

Aplicações do uso da Pilha

- Converter a expressão infixa A + B * C para posfixa:
  - Saber quais das operações (+ ou *) deve ser efetuada primeiro
  - Como a expressão não utiliza parênteses a multiplicação deve ser efetuada primeiro

A + (B * C)	parênteses para obter ênfase
A + (BC •)	converte a multiplicação
A (BC *) +	converte a adição
ABC * +	forma posfixa

Aplicações do uso da Pilha

- Regras a se lembrar ao converter de um tipo para outro:
  - As operações com a precedência mais alta são convertidas em primeiro lugar
  - Depois de uma parte da expressão ter sido convertida para posfixa, ela deve ser tratada como um único operando.
  - Observe o mesmo exemplo com a precedência de operadores invertida pela inserção de parênteses.

(A + B) * C	forma infixa
(AB +) * C	converte a adição
(AB +) C *	converte a multiplicação
AB + C*	forma posfixa

Aplicações do uso da Pilha

- Segue abaixo a ordem de precedência (da superior para a inferior) para esses operadores binários:
  - exponenciação
  - multiplicação/divisão
  - adição/subtração

- Quando operadores sem parênteses e da mesma ordem de precedência são avaliados, pressupõe-se a ordem da esquerda para a direita exceto,

Aplicações do uso da Pilha

- No caso da exponenciação, em que a ordem é supostamente da direita para a esquerda. Sendo assim:
  - *A + B + C* significa *(A + B) + C,*
  - Enquanto *A* ^ *B* ^ *C* significa *A* ^ *(B* ^ C)
- Outros exemplos de expressão infixa e posfixa

Notação Infixa	Notação Pósfixa
A - B * C	A B C * -
A * (B - C)	A B C - *
(A - B) / (C + D)	A B - C D + /
(A - B) / (C + D)* E	A B - C D + / E *
A^B * C - D + E / F / (G - H)	AB ^C * D - EF / GH - / +
((A + B) * C - (D - E)) ^ (F - G)	AB + C * DE - - FG - ^

Algoritmo para conversão de infixa para posfixa

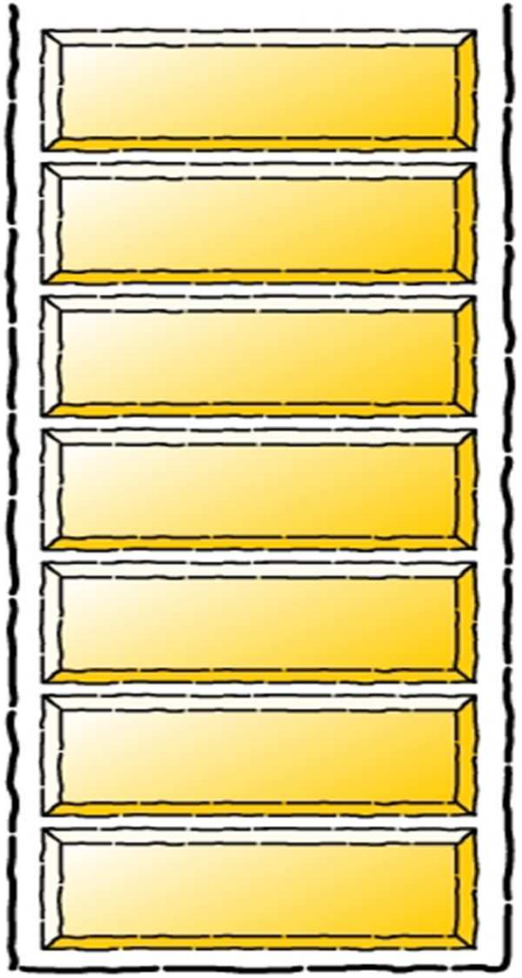
- Para chegar a esse algoritmo, observamos que a ordem dos operandos não é alterada quando a expressão é convertida: eles são copiados para a saída logo que encontrados
- Por outro lado os operadores devem mudar de ordem, já que na posfixa eles aparecem logo depois dos seus operandos
- Numa expressão posfixa, as operações são efetuadas na ordem em que aparecem. Logo, o que determina a posição de um operador é a prioridade que ele tem na forma infixa
- Aqueles operadores de maior precedência aparecem primeiro na expressão de saída e seu escopo será definido por parênteses

Algoritmo para conversão de infixa para posfixa

- Realize uma varredura na expressão infixa
  - Ao encontrar um operando, copie na expressão de saída
  - Ao encontrar o operador
    - Enquanto a pilha não estiver vazia e houver no seu topo um operador com prioridade maior ou igual ao encontrado, desempilhe o operador e copie-o na saída
    - Empilhe o operador encontrado
  - Ao encontrar um parêntese de abertura, empilhe-o
  - Ao encontrar uma parêntese de fechamento, remova os símbolos da pilha e copie-os na saída até que seja desempilhado o parêntese de abertura correspondente
- Ao final da varredura, esvazie a pilha, copiando os símbolos desempilhados para a saída

Conversão da forma infixa para pós-fixa

Pilha

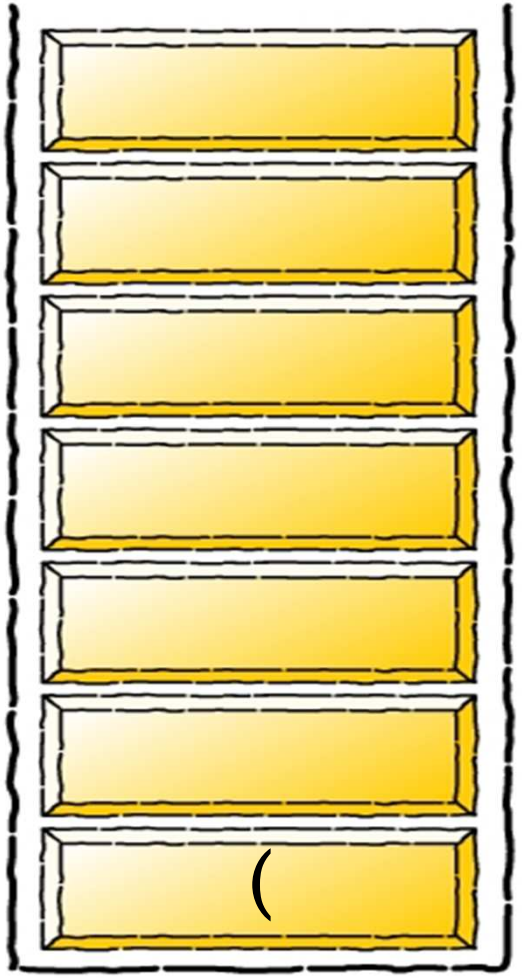


infixa

(a + b - c) * d - (e + f)

pós-fixa

Conversão da forma infixa para pós-fixa



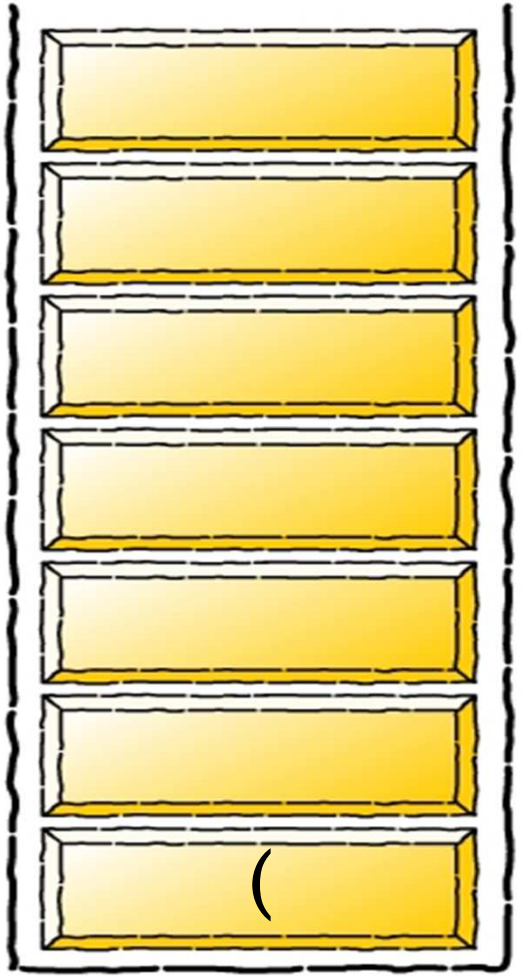
Pilha

infixa

a + b - c) * d - (e + f)

pós-fixa

Conversão da forma infixa para pós-fixa



Pilha

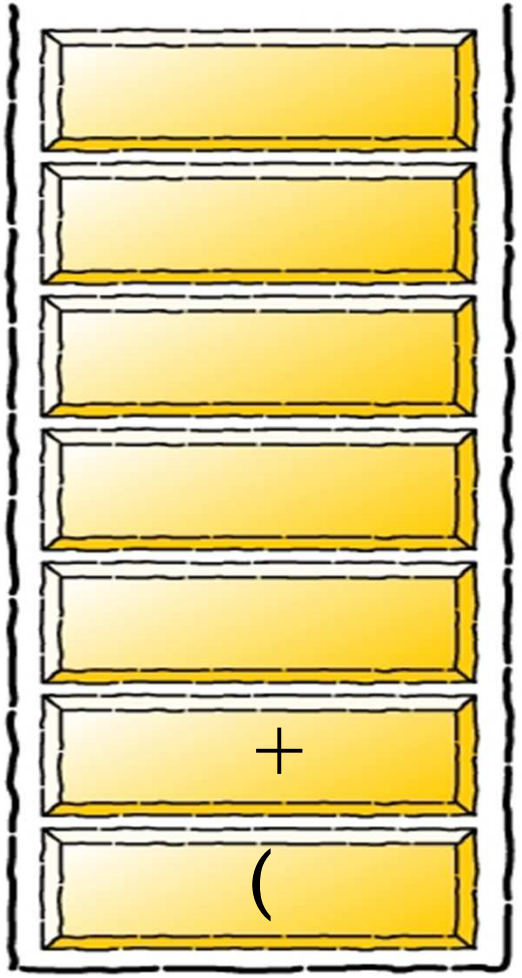
infixa

+ b - c) * d - (e + f)

pós-fixa

a

Conversão da forma infixa para pós fixa



Pilha

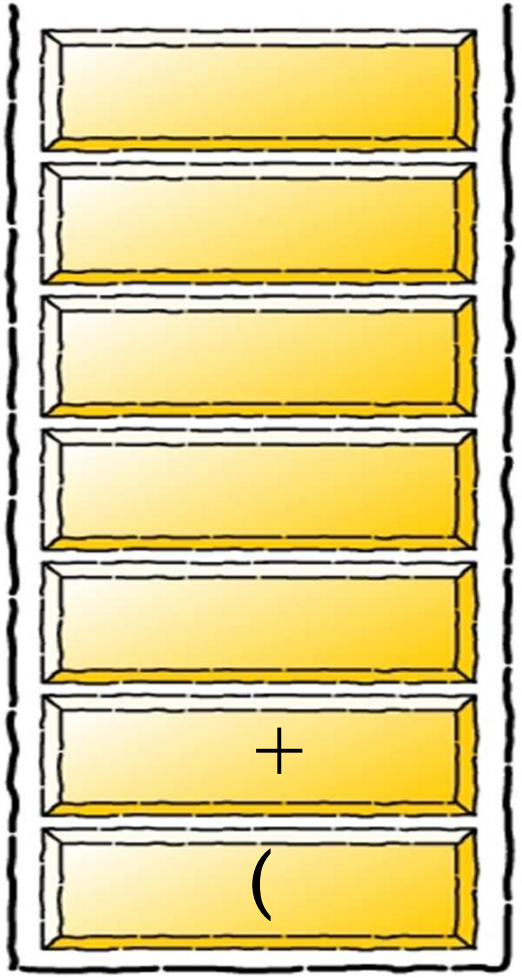
infixa

b - c) * d - (e + f)

pós-fixa

a

Conversão da forma infixa para pós-fixa



Pilha

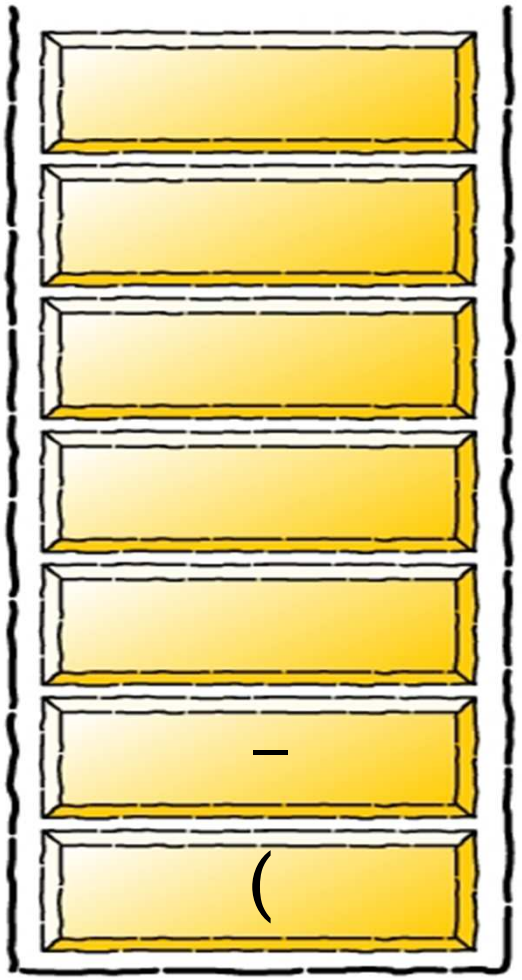
infixa

```
- c ) * d - ( e + f )
```

pós-fixa

```
a b
```

Conversão da forma infixa para pós-fixa



Pilha

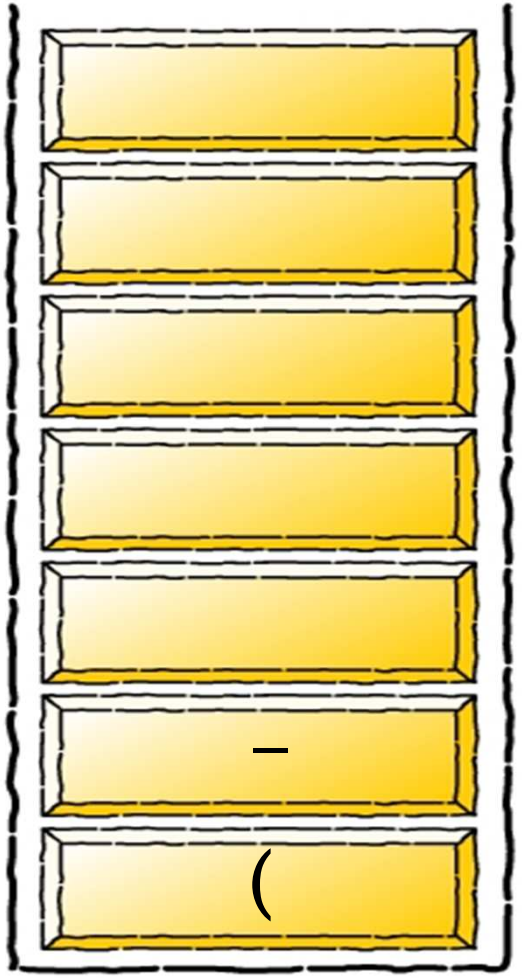
infixa

c) * d - (e + f)

pós-fixa

a b +

Conversão da forma infixa para pós-fixa



Pilha

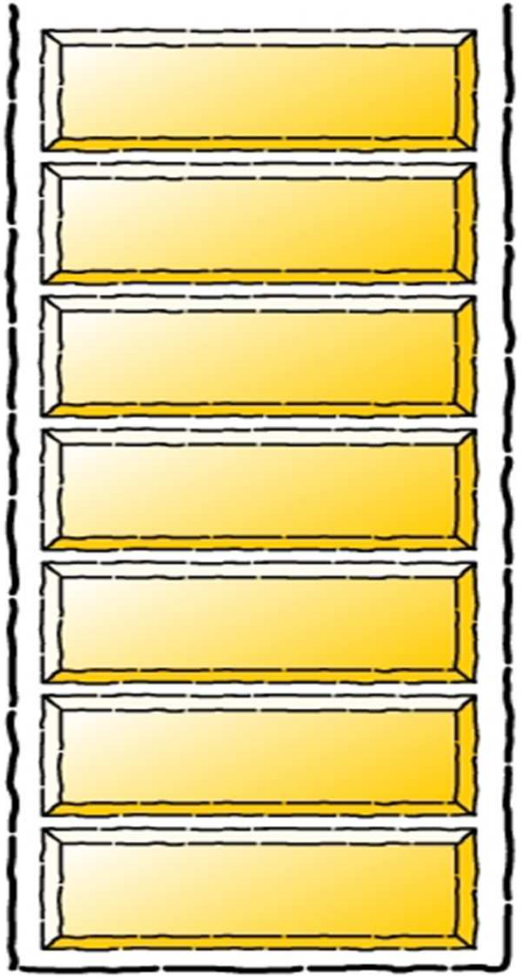
infixa

) * d - (e + f)

pós-fixa

a b + c

Conversão da forma infixa para pós-fixa



Pilha

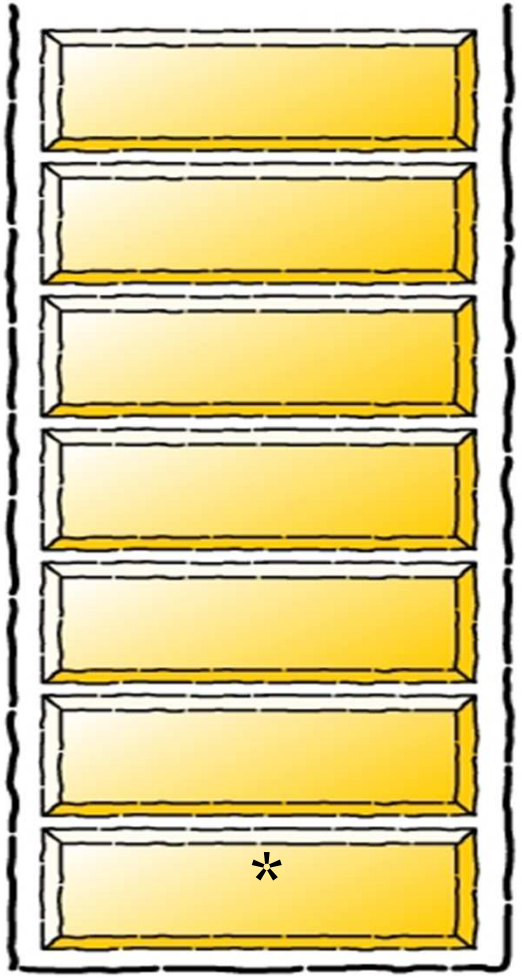
infixa

* d - (e + f)

pós-fixa

a b + c -

Conversão da forma infixa para pós-fixa



Pilha

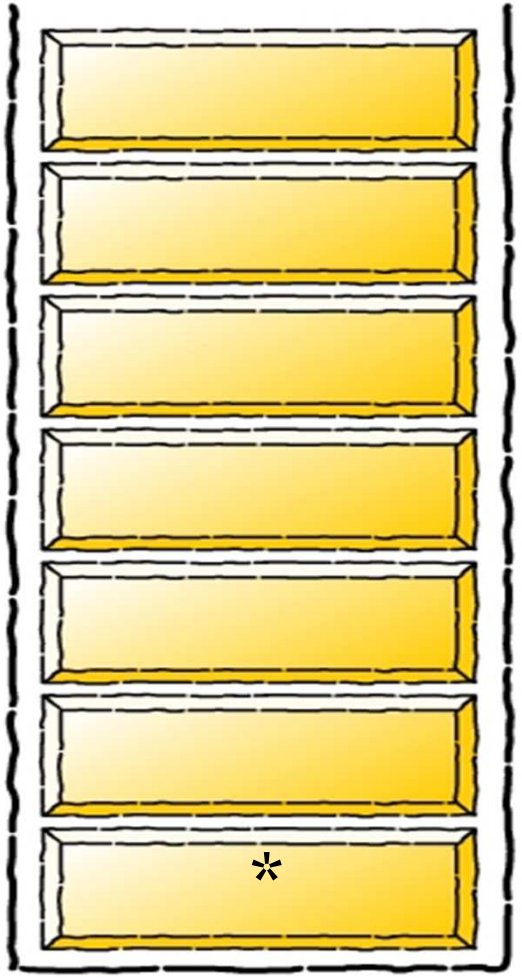
infixa

d - (e + f)

pós-fixa

a b + c -

Conversão da forma infixa para pós-fixa



Pilha

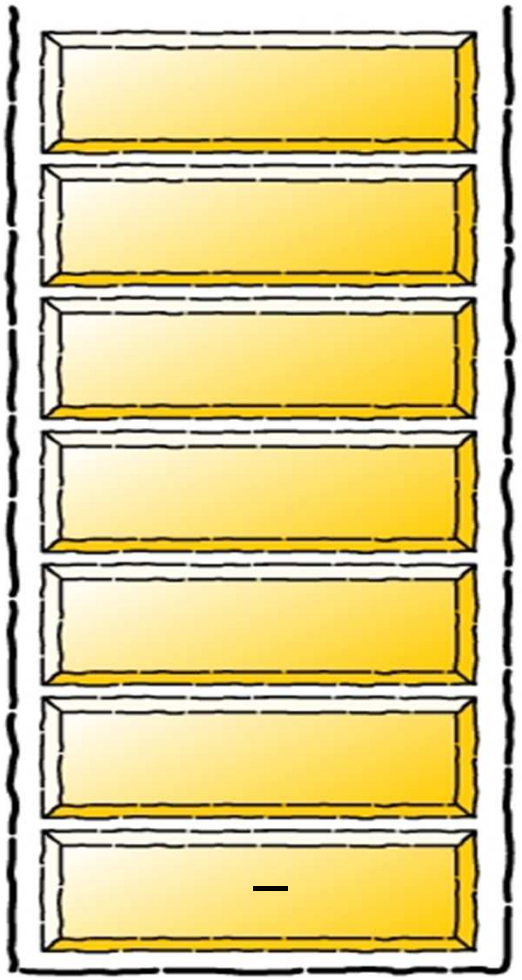
infixa

- (e + f)

pós-fixa

a b + c - d

Conversão da forma infixa para pós-fixa



Pilha

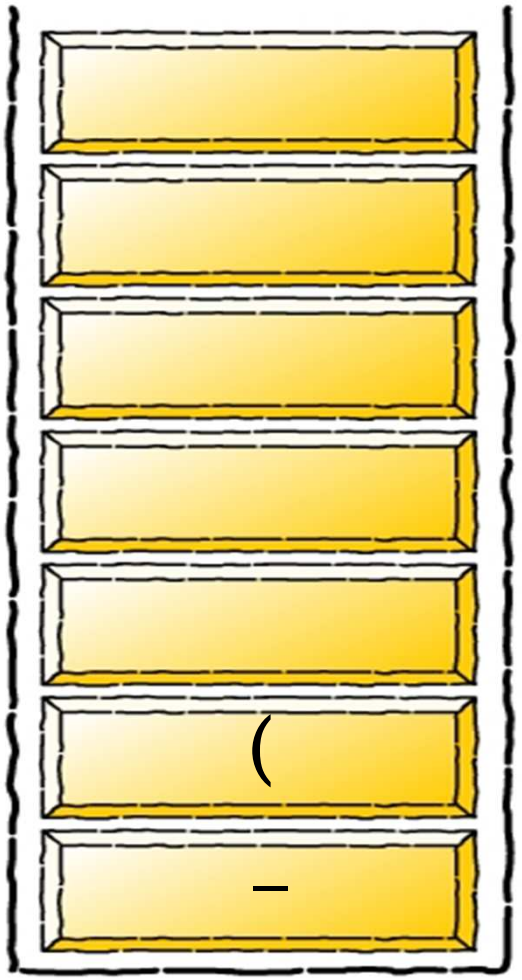
infixa

(e + f)

pós-fixa

a b + c - d *

Conversão da forma infixa para pós-fixa



Pilha

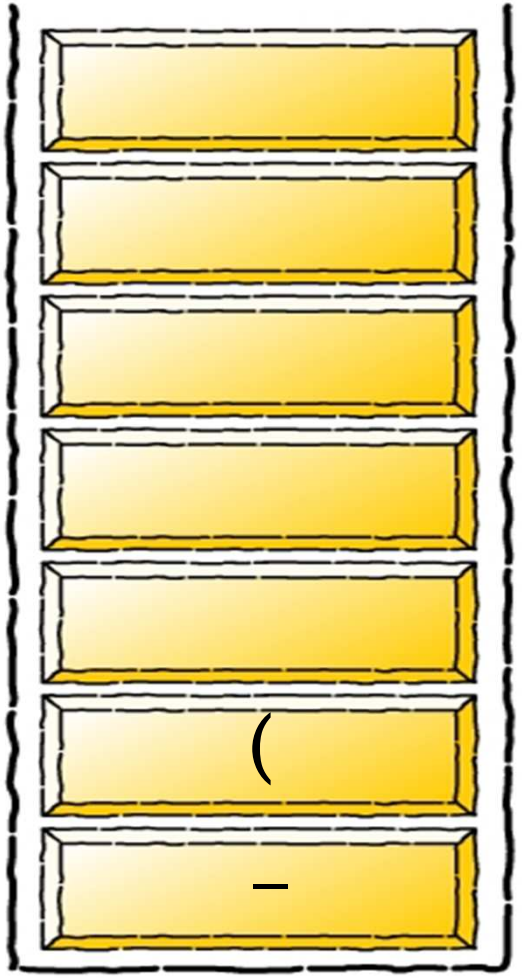
infixa

e + f)

pós-fixa

a b + c - d *

Conversão da forma infixa para pós-fixa



Pilha

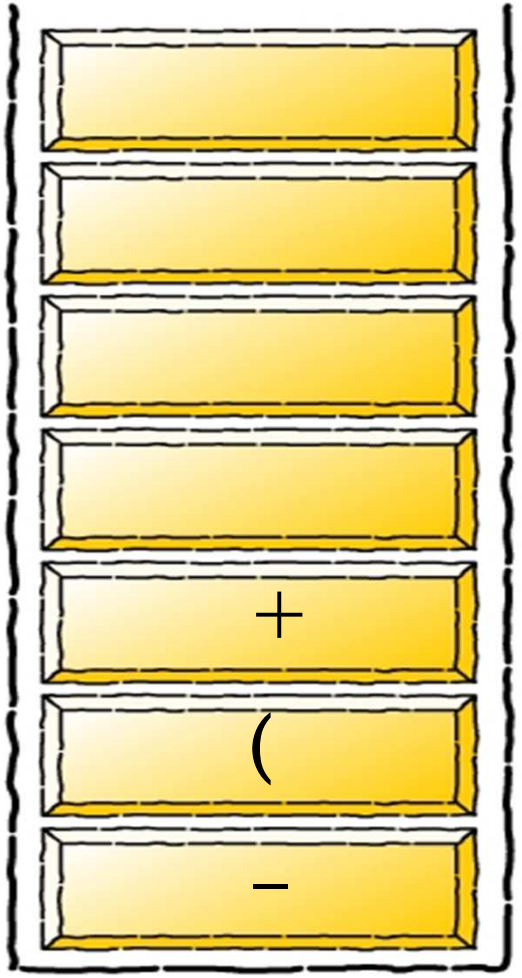
infixa

+ f)

pós-fixa

a b + c - d * e

Conversão da forma infixa para pós-fixa



Pilha

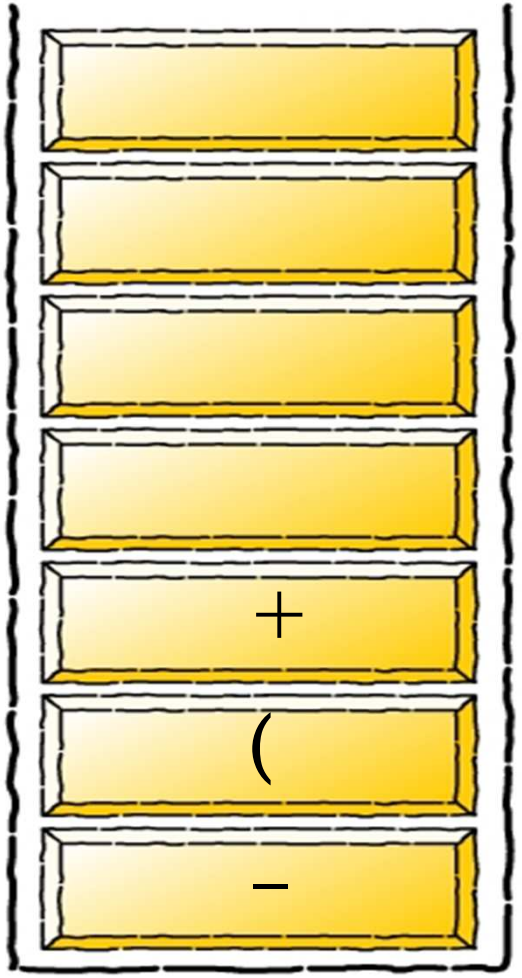
infixa

f)

pós-fixa

a b + c - d * e

Conversão da forma infixa para pós-fixa



Pilha

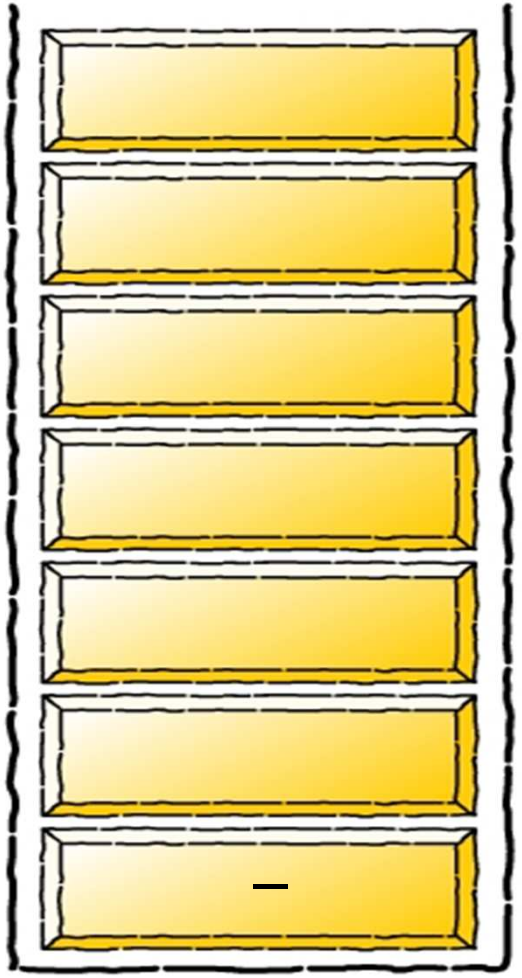
infixa

)

pós-fixa

a b + c - d * e f

Conversão da forma infixa para pós-fixa



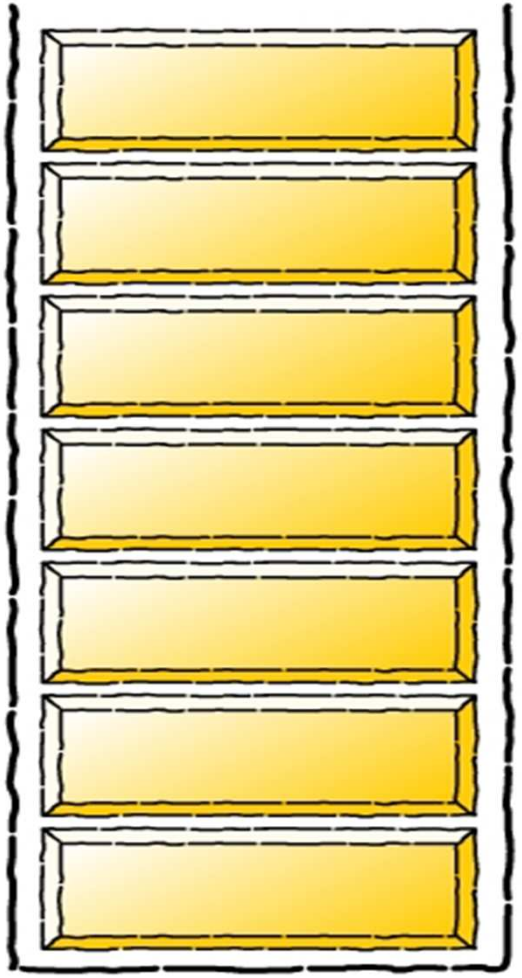
Pilha

infixa

pós-fixa

a b + c – d * e f +

Conversão da forma infixa para pós-fixa



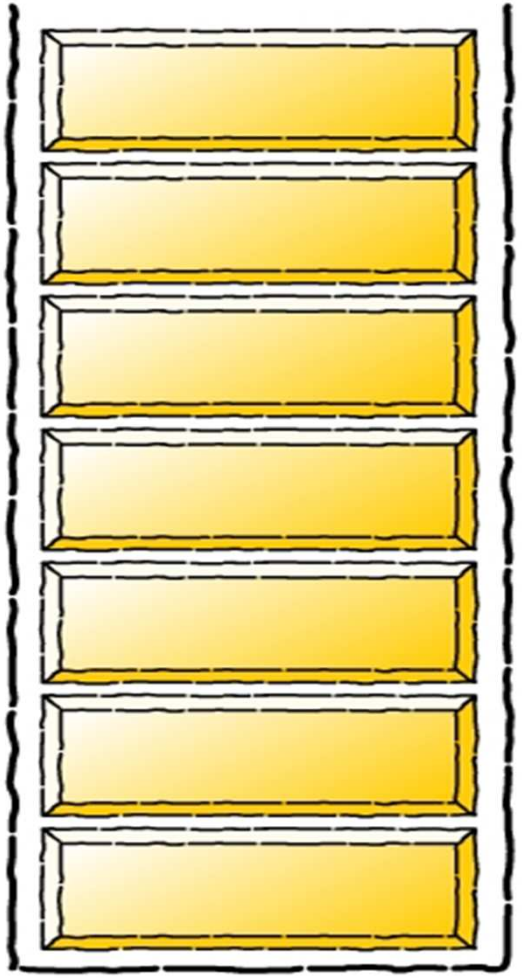
infixa

pós-fixa

a b + c - d * e f + -

Pilha

Conversão da forma infixa para pós-fixa



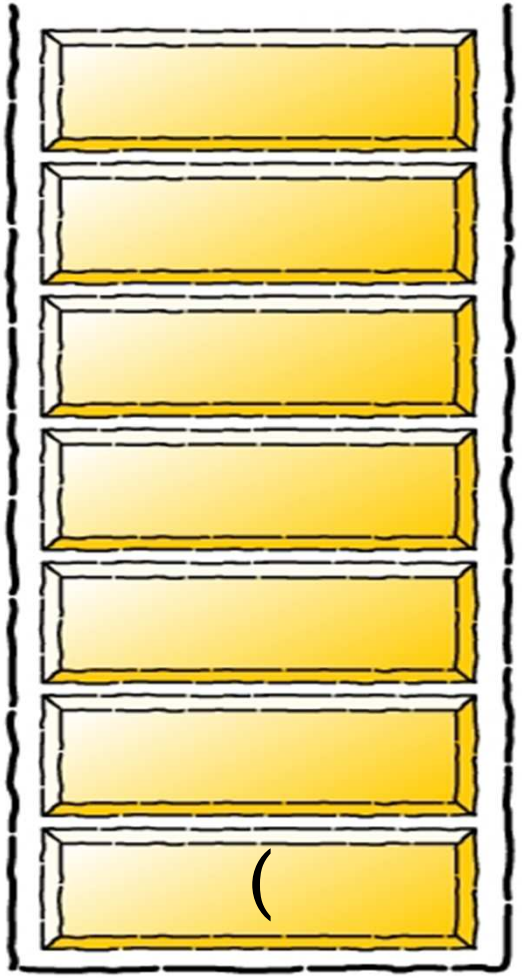
infixa

(a + b * c) * d - (e + f)

pós-fixa

Pilha

Conversão da forma infixa para pós-fixa



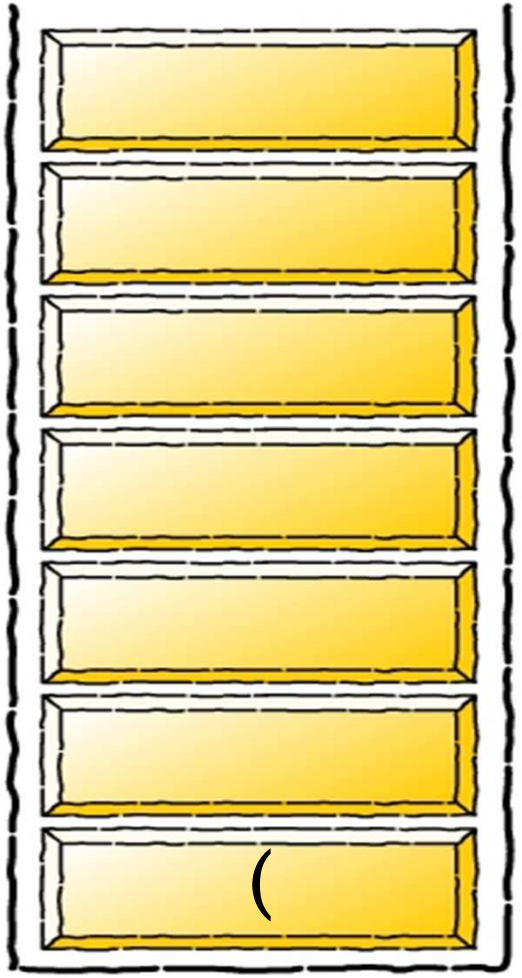
Pilha

infixa

a + b * c) * d - (e + f)

pós-fixa

Conversão da forma infixa para pós-fixa



Pilha

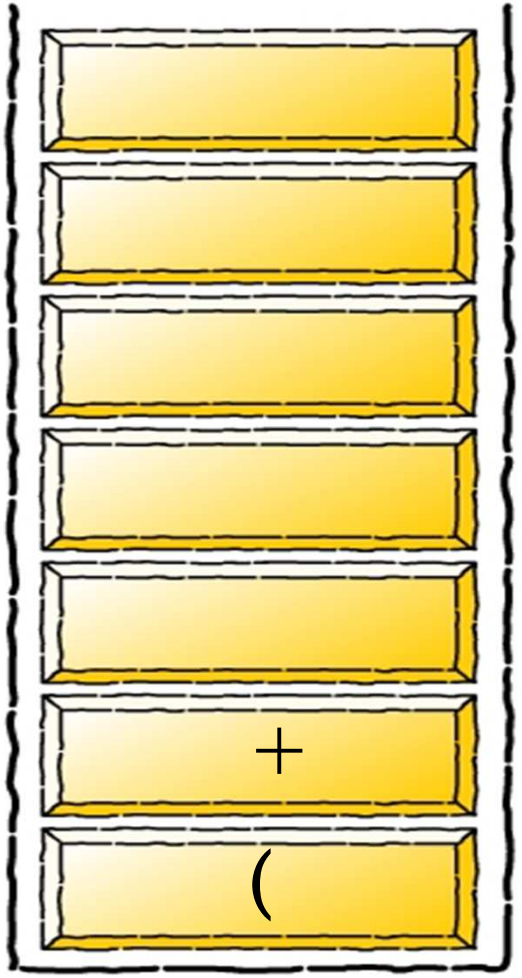
infixa

+ b * c) * d - (e + f)

pós-fixa

a

Conversão da forma infixa para pós-fixa



Pilha

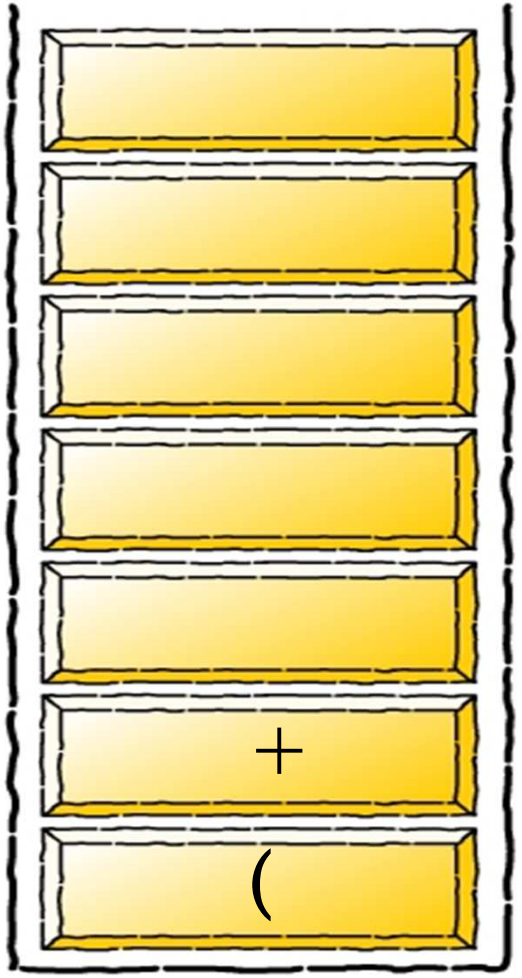
infixa

b * c) * d - (e + f)

pós-fixa

a

Conversão da forma infixa para pós-fixa



Pilha

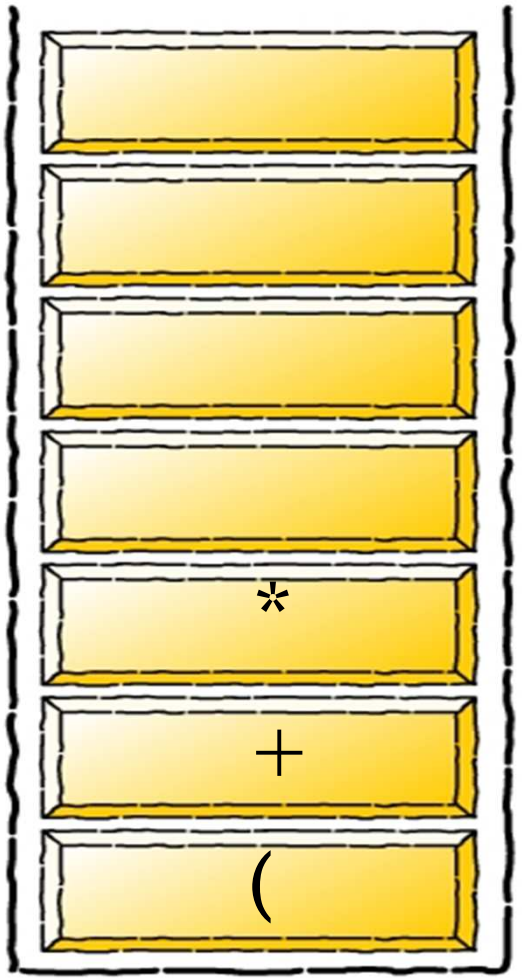
infixa

* c) * d - (e + f)

pós-fixa

a b

Conversão da forma infixa para pós-fixa



Pilha

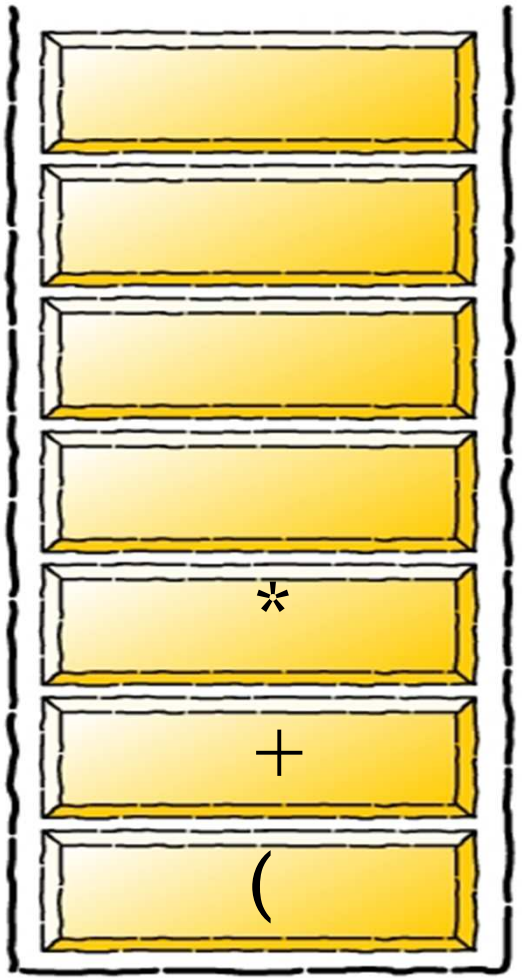
infixa

c) * d - (e + f)

pós-fixa

a b

Conversão da forma infixa para pós-fixa



Pilha

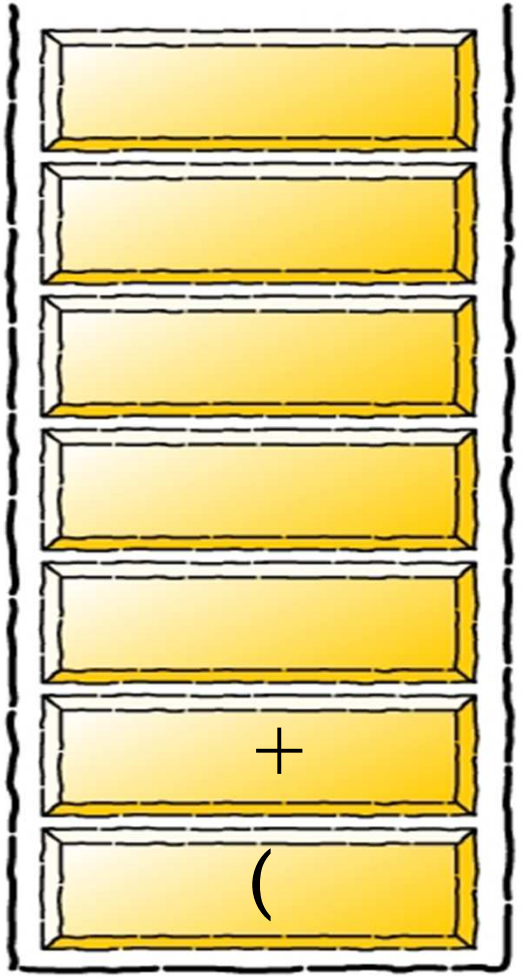
infixa

) * d - (e + f)

pós-fixa

a b c

Conversão da forma infixa para pós-fixa



Pilha

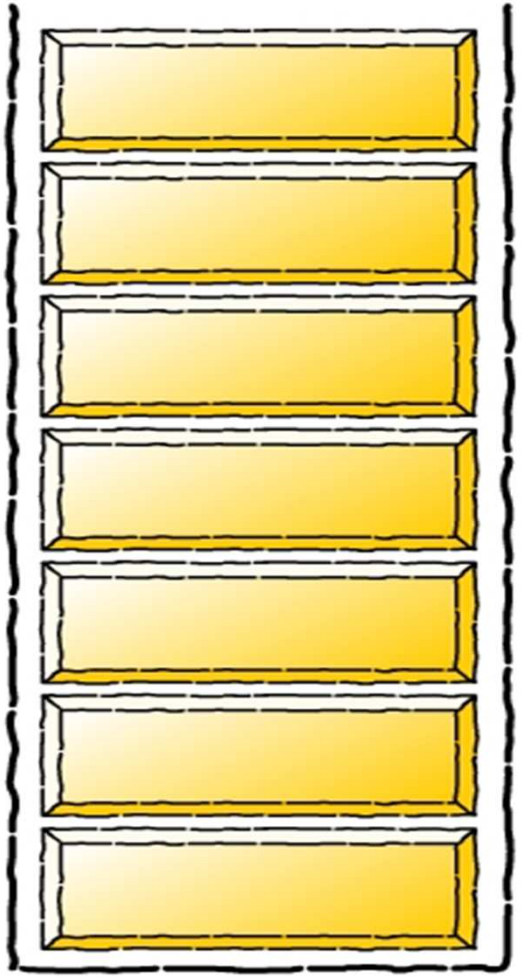
infixa

) * d - (e + f)

pós-fixa

a b c *

Conversão da forma infixa para pós-fixa



Pilha

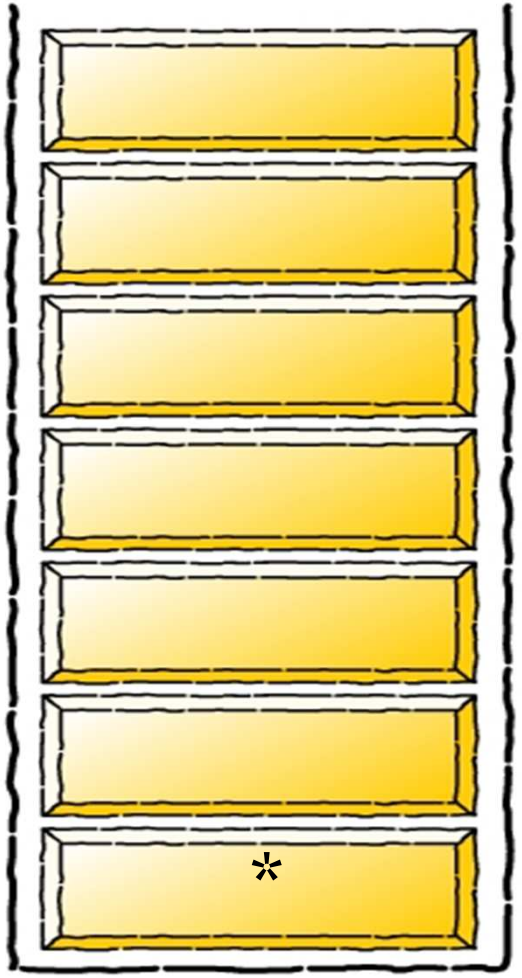
infixa

* d - (e + f)

pós-fixa

a b c * +

Conversão da forma infixa para pós-fixa



Pilha

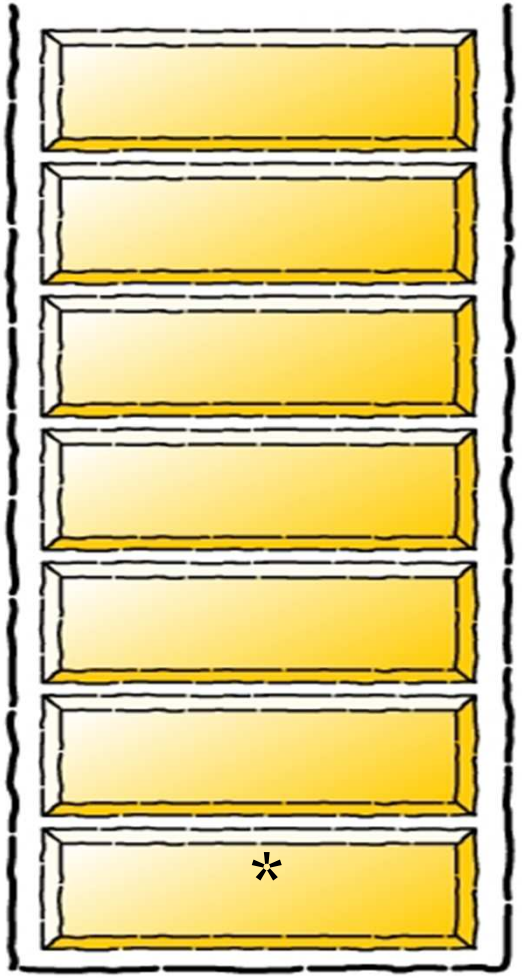
infixa

d - (e + f)

pós-fixa

a b c * +

Conversão da forma infixa para pós-fixa



Pilha

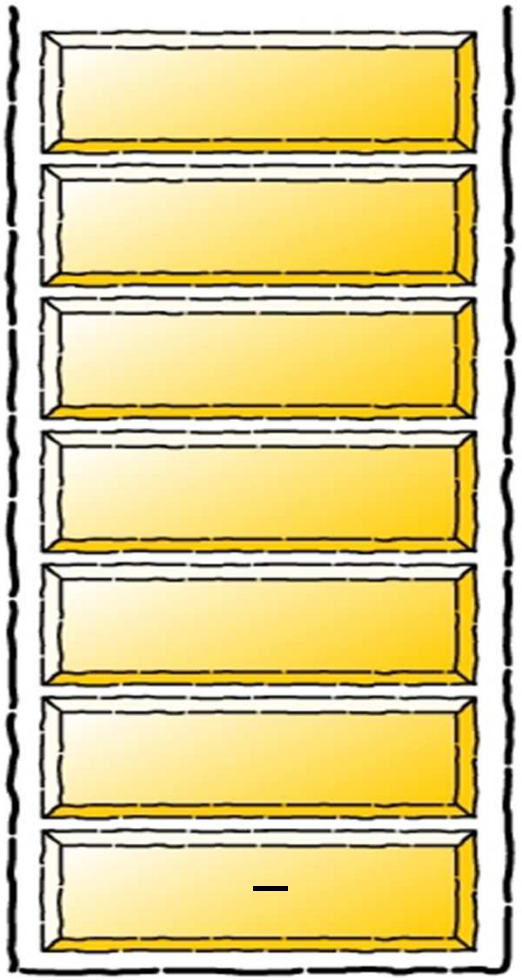
infixa

- (e + f)

pós-fixa

a b c * + d

Conversão da forma infixa para pós-fixa



Pilha

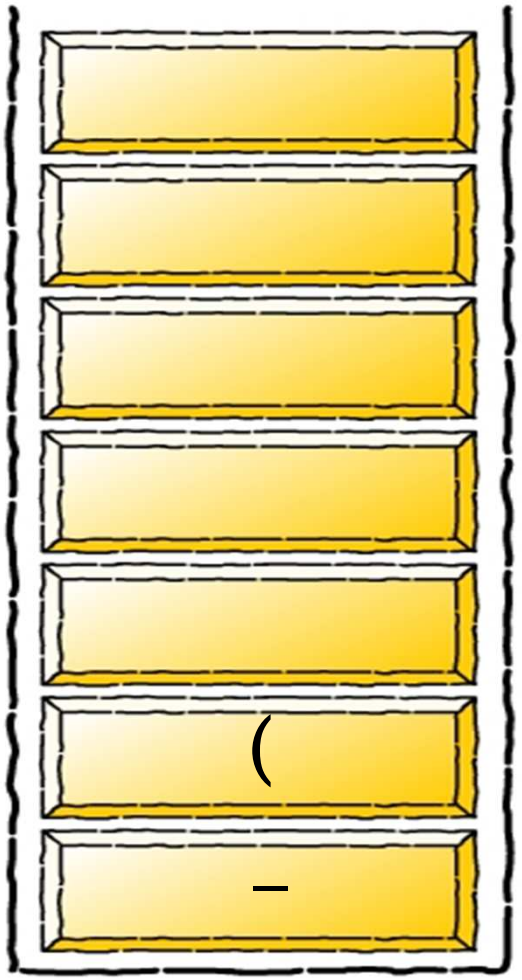
infixa

(e + f)

pós-fixa

a b c * + d *

Conversão da forma infixa para pós-fixa



Pilha

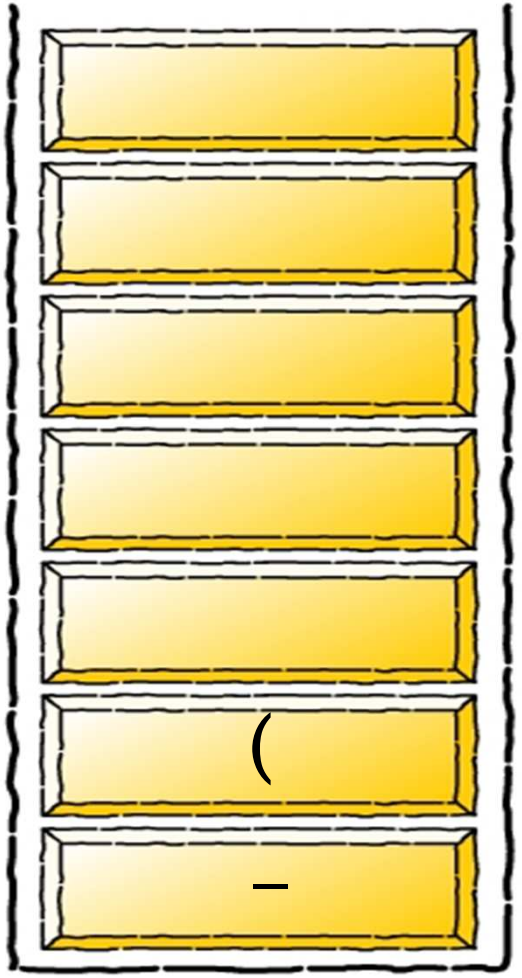
infixa

e + f)

pós-fixa

a b c * + d *

Conversão da forma infixa para pós-fixa



Pilha

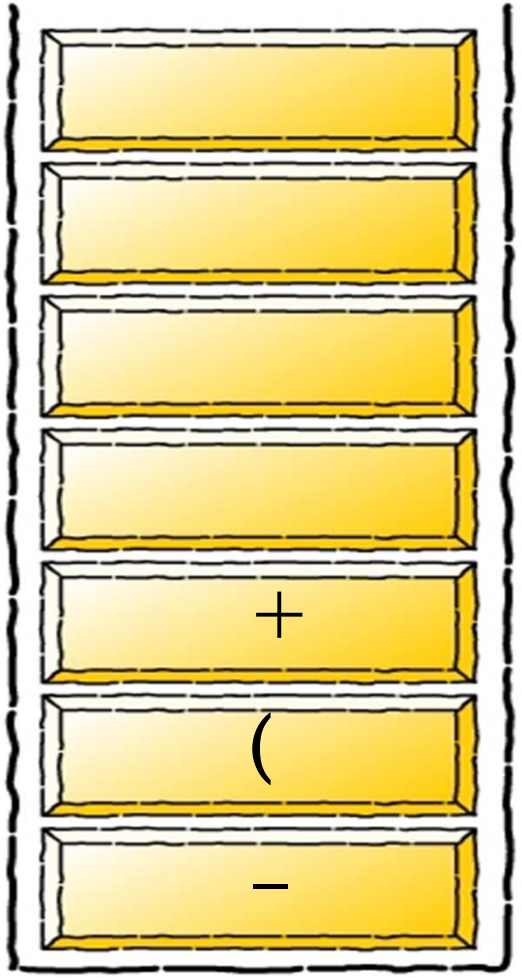
infixa

+ f)

pós-fixa

a b c * + d * e

Conversão da forma infixa para pós-fixa



Pilha

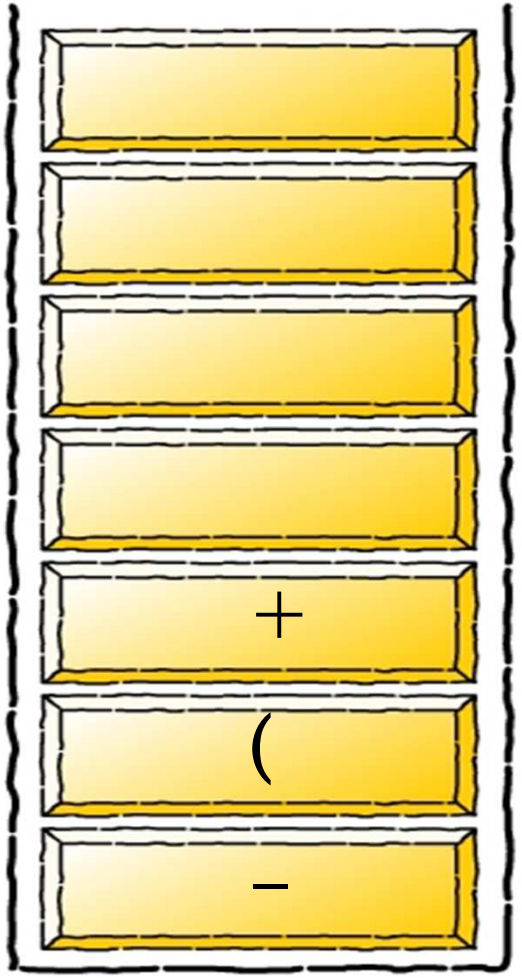
infixa

f)

pós-fixa

a b c * + d * e

Conversão da forma infixa para pós-fixa



Pilha

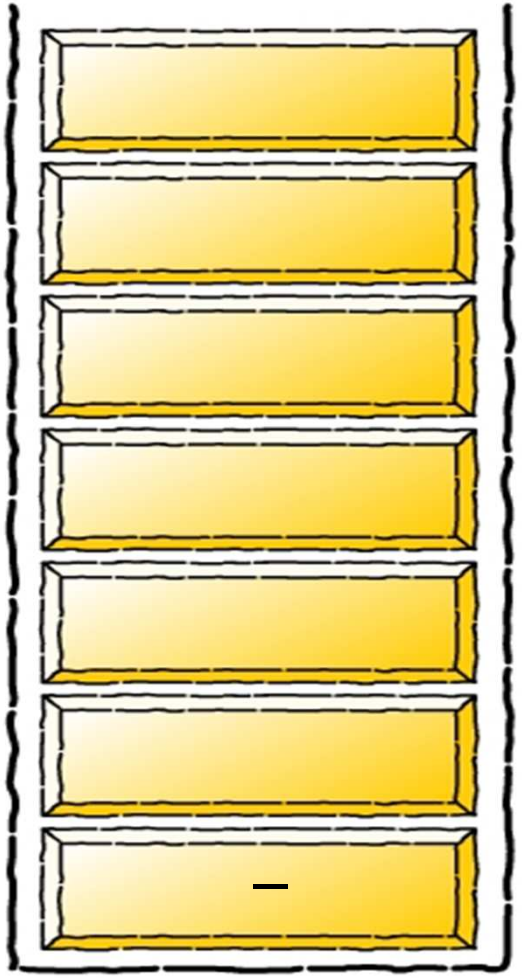
infixa

)

pós-fixa

a b c * + d * e f

Conversão da forma infixa para pós-fixa



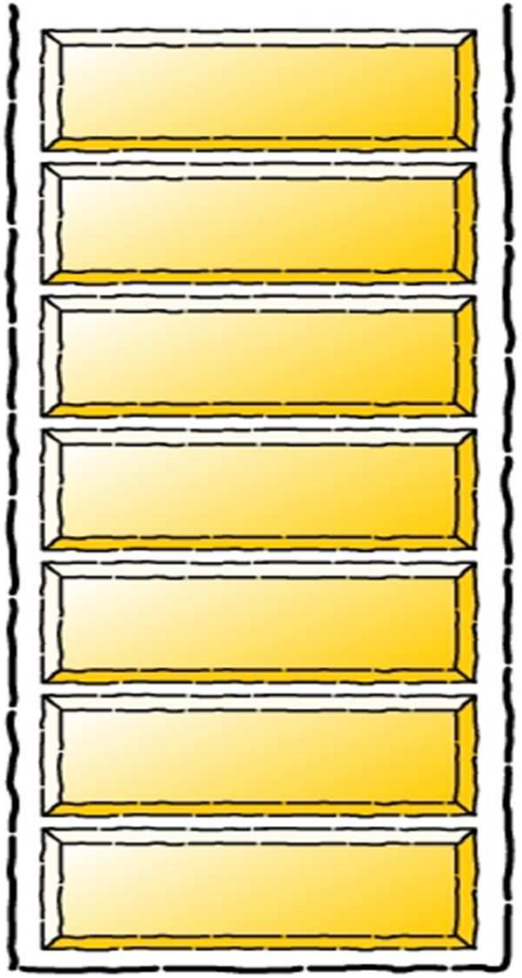
Pilha

infixa

pós-fixa

a b c * + d * e f +

Conversão da forma infixa para pós-fixa



infixa

pós-fixa

a b c * + d * e f + -

Pilha

Exercício

- Utilizando o algoritmo anteriormente apresentado implemente um programa que insira dados em uma pilha A e em seguida remova-os elementos da pilha A e insira-os na pilha B com sua ordem invertida.