

# Tipos abstratos de dados

## Tipo de Dado

- Quando o conceito de "tipo de dado" é dissociado dos recursos do hardware do computador, um número ilimitado de tipos de dados pode ser considerado.
- Um tipo de dado é um conceito abstrato, definido por um conjunto de propriedades lógicas.
- Assim que um tipo de dado abstrato é definido e as operações válidas envolvendo esse tipo são especificadas, podemos implementar esse tipo de dado (ou uma aproximação).
- Uma implementação pode ser uma implementação de hardware, na qual o circuito para efetuar as operações necessárias é elaborado e construído como parte de um computador; ou pode ser uma implementação de software, na qual um programa consistindo em instruções de hardware já existentes é criado para interpretar strings de bits na forma desejada e efetuar as operações necessárias.

## TIPOS DE DADOS ABSTRATOS

- Uma ferramenta útil para especificar as propriedades lógicas de um tipo de dado é o ***tipo de dado abstrato, ou TDA***.
- Fundamentalmente, um tipo de dado significa um conjunto de valores e uma sequência de operações sobre estes valores.
- Este conjunto e estas operações formam uma construção matemática que pode ser implementada usando determinada estrutura de dados do hardware ou do software.
- A expressão "tipo de dado abstrato" refere-se ao conceito matemático básico que define o tipo de dado.

## TIPOS DE DADOS ABSTRATOS

- Ao definir um tipo de dado abstrato como um conceito matemático, não nos preocupamos com a eficiência de tempo e espaço. Estas são questões de implementação.
- Na realidade, a definição de um TDA não se relaciona com nenhum detalhe da implementação.
- É possível até que não se consiga implementar determinado TDA num determinado hardware ou usando determinado sistema de software.
- Por exemplo, já constatamos que o TDA *inteiro* não é universalmente implementado. Apesar disso, especificando-se as propriedades matemáticas e lógicas de um tipo de dado ou estrutura, o TDA será uma diretriz útil para implementadores e uma ferramenta de apoio para os programadores que queiram usar o tipo de dado corretamente.

# Pilhas


implementação estática

Prof. Andreia Machion

## Definição

- Pilha é uma estrutura para armazenar um conjunto de elementos, que funciona da seguinte forma:
  - Novos elementos entram no conjunto, exclusivamente, no topo da pilha;
  - O único elemento que posso retirar da pilha em um dado momento, é o elemento do topo.
- Do Inglês: *Stack*, *LIFO*
  - Uma Pilha (em Inglês: *Stack*) é uma estrutura que obedece o critério L.I.F.O.: *Last In, First Out*. Ou seja, o último elemento que entrou no conjunto será o primeiro a sair.

## Quer ver?

Pilha de Potes	Pilha de Livros	Pilha de Pratos	Pilha de Cartas
			
Imagens: Yew Tree Gallery (potes), iStockPhoto (livros), Darren Maurer (pratos), e Barbosa, Miyoshi, e Gomes (cartas)			

## Idéias de operação

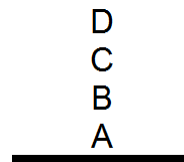
- Uma Pilha é um conjunto ordenado de elementos, ou seja, a ordem dos elementos no conjunto é importante. Se eu tenho três elementos em uma pilha, A, B e C, e se eles entraram na pilha nessa ordem, o elemento que estará no topo da pilha será o elemento C. E se eu quiser retirar um elemento nesse momento, o único elemento que poderei retirar da pilha será exatamente o elemento C

C  
B  
A

---

## Idéias de operação

- Considerando ainda o exemplo da Figura 3.2, se quisermos inserir um elemento D na pilha nesse momento, este elemento passaria a ser o elemento do topo da pilha, conforme mostra a Figura ao lado



## Para Que serve uma Pilha? O Exemplo da Chamada de Subprogramas

- Ao elaborar um programa de computador você já se deparou com um erro de execução chamado *stack overflow*? Sabe o que significa *stack overflow*? Sabe como ocorre? É um erro bastante comum. Se ainda não aconteceu com você, provavelmente ainda vai acontecer.
- Um computador está executando um trecho de programa A, e durante a execução de A encontra o comando *Call B*. Ou seja, uma chamada a um subprograma B. O computador precisa parar de executar A, executar B, e retornar ao programa A no ponto onde parou. Como o computador faz para lembrar a posição exata para onde deve retornar? Essa questão pode se complicar se tivermos várias chamadas sucessivas.

## Na prática...

- Considere o exemplo da tabela abaixo. Temos um programa principal, A, e 3 subprogramas: B, C e D.
- Ao iniciarmos a execução de A, na linha 1 temos o comando *Print A*, que imprime a letra A. Depois temos o comando *Call C*, ou seja, uma chamada ao subprograma C.
- Nesse ponto temos que interromper a execução de A e iniciar a execução do subprograma C. Ao finalizar a execução do subprograma C (comando *Return*, subprograma C linha 3), precisamos retomar a execução de A na linha 3, porque as linhas 1 e 2 de A já foram executadas.
- Vamos simular!

Programa A (principal) 1 print A 2 call C 3 call B 4 call D 5 return	Subprograma B 1 call C 2 print B 3 call D 4 call C 5 return	Subprograma C 1 print C 2 call D 3 return	Subprograma D 1 print D 2 return
--	--	--	--

## Operações - implementação

- *empilha(o)*: insere o objeto o no topo da pilha.
  - Entrada: objeto. Saída: nenhuma.
- *desempilha()*: retira o objeto no topo da pilha e o retorna;
  - Entrada: nenhuma. Saída: objeto.
- *tamanho()*: retorna o número de objetos na pilha.
  - Entrada: nenhuma. Saída: inteiro.
- *vazia()*: Retorna um booleano indicando se a pilha está vazia.
  - Entrada: nenhuma. Saída: booleano.
- *cheia()*: Retorna um booleano indicando se a pilha está cheia.
  - Entrada: nenhuma. Saída: booleano.