Algoritmos, Pilhas

Prof. Ricardo Reis
Universidade Federal do Ceará
Sistemas de Informação
Estruturas de Dados

28 de Maio de 2012

1 Definição

Uma *pilha* (ou *stack* em inglês) é uma estrutura de dados linear elementar que permite armazenar e recuperar objetos sobre as seguintes restrições ,

- Só é possível inserir (*empilhar*) ou remover (*desempilhar*) um objeto de uma extremidade pré-estabelecida da estrutura denominada *topo da pilha* (Figura-1).
- Só é possível ler dados do objeto que estiver no topo da pilha. Diz-se que os demais objetos estão *abaixo do topo*. Estes não podem ser acessados diretamente.
- Uma pilha que não comporta mais objetos (*pilha cheia*) deve reportar tentativas de empilhamentos sem sucesso (*stack overflow*) ao passo que uma pilha *sem* objetos (*pilha vazia*) deve reportar tentativas de desempilhamento sem sucesso (*stack underflow*).

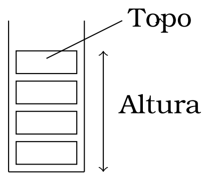

Figura 1: Esquema de Pilha

O número de objetos empilhados em uma pilha define a *altura da pilha*. O máximo de objetos que uma pilha suporta é denominado *capacidade da pilha*. Uma pilha vazia tem altura zero e uma pilha cheia possui altura igual a sua capacidade.

A ordem de empilhamento em uma pilha é sempre inversa a ordem de desempilhamento. Logo o primeiro objeto empilhado será o último a ser desempilhamento. Por essa razão as pilhas são denominadas estruturas FILO (do inglês, *First In Last Out*, ou, *primeiro a entrar último a sair*).

2 Operadores de Pilhas

Os operadores básicos de uma pilha, como estrutura de dados, são,

- **Create**: Operador responsável por construir uma nova pilha.
- **Push**: Operador que empilha um novo objeto no topo de uma pilha consequentemente aumentando sua altura em uma unidade quando a estrutura não estiver cheia.
- **Pop**: Operador que desempilha o objeto no topo da pilha consequentemente diminuindo sua altura em uma unidade quando a estrutura não estiver vazia.
- **Top**: Operador que retorna o valor do objeto no topo de uma pilha.
- **Empty**: Operador que testa se a pilha está vazia.
- **Full**: Operador que testa se a [ilha está cheia.
- **Destroy**: Operador que elimina uma dada pilha.

Note que são utilizados nomes em inglês para os operadores de pilhas (por questões de uso convencional na literatura sobre o assunto).

3 Pilhas Sequenciais

Uma pilha sequencial é aquela construída utilizando-se um vetor como estrutura base. De uma forma geral o acesso aleatório próprio de um vetor é camuflado pela implementação da pilha sendo assim as propriedades de pilhas meramente simuladas.

Tabela 1: Descritor de uma pilha Sequencial

Atributo	**Descrição**
M	vetor base
n	capacidade da pilha
h	Altura da pilha

Operador	**Argumentos**	**Descrição**
CREATE	n	Cria uma pilha sequencial de capacidade n
PUSH	$\mathcal{P}$, x	Empilha x na pilha $\mathcal{P}$
POP	$\mathcal{P}$	Desempilha topo da pilha $\mathcal{P}$
TOP	$\mathcal{P}$	Retorna valor no topo da pilha $\mathcal{P}$
EMPTY	$\mathcal{P}$	Verifica se a pilha $\mathcal{P}$ está vazia
FULL	$\mathcal{P}$	Verifica se a pilha $\mathcal{P}$ está cheia
DESTROY	$\mathcal{P}$	Elimina a pilha $\mathcal{P}$

A Tabela-1 ilustra o descritor de uma pilha sequencial. A implementação dos respectivos operadores é apresentada nos parágrafos seguintes.

O Algoritmo-1 implementa a construção de uma pilha sequencial. De forma similar a uma lista sequencial, é alocado um vetor dinâmico

M de comprimento n (capacidade da pilha). A altura é referenciada por h e como a pilha está inicialmente vazia seu valor inicial é zero. Uma nova pilha $\mathcal{P}$ é retornada como saída.

Algoritmo 1 Construtor de Pilha Sequencial

Função CREATE(n)
    $\mathcal{P}.M \leftarrow$ **alocar**(n)
    $\mathcal{P}.n \leftarrow n$
    $\mathcal{P}.h \leftarrow 0$
    Retorne $\mathcal{P}$

O Algoritmo-2 implementa o empilhador de pilha sequencial. A função deste operador é empilhar o objeto x na pilha de entrada $\mathcal{P}$. Inicialmente testa-se de a pilha ainda contém espaço (altura deve ser menor que a capacidade ou $\mathcal{P}.h < \mathcal{P}.n$). Em seguida aumenta-se a altura h em uma unidade e dispõe-se x no novo topo.

Algoritmo 2 Empilhador de Pilha Sequencial

Função PUSH(**ref** $\mathcal{P}$, x)
    Se $\mathcal{P}.h < \mathcal{P}.n$ **então**
        $\mathcal{P}.h \leftarrow \mathcal{P}.h + 1$
        $\mathcal{P}.M[\mathcal{P}.h] \leftarrow x$

O Algoritmo-3 implementa o desempilhamento em pilha sequencial. Este operador testa se a pilha não está vazia ($h > 0$) e em caso de êxito (ou seja, há um topo a eliminar) diminui a altura da pilha em uma unidade. Note que de fato objeto algum foi eliminado mas como h é reduzido então no próximo empilhamento o antigo topo será sobrescrito.

Algoritmo 3 Desempilhador de Pilha Sequencial

Função POP(**ref** $\mathcal{P}$)
    Se $\mathcal{P}.h > 0$ **então**
        $\mathcal{P}.h \leftarrow \mathcal{P}.h - 1$

No Algoritmo-4 é implementado o operador de leitura de topo de um pilha sequencial. Ele simplesmente retorna o valor $M[h]$ referente ao último objeto empilhado. Note que não há verificação de altura haja vista este operador não modificar a pilha.

No Algoritmo-5 é implementado o operador de verificação de pilha sequencial vazia. Ele simplesmente checa se a altura h da pilha vale zero.

No Algoritmo-5 é implementado o operador de verificação de pilha sequencial cheia. Ele checa se a altura h se igualou a capacidade n.

A destrução de uma pilha sequencial é implementada pelo operador do Algoritmo-7. Note que ele se restringe a desalocar o vetor base M e anular ambas capacidade (n) e altura (h) da pilha.

Por fim note que a pilha $\mathcal{P}$ é repassada aos operadores como referência. Isto de fato só é necessário quando o operador muda a pilha,

Algoritmo 4 Topo de Pilha Sequencial

```
Função TOP(ref 𝒫)
    Retorne 𝒫.M[𝒫.h]
```

Algoritmo 5 Verificador de Pilha Sequencial Vazia

```
Função EMPTY(ref 𝒫)
    Retorne 𝒫.h = 0
```

o que é o caso de PUSH e POP, mas por questões de homogeneidade de implementação foi mantido o **ref** em todos os operadores.

4 Pilhas Encadeadas

Uma pilha encadeada é aquela que utiliza uma lista encadeada como base. A maior vantagem neste caso é o fato de a pilha ter capacidade ilimitada. Uma pilha encadeada mantém internamente, como numa lista encadeada, uma referência para o primeiro nódo e a quantidade de nódos conforme indica o descritor da Tabela-2. Note que usamos a notação $topo$ para denotar a referência ao primeiro nódo. De fato, como se perceberá adiante, os empilhamentos e desempilhamentos ocorrem acerca desta referência porque considera-se que o primeiro nódo seja o topo da pilha (o efeito real dessa estratégia são algoritmos de complexidade na ordem de $O(1)$).

Ainda sobre o descritor da Tabela-2 note que existem duas diferenças entre os operadores e aqueles equivalentes em listas sequenciais. A primeira é no construtor que não possui argumento haja vista pilhas encadeadas não terem valor de capacidade máxima. A segunda é a ausência de um operador FULL de checagem de pilha cheia que, pela mesma razão já mencionada no construtor, não faria sentido existir.

Tabela 2: Descritor de uma pilha encadeada

Atributo	**Descrição**
$topo$	Referência para o nódo no topo da pilha
h	Altura da pilha

Operador	**Argumentos**	**Descrição**
CREATE	-	Cria uma pilha encadeada
PUSH	$\mathcal{P}$, x	Empilha x na pilha $\mathcal{P}$
POP	$\mathcal{P}$	Desempilha topo da pilha $\mathcal{P}$
TOP	$\mathcal{P}$	Retorna valor no topo da pilha $\mathcal{P}$
EMPTY	$\mathcal{P}$	Verifica se a pilha $\mathcal{P}$ está vazia
DESTROY	$\mathcal{P}$	Elimina a pilha $\mathcal{P}$

O operador de construção de pilha encadeada é implementado no Algoritmo-8. Nele os campos $nodo$ e h de uma pilha $\mathcal{P}$ são respectivamente configurados para λ e 0 definindo uma pilha vazia.

Algoritmo 6 Verificador de Pilha Sequencial Cheia

```
Função FULL(ref 𝒫)
    Retorne 𝒫.h = 𝒫.n
```

Algoritmo 7 Destrutor de Pilha Sequencial

```
Função DESTROY(ref 𝒫)
    Desalocar(𝒫.M)
    𝒫.h ← 𝒫.n = 0
```

No Algoritmo-9 é implementado o empilhador de pilha encadeada. Neste operador é criado um novo nódo $\mathcal{N}$ contendo em seu campo $chave$ o valor de empilhamento (repassado ao operador através do parâmetro x). Este novo nódo substitui o nódo topo da pilha $\mathcal{P}$ de trabalho (primeiro argumento do operador) se tornando o novo nódo da estrutura. Esta substituição ocorre em $O(1)$ e corresponde a duas reconexões: a primeira que faz $\mathcal{N}$ apontar para o topo ($\mathcal{N}.prox \leftarrow \mathcal{P}.topo$) e a segunda que muda o topo para nd ($\mathcal{P}.topo \leftarrow \mathcal{N}$). Note que ainda é necessário aumentar de uma unidade a altura da pilha ($\mathcal{P}.h \leftarrow \mathcal{P}.h + 1$).

O operador de desempilhamento em pilha encadeada é implementado no Algoritmo-10. A pilha $\mathcal{P}$ passada como argumento, quando não vazia ($\mathcal{P}.h > 0$), sofre uma extração do nódo topo diminuindo sua altura da unidade ($\mathcal{P}.h \leftarrow \mathcal{P}.h - 1$). A extração possui três etapas: na primeira a referência $\mathcal{N}$ é utilizada para manter o topo da pilha ($\mathcal{N} \leftarrow \mathcal{P}.topo$); em seguida o topo é deslocado para o nódo seguinte ($\mathcal{P}.topo \leftarrow \mathcal{P}.nodo.prox$) e que corresponde, em termos de pilhas, ao elemento logo abaixo do topo; e finalmente o nódo referenciado por $\mathcal{N}$ é desalocado. Note que em caso de um único nódo a extração faz $\mathcal{P}.topo$ receber λ e h zero.

O operador de verificação de topo de pilha encadeada é implementado no Algoritmo-11. Este operador simplesmente verifica e retorna o valor da chave do nódo topo ($\mathcal{P}.topo.chave$). Não há verificação de pilha vazia neste operador, necessidade essa suprida pelo operador EMPTY.

A verificação de vazio em pilha encadeada é implementada no Algoritmo-12. O operador simplesmente checa se a altura da pilha de entrada ($\mathcal{P}.h$) é nula. Outra alternativa, mostrada como comentário, é testar se a referência ao topo da pilha aponta para nulo ($\mathcal{P}.topo = \lambda$).

O operador de destrução de pilha encadeada é implementado pelo Algoritmo-13. De forma similar ao que ocorre em destrução de listas encadeadas, este operador executa um laço que continuamente efetua desempilhamentos e encerra quando a pilha de entrada, $\mathcal{P}$, se esvazia.

5 Aplicações de Pilhas

5.1 Conversão de Base

A conversão de base é o processo de transformação de um número escrito normalmente em base decimal para uma base não decimal normalmente 2 (binária), 8 (octal) ou 16 (hexadecimal).

Algoritmo 8 Construtor de Pilha Encadeada

Função CREATE(n)
  $\mathcal{P}.topo \leftarrow \lambda$
  $\mathcal{P}.h \leftarrow 0$
  Retorne $\mathcal{P}$

Algoritmo 9 Empilhador de Pilha Encadeada

Função PUSH(**ref** $\mathcal{P}$, x)
  $\mathcal{N}.chave \leftarrow x$
  $\mathcal{N}.prox = \mathcal{P}.topo$
  $\mathcal{P}.topo = \mathcal{N}$
  $\mathcal{P}.h \leftarrow \mathcal{P}.h + 1$

Para converter um número n na base decimal numa base arbitrária b deve-se proceder da seguinte forma. Divide-se n por b e coleta-se o resto. Em seguida faz-se o mesmo com o quociente desta divisão e repete-se o processo até que se obtenha quociente zero. A ordem inversa dos restos coletados representa o novo número na base b. Os valores de restos coletados, que também estão na base decimal, são representados por valores na faixa $0..b-1$ e logo podem possuir mais de um dígito o que prejudicaria a nova representação na base b. Uma saída para simplificar a representação de n na nova base é atribuir letras aos restos cujos valores são maiores que nove (ou seja, que possuem mais de um dígito). O exemplo mais comum desta notação ocorre na base 16 onde a faixa de valores de restos é $0..15$ e a associação com letras é dada por,

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Utilizando=se uma pilha pode-se coletar (empilhar) os restos das divisões de um processo de conversão de base e em seguida efetuar sucessivos desempilhamentos até esvaziamento da pilha. Os valores desempilhados e impressos nesta ordem representam a ordem inversa de geração de restos e consequentemente a ordem natural de dígitos da saída procurada.

A função CONV, Algoritmo-14, utiliza uma pilha $\mathcal{P}$ para converter um número n na base b onde $b \in \{2, 8, 16\}$. A ideia desta função é guardar numa pilha $\mathcal{P}$ todos os restos de sucessivas divisões de n por b (linha-4) e em seguida esvaziá-la (linha-7) com pré-impressões do topo. Mote que é utilizada uma string de mapeamento $mapa$ onde é buscada o caractere equivalente ao resto t coletado da pilha. Por exemplo, se $b = 16$ então t admitirá valores na faixa $0..15$ que adicionados de 1 (como na linha-9) representarão índices válidos à string $mapa$ e cujos caracteres correspondentes são exatamente os desejados.

Algoritmo 10 Desempilhador de Pilha Encadeada

Função POP(**ref** $\mathcal{P}$)
  Se $\mathcal{P}.h > 0$ **então**
    $\mathcal{N} \leftarrow \mathcal{P}.topo$
    $\mathcal{P}.topo = \mathcal{P}.topo.prox$
    $\mathcal{P}.h \leftarrow \mathcal{P}.h - 1$
    Desalocar($\mathcal{N}$)

Algoritmo 11 Topo de Pilha Encadeada

Função TOP(**ref** $\mathcal{P}$)
  Retorne $\mathcal{P}.topo.chave$

5.2 Avaliação de Expressões Algébricas

5.2.1 Representação Pós-fixa

Em expressões algébricas a utilização de operadores *binários* (ou seja, que operam dois operandos) é comumente *infixa*, ou seja, são dispostos entre os operadores. Por exemplo, na expressão $A + B$ o operador $+$ é binário pois opera A e B e é infixo por se dispor entre estas variáveis. A consequência direta desta notação é a necessidade do uso de parênteses quando se deseja quebrar a precedência entre os operadores [1]. Por exemplo, nas expressões $A + B \times C$ e $(A + B) \times C$ respectivamente ocorrem nesta ordem multiplicação/soma e soma/multiplicação. Pela possibilidade de uso de parênteses as expressões infixas são também às vezes chamadas *expressões parentizadas*.

Em geral computar expressões parentizadas é problemático porque os parênteses (ou a falta deles) geram ambiguidades. Para contornar isso utiliza-se a notação *pósfixa* onde os operadores são dispostos após os operadores. Por exemplo $A + B$ em notação pósfixa se torna $AB+$.

O processo de formação de uma expressão pósfixa ocorre por processamento de pares dispostos por precedência na expressão infixa. Por exemplo, a expressão,

$$(A + B \times (C - D))/E$$

em notação pósfixa deve se tornar,

$$ABCD - \times + E/$$

A lógica desta expressão é a seguinte. Os operandos devem manter-se na mesma ordem em que aparecem na expressão infixa e os operado-

[1]Os operadores aritméticos com maior ordem de precedência são a multiplicação e a divisão e de menor precedência a soma e a subtração. Concretamente isso equivale a dizer que numa expressão algébrica devem ser feitas primeiramente multiplicações e divisões e só então somas e subtrações. Por exemplo, a expressão,

$$3 + 5 \times 7$$

vale 38 pois deve-se primeiro multiplicar para então depois somar.

Algoritmo 12 Verificador de Pilha Encadeada Vazia

```
Função EMPTY(ref 𝒫)
    Retorne 𝒫.h = 0                      ▷ ou 𝒫.topo = λ
```

Algoritmo 13 Destrutor de Pilha Encadeada

```
Função DESTROY(ref 𝒫)
    Enquanto não EMPTY(𝒫) faça
        POP(𝒫)
```

res devem surgir logo após ao par de operandos que devem computar. Assim, como o primeiro operando é o $-$ e há o par CD imediatamente antes, então a primeira operação computada deverá ser $C - D$. Em seguida o par de $\times$ é formado por B e a parcela já computada, $C - D$, obtendo-se a computação parcial $B \times (C - D)$. O próximo operador, $+$, opera o par formado por A e a computação de $B \times (C - D)$ obtendo-se $A + B \times (C - D)$. Por fim a computação de / opera o par formado entre $A + B \times (C - D)$ e E gerando $(A + B \times (C - D))/E$ que é a expressão inicial infixa.

5.2.2 Transformação de Expressões Infixas em Pós-fixas

Apresentaremos nessa sessão um algoritmo que utiliza uma pilha para transformar uma expressão infixa (parentizada) em notação pósfixa. Os elementos utilizados neste algoritmo são,

- Uma expressão infixa de entrada E formada pelos operadores $+$, $-$, $\times$ e $/$, por parênteses (e) e por variáveis de um único caractere na faixa $A \ldots Z$.
- Uma pilha $\mathcal{P}$ que empilha apenas os operadores $+$, $-$, $\times$, $/$ e ((abre-parêntese).
- Uma string R de saída onde será armazenada a expressão pósfixa e que inicialmente está vazia.

O algoritmo de transformação de uma expressão infixa para pósfixa consiste em varrer E da esquerda para a direita e processar cada caractere ch da forma seguinte,

- Se ch for uma variável (ou seja, pertence a $A \ldots Z$) então deve ser concatenado a string R.
- Se ch for um operador aritmético ($+$, $-$, $\times$, $/$) deve ser empilhado em $\mathcal{P}$ sobre as seguintes restrições,
  - Se $\mathcal{P}$ estiver vazia ou a precedência do operador no topo de $\mathcal{P}$ for menor que o operador em ch então ch deverá ser diretamente empilhado em $\mathcal{P}$.

Algoritmo 14 Conversão de número decimal para outra base numérica

Função CONV(n, b)
    $mapa \leftarrow$ "0123456789ABCDEF"
    $\mathcal{P} \leftarrow$ CREATE() ▷ Usando pilha encadeada
    Enquanto $n > 0$ **faça**
        PUSH($\mathcal{P}$, $n \bmod b$) ▷ Coleta de restos
        $n \leftarrow \lfloor n/b \rfloor$ ▷ Sucessivas divisões
    Enquanto não EMPTY($\mathcal{P}$) **faça**
        $t \leftarrow$ TOP($\mathcal{P}$)
        Escreva $mapa[t+1]$ ▷ Mapeamento
        POP($\mathcal{P}$)
    DESTROY($\mathcal{P}$)

Tabela 3: Precedência de Operadores

Operadores	Valor de Precedência
(	1
$+ \; -$	2
$\times \; /$	3

  - Se o operador no topo de $\mathcal{P}$ tiver precedência maior que aquele em ch então deve-se sucessivamente desempilhar operadores de $\mathcal{P}$ até que a pilha se esvazie ou até que a prioridade do operador no topo dela seja menor que o operador em ch. Só então deve-se empilhar ch em $\mathcal{P}$. Os operadores desempilhados de $\mathcal{P}$ devem ser concatenados a R na ordem de desempilhamento.

- Se ch for um abre-parêntese deve ser empilhado diretamente em $\mathcal{P}$.

- Se ch for um fecha-parêntese deve-se efetuar sucessivos desempilhamentos em $\mathcal{P}$ em busca do abre-parêntese correspondente. Quando encontrado deve-se fazer um desempilhamento adicional para eliminar o abre-parêntese agora no topo. Os operadores desempilhados de $\mathcal{P}$ devem ser concatenados a R na ordem de desempilhamento.

Se no final da varredura a pilha $\mathcal{P}$ não estiver vazia então deve-se executar sucessivos desempilhamentos até esvaziá-la. Os operadores desempilhados de $\mathcal{P}$ devem ser concatenados a R na ordem de desempilhamento. O conteúdo em R após todas as concatenações representa a expressão pósfixa procurada.

Os valores de prioridade dos operadores empilhados em $\mathcal{P}$ devem seguir a Tabela-3. Note que o operador abre-parêntese, ao contrário da intuição matemática, possui a menor prioridade. Isso se deve ao fato de que eles não podem impedir os operadores aritméticos de entrarem na pilha $\mathcal{P}$. Além do mais eles devem estar necessariamente em $\mathcal{P}$ para que o processamento dos fecha-parênteses ocorram normalmente.

O Algoritmo-15 implementa a transformação de expressões infixas em pósfixas. Há duas funções. A primeira, PRIOR, retorna a prioridade de um operador passado via argumento ch (segue valores da Tabela-3). A segunda função, POSFIXAR, é o transformador propriamente dito e segue as quatro etapas descritas anteriores.

Algoritmo 15 Transformação de uma expressão Infixa em Pósfixa

```
Função PRIOR(ch)
    Se ch = ( então
        Retorne 1
    senão Se ch ∈ {+, −} então
        Retorne 2
    senão Se ch ∈ {×, /} então
        Retorne 3

Função POSFIXAR(E)
    n ← Comprimento(E)          ▷ comprimento da expressão de entrada
    P ← CREATE(n)               ▷ Pilha de Operadores
    R ← " "                     ▷ String de saída: inicia vazia
    Para i ← 1 . . . n faça     ▷ Laço de varredura
        ch ← E[i]
        Se ch ∈ {A . . . Z} então          ▷ Encontra variável
            R ← R + ch
        senão Se ch ∈ {+, −, ×, /} então   ▷ Encontra operador
            t ← PRIOR(ch)
            Enquanto não EMPTY(P) e PRIOR(TOP(P)) ≥ t faça
                R ← R + TOP(P)
                POP(P)
            PUSH(P, ch)
        senão Se ch = ( então              ▷ Encontra abre-parêntese
            PUSH(P, ch)
        senão Se ch = ) então              ▷ Encontra fecha-parêntese
            Enquanto TOP(P) ≠ '(' faça
                R ← R + TOP(P)
                POP(P)
            POP(P)
    Enquanto não EMPTY(P) faça            ▷ Esvazia pilha
        R ← R + TOP(P)
        POP(P)
    DESTROY(P)                            ▷ Destrói Pilha
    Retorne R                             ▷ Retorna expressão pósfixa
```

5.2.3 Avaliando uma Expressão Algébrica

Avaliar uma expressão algébrica significa atribuir valores numéricos a suas variáveis e em seguida simplificá-la a um único número. Por exemplo, avaliando,

$$A + B \times (C - D)$$

com $A = 4$, $B = 11$, $C = 2$ e $D = 5$ obtém-se,

$$4 + 11 \times (2 - 5) = -29$$

Para avaliar uma expressão algébrica E de entrada deve-se convertê-la em uma expressão pósfixa R e depois, utilizando uma pilha numérica $\mathcal{P}$ (que empilha números de ponto flutuante), aplicar o algoritmo descrito a seguir. Varre-se R da esquerda para a direita e para cada caractere ch visitado faz-se o seguinte,

- Se ch for uma variável ($ch \in \{A \dots Z\}$) então deve-se empilhar em $\mathcal{P}$ seu equivalente valor numérico.

- Se ch for um operador devem-se efetuar dois desempilhamentos em $\mathcal{P}$ recuperando-se o primeiro valor desempilhado através de uma variável auxiliar y e o segundo através de uma variável auxiliar x. Em seguida deve-se operar x e y com o operador em ch e por fim reempilhar em $\mathcal{P}$ o valor obtido. A ordem do par de desempilhamentos é inversa a da operação e por essa razão o primeiro desempilhamento é recuperado por y e o segundo por x tornando as operações $x + y$, $x - y$, $x \times y$ e $x \div y$ coerentes.

Quando esse processo se encerra a pilha $\mathcal{P}$ contém apenas um valor que corresponde ao valor da avaliação o qual pode ser obtido por um último desempilhamento. O Algoritmo-16 implementa a avaliação de uma expressão pósfixa passada como entrada. O argumento de entrada $\mathcal{M}$ representa um *mapa de variável* que mostraremos adiante como gerar. Se ch é uma variável então $\mathcal{M}$ mapeia seu valor por $\mathcal{M}[ch].valor$. As etapas do algoritmo de avaliação descrito são comentadas na implementação.

Definimos como *mapa de variáveis* a uma estrutura de dados que mantém pares atrelados formados por um rótulo de variável e seu respectivo valor. Para montarmos um mapa de variáveis consideraremos um vetor indexado pelos caracteres $A \dots Z$ e cujas células contém uma parte lógica $usada$, que informa se a variável está sendo ou não utilizada, e uma parte numérica $valor$ contendo o valor propriamente dito da variável.

A função MAPEAR, Algoritmo-17, implementa a geração de uma mapa de variáveis a partir de uma expressão algébrica de entrada E (que pode ser infixa ou pósfixa). As etapas do algoritmo são as seguintes,

- Criar um mapa $\mathcal{M}$ com a descrição anterior (vetor indexado pelos caracteres $A \dots Z$ cujas células contém campos $valor$ e $usada$)

- Carregar os campos $valor$ e $usada$ de todas as células respectivamente com zero e **`Falso`** (mapa vazio). Veja linha-2.

- Varrer a expressão (linha-6) de entrada E em busca de caracteres na faixa $A \dots Z$ (linha-8) e para cada variável encontrada requisitar seu valor ao usuário e armazená-la no campo $valor$ correspondente.

Algoritmo 16 Avaliação de uma expressão pósfixa

Função AVALIAR(R, $\mathcal{M}$)
  $n \leftarrow$ **Comprimento**(R)
  $\mathcal{P} \leftarrow$ CREATE(n) ▷ Pilha numérica
  Para $i \leftarrow 1 \dots n$ **faça** ▷ Laço de varredura da expressão
    $ch \leftarrow R[i]$ ▷ Caractere avaliado
    Se $ch \in \{A \dots Z\}$ **então** ▷ Empilha valor de variável
      PUSH($\mathcal{P}$, $\mathcal{M}[ch].valor$)
    senão ▷ Desempilha par, opera e reempilha
      $y \leftarrow$ TOP$(\mathcal{P})$
      POP($\mathcal{P}$)
      $x \leftarrow$ TOP$(\mathcal{P})$
      POP($\mathcal{P}$)
      Se $ch = +$ **então** PUSH($\mathcal{P}$, $x + y$)
      senão Se $ch = -$ **então** PUSH($\mathcal{P}$, $x - y$)
      senão Se $ch = \times$ **então** PUSH($\mathcal{P}$, $x * y$)
      senão Se $ch = /$ **então** PUSH($\mathcal{P}$, x/y)
  $x \leftarrow$ TOP$(\mathcal{P})$ ▷ Resgata valor da avaliação
  DESTROY($\mathcal{P}$) ▷ Destrói pilha
  Retorne x ▷ Retorna valor da avaliação

- Cada variável carregada deve ter o campo *usada* mudado para **Verdadeiro** impedindo que seja solicitada mais de uma vez (teste da linha-9).

Algoritmo 17 Gerador de mapa de variáveis

1: **Função** MAPEAR(E)
2:   **Para** $ch \leftarrow \{A \dots Z\}$ **faça**
3:     $\mathcal{M}[ch].valor \leftarrow 0$
4:     $\mathcal{M}[ch].usada \leftarrow$ **Falso**
5:   $n \leftarrow$ **Comprimento**(E)
6:   **Para** $k \leftarrow 1 \dots n$ **faça**
7:     $ch \leftarrow E[i]$
8:     **Se** $ch \in \{A \dots Z\}$ **então**
9:       **Se não** $\mathcal{M}[ch].usada$ **então**
10:         **Escreva** "Escreva valor de", ch, ": "
11:         **Leia** $\mathcal{M}[ch].valor$
12:         $\mathcal{M}[ch].usada \leftarrow$ **Verdadeiro**
13:   **Retorne** $\mathcal{M}$

5.3 Transformação de Algoritmos Recursivos em Iterativos

5.3.1 QuickSort Iterativo

Consideremos nessa sessão a transformação do algoritmo de ordenação rápida (quicksort) em forma iterativa utilizando uma pilha. A ideia de transformação é implementada pelo Algoritmo-18 e descrita a seguir.

1. Define-se uma pilha $\mathcal{P}$ de faixas (linha-4), ou seja, cada objeto f empilhado traz dois campos $f.p$ e $f.q$ que denotam respectivamente os índices inferior e superior de uma faixa de vetor.
2. Empilha-se em $\mathcal{P}$ a faixa $[1, n]$ referente a um vetor M de comprimento n que se deseja ordenar (linha-5).
3. Enquanto a pilha $\mathcal{P}$ não estiver vazia efetua-se as etapas (linha-6),
    (a) Desempilha-se a faixa no topo de $\mathcal{P}$ e aplica-lhe o particionamento (linha-7).
    (b) Reempilham-se as duas faixas providas pelo particionamento desde que sejam válidas, ou seja, o campo p seja menor que o campo q (Entre linha-9 e linha-16).

Algoritmo 18 QuickSort iterativo utilizando uma pilha

1: **Função** IQUICKSORT(**ref** L, n)
2: $\quad f.p \leftarrow 1$ $\triangleright$ Faixa de índices do vetor
3: $\quad f.q \leftarrow n$
4: $\quad \mathcal{P} \leftarrow \text{CREATE}(n)$
5: $\quad$ PUSH($\mathcal{P}$, f)
6: $\quad$ **Enquanto não** EMPTY($\mathcal{P}$) **faça**
7: $\qquad f \leftarrow \text{TOP}(\mathcal{P})$
8: $\qquad r \leftarrow \text{PART}(L, f.p, f.q)$
9: $\qquad$ **Se** $f.p < r - 1$ **então**
10: $\qquad\quad z.p \leftarrow f.p$
11: $\qquad\quad z.q \leftarrow r - 1$
12: $\qquad\quad$ PUSH($\mathcal{P}$, z)
13: $\qquad$ **Se** $r + 1 < f.q$ **então**
14: $\qquad\quad z.p \leftarrow r + 1$
15: $\qquad\quad z.q \leftarrow f.q$
16: $\qquad\quad$ PUSH($\mathcal{P}$, z)

É importante notar que o Algoritmo-18 mantém a essência da ordenação rápida, ou seja, progressivamente sub-faixas do vetor de entrada sofrem particionamentos que provocarão ao final a ordenação do vetor. Entretanto ao invés de uma *pilha de recursão* (que é o que se forma num processo recursivo), é utilizada uma pilha explícita, ou seja, visível ao implementador. Note que $\mathcal{P}$ cresce em altura enquanto as faixas oriundas do particionamento forem válidas começando entretanto a decair quando as faixas atingem tamanho igual a um ou zero deixando de entrar na pilha (testes da linha-9 e da linha-13).

6 Exercícios

1. Dadas duas pilhas contendo chaves ordenadas na ordem de empilhamento, escreva algoritmo que imprima essas chaves em ordem ascendente na saída padrão. O algoritmo deve ter complexidade $O(n)$ onde n é a soma das alturas das pilhas.
2. Utilize uma pilha para converter as expressões infixas a seguir em notação pósfixa,
   (a) $A/(B+C)$
   (b) $(A+B)\times(C-D)$
   (c) $(A+(B\times((A-B)/E)))$
   (d) $(A\times(B/(C\times(D+A-E))))$
   (e) $((A\ times(B-C))/(E/(D+E)))\times(A/(B\times(D-E)))$
3. Utilize uma pilha para ilustrar todas as etapas de avaliação das expressões numéricas seguintes,
   (a) $5\times(56-9)$
   (b) $(3-7)\times(22+13)$
   (c) $((12-16)\times 25-4)/(3-7)$
4. Reimplemente a conversão em notação pósfixa de uma expressão algébrica de forma a aceitar também a operação de potência (uso o símbolo $\wedge$). A precedência da potência é maior que da multiplicação e divisão.
5. Use três pilhas para simular as movimentações de discos entre pinos no problema clássico da Torre de Hanói.
6. Um *palíndromo* é uma string cuja ordem reversa de caracteres revela a própria string de base. Por exemplo, `ana` e `radar` são exemplos de palíndromos. Utilizando uma pilha construa uma função que verifique se uma string de entrada é ou não um palíndromo.
7. Expressões que utilizam parênteses podem ser acidentalmente escritas erroneamente. Os exemplos a seguir mostram sequencias de parênteses inválidas: $(()(())$, $((())$,$()())($. Utilizando pilhas construa uma função que teste se a parentização de uma dada expressão (que pode possuir outros elementos além de parênteses) está correta.
8. Seja $\mathcal{P}$ uma pilha e U_n o vetor formado pelos n primeiros naturais não nulos. U_n é varrido da esquerda para a direita e suas chaves são empilhadas em $\mathcal{P}$ (operação I) havendo ainda intercalações com operações de desempilhamento (operações R) cujos valores são impressos nessa ordem. Por exemplo, se $U_3=\{1,2,3\}$ e a sequência de operações executadas forem $IIRIRR$ então $\mathcal{P}$ se esvazia e os valores impressos são $2,3,1$ que é uma permutação das chaves de U_3. Determine sequências de operações que gerem permutações válidas nos casos de U_4 e U_5.
9. Utilizando pilhas construa uma versão iterativa para o algoritmo de ordenação por fusão (MergeSort).
10. Utilizando pilhas implemente versões iterativas para as funções a seguir,
   (a)
   ```
   Função FNC1(n)
       Se n > 3 então
           FNC1(n/3)
           Escreva n
   ```

FNC1($n/3$)

(b) **Função** FNC2(n)
Se $n > 3$ **então** FNC2($n/3$)
Escreva n
Se $n > 5$ **então** FNC2($n/5$)

(c) **Função** FNC3(**ref** M, p, q)
Se $p < q$ **então**
Retorne 0
senão Se $p = q$ **então**
Retorne $M[p]$
senão
$r \leftarrow \lfloor p + q \rfloor /2$
Retorne $M[r] +$ FNC3$(M, p, r - 1) +$ FNC3$(M, r + 1, q)$