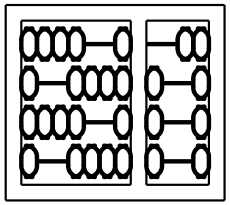


Universidade Estadual de Campinas - UNICAMP
Instituto de Computação - IC

Estruturas lineares

Estruturas lineares em geral

- Muitas aplicações envolvem representação e manipulação de sequências ordenadas de objetos.

- Operações típicas:
  1. selecionar e modificar o k-ésimo elemento,
  2. inserir um novo elemento entre as posições k e $k+1$,
  3. remover o k-ésimo elemento,
  4. concatenar duas sequências,
  5. copiar uma sequência,
  6. determinar o tamanho de uma sequência,
  7. buscar um elemento que satisfaz um critério de busca,
  8. ordenar uma sequência,
  9. aplicar um ou vários procedimentos a todos ou parte dos elementos da sequência,
  10. ...

Estruturas lineares em geral

• Muitas aplicações envolvem representação e manipulação de sequências ordenadas de objetos.

• Operações típicas:

1. selecionar e modificar o k-ésimo elemento,
2. inserir um novo elemento entre as posições k e $k+1$,
3. remover o k-ésimo elemento,
4. concatenar duas sequências,
5. copiar uma sequência,
6. determinar o tamanho de uma sequência,
7. buscar um elemento que satisfaz um critério de busca,
8. ordenar uma sequência,
9. aplicar um ou vários procedimentos a todos ou parte dos elementos da sequência,
10. ...

• A **estrutura de dados** empregada depende da aplicação. Por exemplo, inserir dado num vetor é mais complicado do que numa lista.

• Considerações importantes: eficiência, memória utilizada, facilidade de implementação e manutenção, generalidade etc.

• A **estrutura de dados** empregada depende da aplicação. Por exemplo, inserir dado num vetor é mais complicado do que numa lista.

• Considerações importantes: eficiência, memória utilizada, facilidade de implementação e manutenção, generalidade etc.

Listas lineares especiais

- Exemplos de listas especiais:

- Pilha (*stack*): inserção e remoção na mesma extremidade da estrutura.

- Fila (*queue*): inserção numa extremidade (fim) e remoção na outra (início).

- Fila dupla (*double ended queue*): inserção e remoção em ambas as extremidades da estrutura.

- Fila ordenada (*ordered queue*): conjunto de filas com diferentes prioridades de inserção e remoção.

Listas lineares especiais

- Exemplos de listas especiais:

- Pilha (*stack*): inserção e remoção na mesma extremidade da estrutura.

- Fila (*queue*): inserção numa extremidade (fim) e remoção na outra (início).

- Fila dupla (*double ended queue*): inserção e remoção em ambas as extremidades da estrutura.

- Fila ordenada (*ordered queue*): conjunto de filas com diferentes prioridades de inserção e remoção.

Listas lineares especiais

- Exemplos de listas especiais:

- Pilha (*stack*): inserção e remoção na mesma extremidade da estrutura.

- Fila (*queue*): inserção numa extremidade (fim) e remoção na outra (início).

- Fila dupla (*double ended queue*): inserção e remoção em ambas as extremidades da estrutura.

- Fila ordenada (*ordered queue*): conjunto de filas com diferentes prioridades de inserção e remoção.

Listas lineares especiais

- Exemplos de listas especiais:

- Pilha (*stack*): inserção e remoção na mesma extremidade da estrutura.

- Fila (*queue*): inserção numa extremidade (fim) e remoção na outra (início).

- Fila dupla (*double ended queue*): inserção e remoção em ambas as extremidades da estrutura.

- Fila ordenada (*ordered queue*): conjunto de filas com diferentes prioridades de inserção e remoção.

Listas lineares especiais

• Exemplos de listas especiais:

• Pilha (*stack*): inserção e remoção na mesma extremidade da estrutura.

• Fila (*queue*): inserção numa extremidade (fim) e remoção na outra (início).

• Fila dupla (*double ended queue*): inserção e remoção em ambas as extremidades da estrutura.

• Fila ordenada (*ordered queue*): conjunto de filas com diferentes prioridades de inserção e remoção.

Pilha

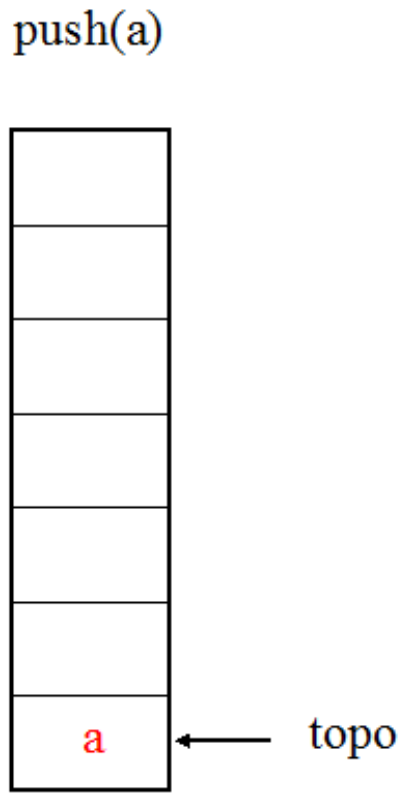
• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

• Ilustração:

Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

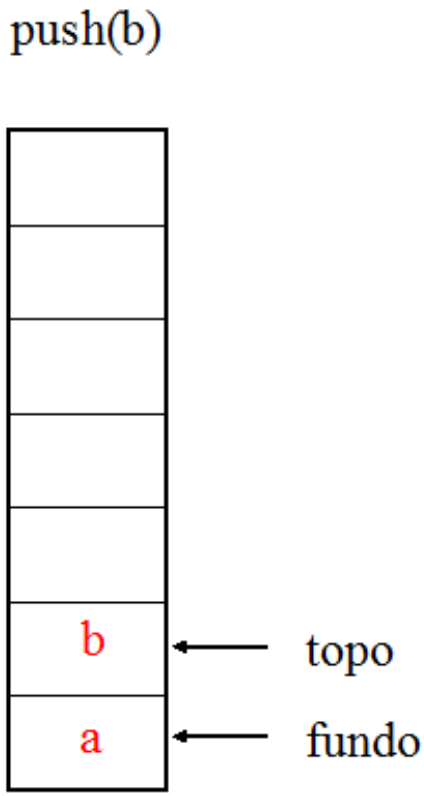
• Ilustração:



Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

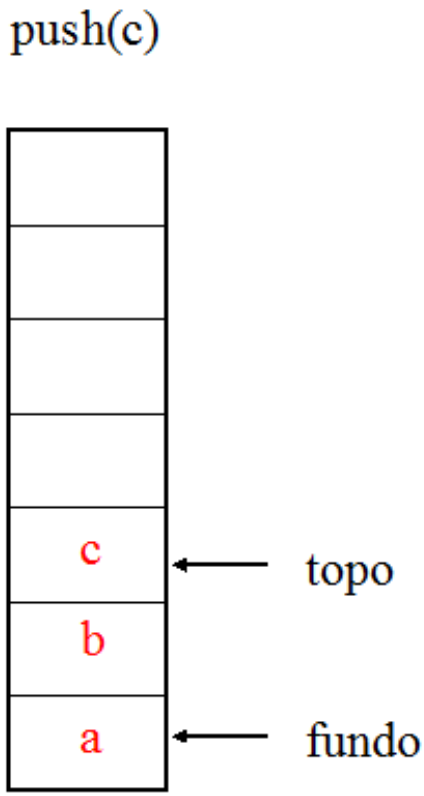
• Ilustração:



Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

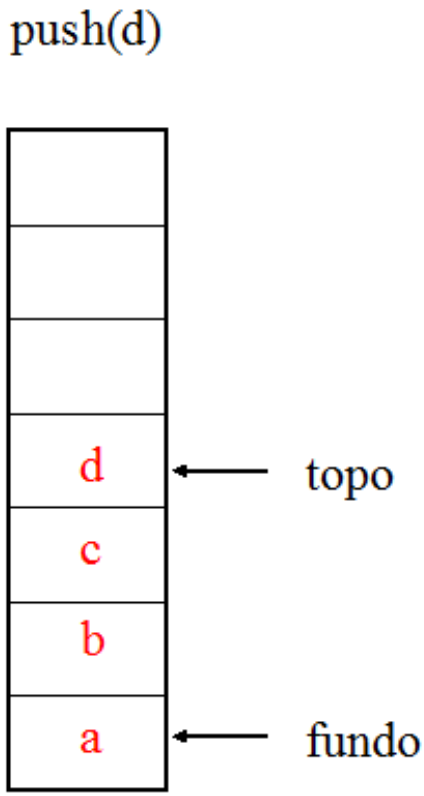
• Ilustração:



Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

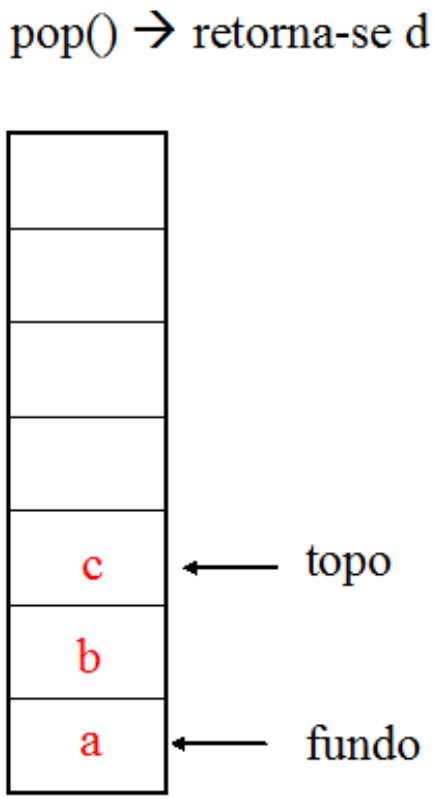
• Ilustração:



Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

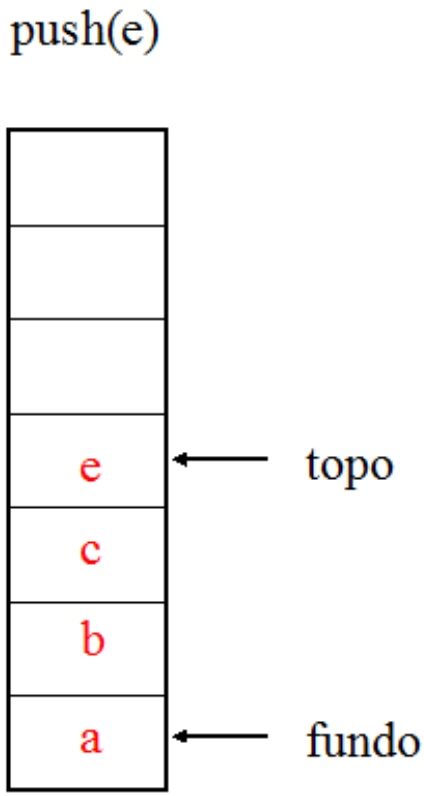
• Ilustração:



Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

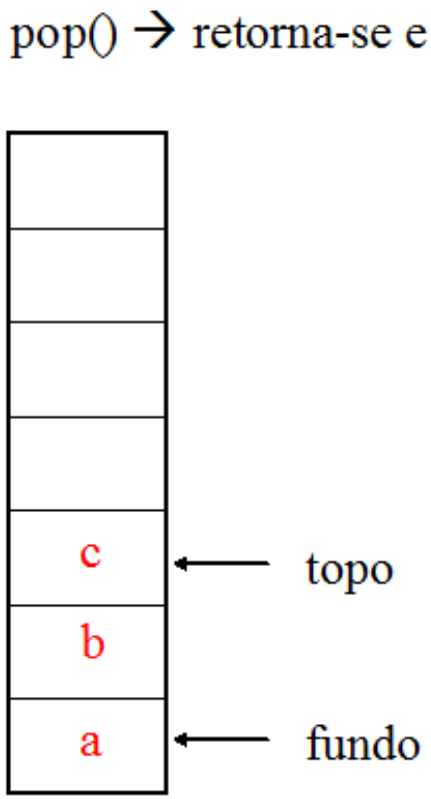
• Ilustração:



Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

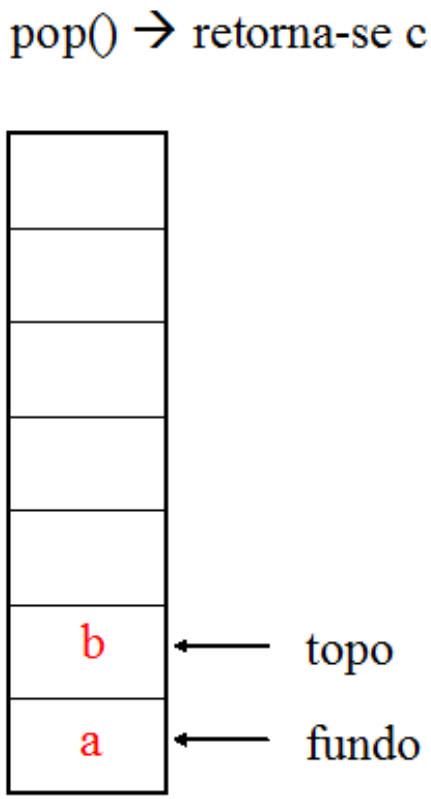
• Ilustração:



Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

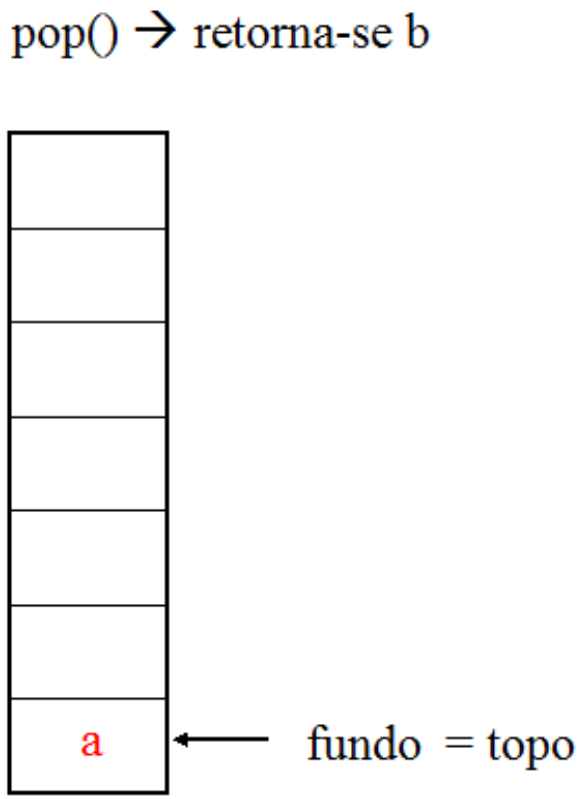
• Ilustração:



Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

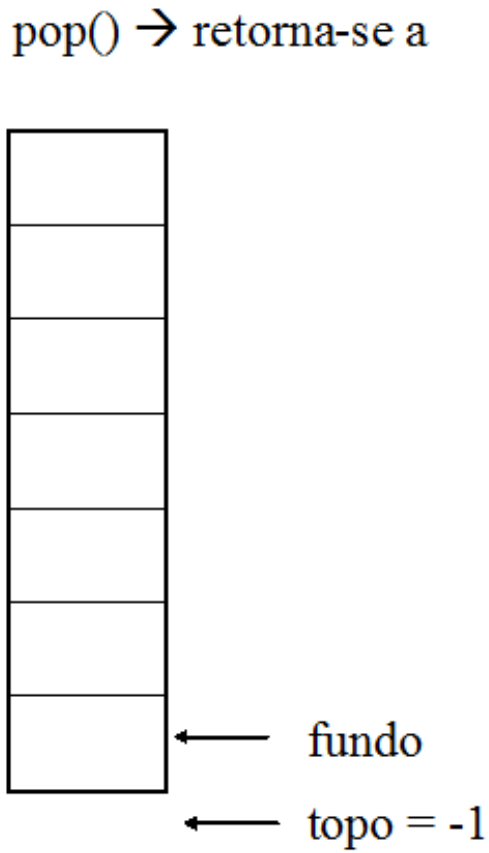
• Ilustração:



Pilha

• Definição: uma pilha é uma lista linear na qual as operações de inserção e remoção são efetuadas sempre num mesmo extremo denominado *topo* da pilha.

• Ilustração:



Interface

- A interface do TAD pilha contém funções tais como:
  - criar uma pilha vazia,
  - inserir um elemento,
  - remover um elemento,
  - verificar se a pilha está vazia,
  - liberar a estrutura da pilha.

Interface

- A interface do TAD pilha contém funções tais como:
  - criar uma pilha vazia,
  - inserir um elemento,
  - remover um elemento,
  - verificar se a pilha está vazia,
  - liberar a estrutura da pilha.

Interface

- A interface do TAD pilha contém funções tais como:
  - criar uma pilha vazia,
  - inserir um elemento,
  - remover um elemento,
  - verificar se a pilha está vazia,
  - liberar a estrutura da pilha.

Interface

- A interface do TAD pilha contém funções tais como:
  - criar uma pilha vazia,
  - inserir um elemento,
  - remover um elemento,
  - verificar se a pilha está vazia,
  - liberar a estrutura da pilha.

Interface

- A interface do TAD pilha contém funções tais como:
  - criar uma pilha vazia,
  - inserir um elemento,
  - remover um elemento,
  - verificar se a pilha está vazia,
  - liberar a estrutura da pilha.

Arquivo *.h*

- Exemplo:

```
typedef struct pilha Pilha ;
typedef enum{false, true} boolean ;

Pilha *pilha_cria(void) ;
void pilha_push(Pilha *p, float v) ;
float pilha_pop(Pilha *p) ;
boolean pilha_vazia(Pilha *p);
void pilha_libera(Pilha *p) ;
```

O módulo *Pilha.c*

- Formas de implementação:

$\Rightarrow$ Implementação em vetor,

$\Rightarrow$ Implementação em lista ligada

- Implementação em vetor:

$\rightarrow$ Se a pilha está armazenada num vetor $p[0..N-1]$, então a parte do vetor ocupada pela pilha é $p[0..t-1]$.

$\rightarrow$ O índice $t-1$ define o **topo** da pilha.

$\rightarrow$ A pilha está **vazia** se t vale 0 e **cheia** se t vale N.

$\rightarrow$ **Remover** ou **desempilhar** um elemento da pilha corresponde ao comando: $x = p[--t]$ $(t-=1;\ x = p[t];)$

$\rightarrow$ **Consultar** a pilha sem desempilhar corresponde ao comando: $x = p[t-1];$.

$\rightarrow$ **Inserir** ou **empilhar** um elemento y corresponde ao comando: $p[t++] = y;$ $(p[t] = y;\ t+=1;)$.

- Implementação em vetor:

→ Se a pilha está armazenada num vetor $p[0..N-1]$, então a parte do vetor ocupada pela pilha é $p[0..t-1]$.

→ O índice $t-1$ define o **topo** da pilha.

→ A pilha está **vazia** se t vale 0 e **cheia** se t vale N.

→ **Remover** ou **desempilhar** um elemento da pilha corresponde ao comando: $x = p[--t]$ $(t -= 1;\ x = p[t];)$

→ **Consultar** a pilha sem desempilhar corresponde ao comando: $x = p[t-1];$.

→ **Inserir** ou **empilhar** um elemento y corresponde ao comando: $p[t++] = y;$ $(p[t] = y;\ t += 1;)$.

- Implementação em vetor:

→ Se a pilha está armazenada num vetor $p[0..N-1]$, então a parte do vetor ocupada pela pilha é $p[0..t-1]$.

→ O índice $t-1$ define o **topo** da pilha.

→ A pilha está **vazia** se t vale 0 e **cheia** se t vale N.

→ **Remover** ou **desempilhar** um elemento da pilha corresponde ao comando: $x = p[--t]$ $(t -= 1;\ x = p[t];)$

→ **Consultar** a pilha sem desempilhar corresponde ao comando: $x = p[t-1];$.

→ **Inserir** ou **empilhar** um elemento y corresponde ao comando: $p[t++] = y;$ $(p[t] = y;\ t += 1;)$.

- Implementação em vetor:

$\rightarrow$ Se a pilha está armazenada num vetor $p[0..N-1]$, então a parte do vetor ocupada pela pilha é $p[0..t-1]$.

$\rightarrow$ O índice $t-1$ define o **topo** da pilha.

$\rightarrow$ A pilha está **vazia** se t vale 0 e **cheia** se t vale N.

$\rightarrow$ **Remover** ou **desempilhar** um elemento da pilha corresponde ao comando: $x = p[--t]$ $(t -= 1;\ x = p[t];)$

$\rightarrow$ **Consultar** a pilha sem desempilhar corresponde ao comando: $x = p[t-1];$.

$\rightarrow$ **Inserir** ou **empilhar** um elemento y corresponde ao comando: $p[t++] = y;$ $(p[t] = y;\ t += 1;)$.

- Implementação em vetor:

→ Se a pilha está armazenada num vetor $p[0..N-1]$, então a parte do vetor ocupada pela pilha é $p[0..t-1]$.

→ O índice $t-1$ define o **topo** da pilha.

→ A pilha está **vazia** se t vale 0 e **cheia** se t vale N.

→ **Remover** ou **desempilhar** um elemento da pilha corresponde ao comando: $x = p[--t]$ $(t -= 1;\ x = p[t];)$

→ **Consultar** a pilha sem desempilhar corresponde ao comando: $x = p[t-1];$.

→ **Inserir** ou **empilhar** um elemento y corresponde ao comando: $p[t++] = y;$ $(p[t] = y;\ t += 1;)$.

- Implementação em vetor:

$\rightarrow$ Se a pilha está armazenada num vetor $p[0..N-1]$, então a parte do vetor ocupada pela pilha é $p[0..t-1]$.

$\rightarrow$ O índice $t-1$ define o **topo** da pilha.

$\rightarrow$ A pilha está **vazia** se t vale 0 e **cheia** se t vale N.

$\rightarrow$ **Remover** ou **desempilhar** um elemento da pilha corresponde ao comando: $x = p[--t]$ $(t -= 1;\ x = p[t];)$

$\rightarrow$ **Consultar** a pilha sem desempilhar corresponde ao comando: $x = p[t-1];$.

$\rightarrow$ **Inserir** ou **empilhar** um elemento y corresponde ao comando: $p[t++] = y;$ $(p[t] = y;\ t += 1;)$.

Exemplo

- A estrutura do tipo pilha:

```
#define MAX 1000   /* numero maximo de elementos */

struct pilha {
 int t ;
 float vet[MAX] ;
}
```

Exemplo de funções

- Inicializar a pilha

```
Pilha *pilha_cria(void) {
 Pilha *p = (Pilha *)malloc(sizeof(Pilha)) ;
 p->t = 0;       /* inicializa pilha vazia */

 return p ;
}
```

Exemplo de funções

- Verificar se pilha está vazia

```
boolean pilha_vazia(Pilha *p) {

 return (p->t == 0) ;
}
```

Exemplo de funções

- Inserir um elemento na pilha

```
void pilha_push(Pilha *p, float v) {
 if(p->t == MAX) {    /* capacidade esgotada */
  printf(‘‘Pilha cheia\n’’) ;
  exit(1) ;                  /* aborta programa */

 }

 /* insere elemento na proxima posicao livre */
 p->vet[p->t] = v ;
 p->t++ ;
}
```

Exemplo de funções

- Remover um elemento da pilha

```
float pilha_pop(Pilha *p) {
 float v ;
 if (pilha_vazia(p)) {
   printf(‘‘Pilha vazia. \n’’) ;
   exit(1) ;      /* aborta programa * /
 }

 /* retira elemento do topo */
 v = p->vet[p->t - 1] ;
 p->t-- ;

 return v ;
}
```

Exemplo de funções

- Liberar a memória alocada pela pilha

```
void pilha_libera(Pilha *p) {

 free(p) ;
}
```

Exemplo de aplicação

• Algumas aplicações podem ser implementadas facilmente a partir de um vetor e dos comandos:

→ **Empilhar** um elemento y: $p[t++] = y$;

→ **Desempilhar** um elemento da pilha: $x = p[--t]$

→ **Consultar** a pilha sem desempilhar: $x = p[t-1]$;.

• Exemplo: verificação de balanceamento correto de parênteses e chaves.

→ balanceamento correto: (({ ( ) }))

→ balanceamento incorreto: ({) }

Exemplo de aplicação

• Algumas aplicações podem ser implementadas facilmente a partir de um vetor e dos comandos:

→ **Empilhar** um elemento y: $p[t++] = y$;

→ **Desempilhar** um elemento da pilha: $x = p[--t]$

→ **Consultar** a pilha sem desempilhar: $x = p[t-1]$;.

• Exemplo: verificação de balanceamento correto de parênteses e chaves.

→ balanceamento correto: (({ ( ) }))

→ balanceamento incorreto: ({) }

Exemplo de aplicação

• Algumas aplicações podem ser implementadas facilmente a partir de um vetor e dos comandos:

→ **Empilhar** um elemento y: $p[t++] = y$;

→ **Desempilhar** um elemento da pilha: $x = p[--t]$

→ **Consultar** a pilha sem desempilhar: $x = p[t-1]$;.

• Exemplo: verificação de balanceamento correto de parênteses e chaves.

→ balanceamento correto: (({ ( ) }))

→ balanceamento incorreto: ({) }

Exemplo de aplicação

• Algumas aplicações podem ser implementadas facilmente a partir de um vetor e dos comandos:

→ **Empilhar** um elemento y: $p[t++] = y$;
→ **Desempilhar** um elemento da pilha: $x = p[--t]$
→ **Consultar** a pilha sem desempilhar: $x = p[t-1]$;.

• Exemplo: verificação de balanceamento correto de parênteses e chaves.

→ balanceamento correto: (({ ( ) }))

→ balanceamento incorreto: ({) }

Exemplo de aplicação

- Balanceamento de parenteses:

Correto	Incorreto
ϵ	(
( )	)
[( )]	[ )
[ ] ( ) [( ) [ ]]	( ) ( ) [
((([[[ ]]])))	) (

Exemplo de aplicação

Pilha	*Resto da seqüência*
Vazia	([([] [( )])])
(	[([] [( )])])
([	([] [( )])])
([(	[] [( )])])
([([	] [( )])])
([(	[( )])])
([([	( )])])
([([(	)])])
([([	])])
([(	)])
([	])
(	)
Vazia	ϵ

Programa

```
/* Verifica se sequencia de parenteses e chaves esta
   correta. */

int SeqCorreta(char s[]) {
  char *p ;   int t ;
  int n, i ;
  n = strlen(s) ;
  p = (char *)malloc(n*sizeof(char)) ;  /* p eh uma pilha */
  t = 0 ;
```

```
    for(i=0; s[i]!='\0'; i++) {
              switch(s[i]) {
                  case ')': if(t!=0 && p[t-1] == '(')  --t;
                            else return 0 ;
                            break ;
                  case '}': if(t!=0 && p[t-1] == '{') --t ;
                            else return 0 ;
                            break ;
                  default: p[t++] = s[i] ;
              }
    }
    free(p) ;
    return t == 0 ;          /* retorna 1 se pilha vazia */


}   /* fim da funcao */
```

Implementação do TAD pilha com lista ligada e nó cabeça

• Considera-se o caso em que o número máximo de elementos da pilha não é conhecido a priori.

• A estrutura de um nó da pilha

```
struct NoPilha {
 float  info ;
 struct NoPilha *prox ;
} ;

typedef struct NoPilha Pilha, *ApNoPilha ;
```

Exemplo de funções

- Inicializar a pilha com nó cabeça

```
Pilha *pilha_cria(void) {
 Pilha *p = (Pilha *)malloc(sizeof(Pilha)) ;
 p->info = -1 ;          /* inicializa no cabeca da pilha */
 p->prox = NULL ;

 return p ;
}
```

Exemplo de funções

- Empilhar um valor

```
void pilha_push(Pilha *p, float v) {

 ApNoPilha q ;
 q = (Pilha *)malloc(sizeof(Pilha)) ;
 q->info = v ;
 q->prox = p->prox ;
 p->prox = q ;           /* insere no inicio da lista */
}
```

Exemplo de funções

- Desempilhar um valor

```
float pilha_pop(Pilha *p) {

 ApNoPilha q ;
 float v ;
 if(p->prox == NULL) {
   printf(''Pilha vazia\n'') ;
   exit(1) ;                  /* aborta programa */
 }
 q = p->prox ;
 v = q->info ;
 p->prox = q->prox ;
 free(q) ;

 return v ;
}
```

Exemplo de funções

- Verificar se a pilha eatá vazia

```
boolean pilha_vazia(Pilha *p) {

  return (p->prox == NULL) ;
}
```

Exemplo de funções

- Liberar a memória alocada pela pilha

```
void pilha_libera(Pilha *p) {

 ApNoPilha t ;
 ApNoPilha q = p->prox ;

 while(q != NULL) {
  t = q->prox ;
  free(q) ;
  q = t ;
 }

 free(p) ;
}
```

- Outros exemplos de aplicações
  - Soma de grandes valores inteiros
  - Avaliação de expressões na forma pós-fixa / pré-fixa
  - Conversão de infixa para pós-fixa / pré-fixa
  - Pilha de execução de programa

- Outros exemplos de aplicações
  - Soma de grandes valores inteiros
  - Avaliação de expressões na forma pós-fixa / pré-fixa
  - Conversão de infixa para pós-fixa / pré-fixa
  - Pilha de execução de programa

- Outros exemplos de aplicações
  - Soma de grandes valores inteiros
  - Avaliação de expressões na forma pós-fixa / pré-fixa
  - Conversão de infixa para pós-fixa / pré-fixa
  - Pilha de execução de programa

- Outros exemplos de aplicações
  - Soma de grandes valores inteiros
  - Avaliação de expressões na forma pós-fixa / pré-fixa
  - Conversão de infixa para pós-fixa / pré-fixa
  - Pilha de execução de programa

Exemplo de aplicação

- Notações para expressões aritméticas:

infixa	*pós-fixa*	*pré-fixa*
a	a	a
$a + b$	$ab+$	$+ab$
$a + b * c$	$abc * +$	$+a * bc$
$(a + b) * c$	$ab + c*$	$* + abc$

Exemplo de aplicação: avaliação de expressão

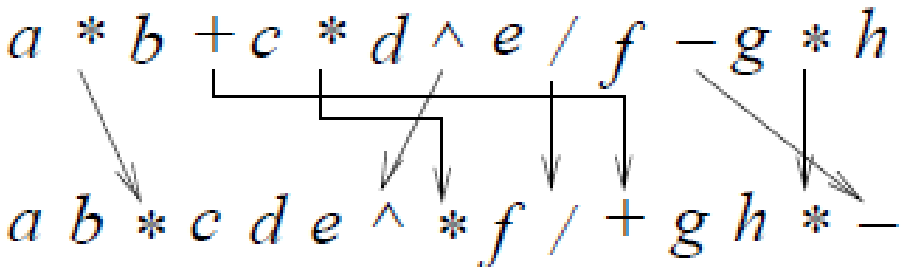
- Notação pós-fixa: 3 5 + 2 * 10 3 – 2/ – ⇒ (3+5)*2-(10-3)/2

Estados da pilha:

Vazia	$3\ \ 5+2\star 10\ \ 3-2/-$
3	$5+2\star 10\ \ 3-2/-$
3 5	$+2\star 10\ \ 3-2/-$
8	$2\star 10\ \ 3-2/-$
8 2	$\star 10\ \ 3-2/-$
16	$10\ \ 3-2/-$
16 10	$3-2/-$
16 10 3	$-2/-$
16 7	$2/-$
16 7 2	$/-$
16 3	$-$
13	Vazia

Exemplo de aplicação

- Transformação de notação infixa para pós-fixa



Exemplo de aplicação

Saída	*Pilha*	*Entrada*
		$a * b + c * d \wedge e / f - g * h$
a		$* b + c * d \wedge e / f - g * h$
a	$*$	$b + c * d \wedge e / f - g * h$
ab	$*$	$+ c * d \wedge e / f - g * h$
$ab*$		$+ c * d \wedge e / f - g * h$
$ab*$	$+$	$c * d \wedge e / f - g * h$
$ab * c$	$+$	$* d \wedge e / f - g * h$
$ab * c$	$+*$	$d \wedge e / f - g * h$
$ab * cd$	$+*$	$\wedge e / f - g * h$
$ab * cd$	$+ * \wedge$	$e / f - g * h$
$ab * cde$	$+ * \wedge$	$/ f - g * h$
$ab * cde\wedge$	$+*$	$/ f - g * h$

(continua)

Exemplo de aplicação

Saída	*Pilha*	*Entrada*
$ab*cde\wedge*$	$+$	$/f-g*h$
$ab*cde\wedge*$	$+/$	$f-g*h$
$ab*cde\wedge*f$	$+/$	$-g*h$
$ab*cde\wedge*f/$	$+$	$-g*h$
$ab*cde\wedge*f/+$		$-g*h$
$ab*cde\wedge*f/+$	$-$	$g*h$
$ab*cde\wedge*f/+g$	$-$	$*h$
$ab*cde\wedge*f/+g$	$-*$	h
$ab*cde\wedge*f/+gh$	$-*$	
$ab*cde\wedge*f/+gh*$	$-$	
$ab*cde\wedge*f/+gh*-$		

- Bibliografia:

– Paulo Feofiloff. Algoritmos em linguagem C. Elsevier 2009

– Apostila dos profs. Tomasz e Lucchesi disponível na página do curso