



SCC-223 Estruturas de Dados I

Pilhas

Profa. Elaine Parros Machado de Sousa



Pilhas

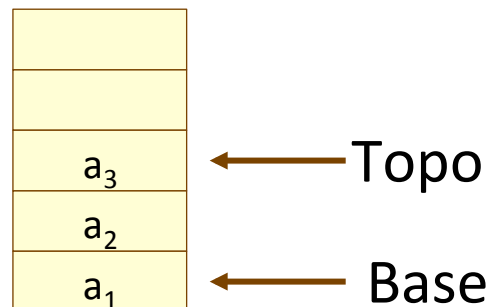
- O quê são?
 - Exemplos do mundo real...
 - pilha de livros
 - pilha de bandejas do “Bandejão”
 - pilha de produtos no supermercado
 - ...
- Para quê servem?
 - Auxiliam em problemas práticos em computação
 - exemplos?

Aplicações

- Exemplos de aplicações de pilhas
 - O botão “back” de um navegador web ou a opção “undo” de um editor de textos
 - Controle de chamada de procedimentos
 - Compilador
 - Estrutura de dados auxiliar em alguns algoritmos clássicos, como a busca em profundidade
 - Avaliação de expressões aritméticas com notação “Posfixa”
 - ...

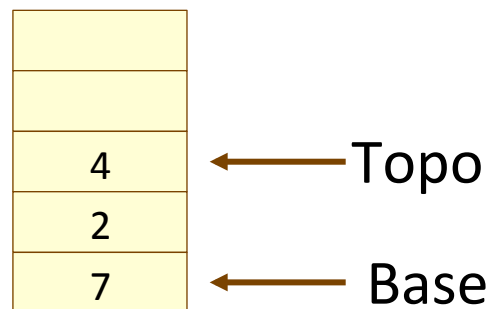
Pilhas - definição

- **Pilhas** são estruturas de dados (listas lineares) nas quais as inserções e remoções são feitas na mesma extremidade, chamada **topo**
- Dada uma Pilha $P = (a_1, a_2, \dots, a_n)$:
 - a_1 é o elemento da **base** da pilha;
 - a_n é o elemento do **topo**
 - a_{i+1} está acima de a_i na pilha.



Pilhas - definição

- Nas pilhas elementos são adicionados no topo e removidos do topo
 - Política *Last-In/First-Out* (LIFO)



TAD Pilha

- Operações principais
 - **push(P, x)**: insere o elemento **x** no topo de **P** (empilhar)
 - **pop(P)**: remove o elemento do topo de **P**, e retorna esse elemento (desempilhar)

push(1, P);

push(2, P);

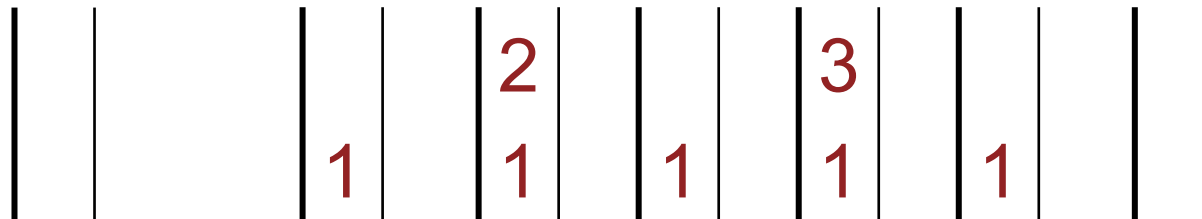
pop(P);

push(3, P);

pop(P);

pop(P);

P=NULL



TAD Pilha

- Operações auxiliares
 - **criar(P)**: cria uma pilha P vazia
 - **apagar(P)**: apaga a pilha P da memória
 - **esvaziar(P)**: remove todos os elementos da pilha P e deixá-la vazia
 - **topo(P)**: retorna o elemento do topo de P, sem remover
 - **tamanho(P)**: retorna o número de elementos em P
 - **vazia(P)**: indica se a pilha P está vazia
 - **cheia(P)**: indica se a pilha P está cheia (útil para implementações estáticas).



Exemplo prático

- Elabore um algoritmo para converter um número decimal em sua respectiva representação binária usando o TAD Pilha.
- Implemente o algoritmo em C.


```

#include <stdio.h>
#include "Pilha.h"

int main(void){
    int resto, quociente;
    PILHA *p;

    p = pilha_criar();

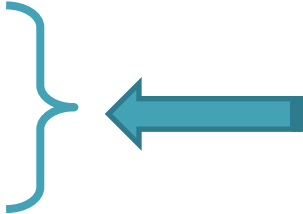
    printf("\n Entre um numero inteiro: ");
    scanf("%d", &quociente);

    while(quociente != 0){
        resto = quociente % 2;
        quociente /= 2;
        pilha_push(p, resto);
    }

    printf("\n A representacao em binario eh: ");
    while(!pilha_vazia(p)){
        printf("%d", pilha_pop(p));
    }
    ....
}

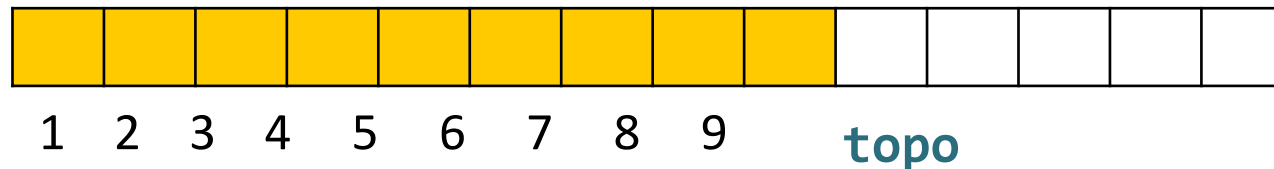
```

Pilha – Implementação

- Sequencial e estática: *arrays*
 - Encadeada e dinâmica: ponteiros
 - Encadeada e estática: *array* simulando encadeamento
 - Sequencial e dinâmica: alocação dinâmica de *array*
- 

Pilha - Implementação Estática e Sequencial

- Implementação simples
 - similar a lista sequencial estática e fila sequencial estática
- Variável **topo** mantém o controle da posição do topo da pilha e pode ser utilizada também para informar o número de elementos na pilha (tamanho)



TAD Pilha – Exemplo

```
/* interface (pilha.h)*/

#define TAM 100
#define ERRO -32000

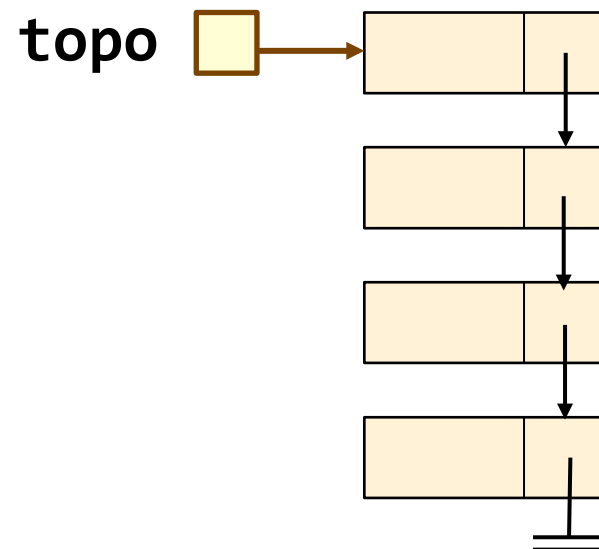
typedef int ITEM;
typedef struct pilha_ PILHA;

...

/* implementação (pilha.c)*/
struct pilha_ {
    ITEM item[TAM];
    int topo;
};
...
```

Pilha - Implementação Encadeada e Dinâmica

- Alocação dinâmica e ponteiros
 - similar à lista encadeada dinâmica e fila encadeada dinâmica
- O topo é o início da pilha



TAD Pilha – Exemplo

```
// interface (arquivo .h)
#define TAM 100
#define ERRO -32000

typedef int ITEM;
typedef struct pilha_ PILHA;

...

// implementação (arquivo .c)
#include <stdio.h> ...
#include "Pilha.h"
typedef struct no_ NO;
struct no_ {
    ITEM item;
    NO *anterior;
};

struct pilha_ {
    NO *topo;
    int tamanho;
};

...
```

TAD Pilha – Exemplo de Implementação

```
PILHA *pilha_criar() {  
    PILHA *pilha = (PILHA *) malloc(sizeof (PILHA));  
    if (pilha != NULL) {  
        pilha->topo = NULL;  
        pilha->tamanho = 0;  
    }  
    return (pilha);  
}
```

TAD Pilha – Exemplo de Implementação

```
void pilha_apagar(PILHA **pilha) {  
    NO *paux;  
    if ( ((*pilha) != NULL) && (!pilha_vazia(*pilha)) ) {  
  
        while (pilha->topo != NULL) {  
            paux = (*pilha)->topo;  
            (*pilha)->topo = (*pilha)->topo->anterior;  
            paux->anterior = NULL;  
            free(paux);  
            paux = NULL;  
        }  
    }  
    free(*pilha);  
    *pilha = NULL;  
}
```


TAD Pilha – Exemplo de Implementação

```
int pilha_push(PILHA *pilha, ITEM item) {
    NO *pnovo = (NO *) malloc(sizeof (NO));
    if (pnovo != NULL) {
        .... // exercício
        return (1);
    }
    return (ERRO);
}

/*****
ITEM pilha_pop(PILHA *pilha) {
    if ((pilha != NULL) && (!pilha_vazia(pilha)) ){
        ... // exercício

        return (item);
    }
    return (ERRO);
}
```

Estática versus Dinâmica

- Complexidade das operações principais?

Operação	Sequencial	Encadeada
Criar	$O(1)$	$O(1)$
Apagar	$O(1)$	$O(n)$
Empilhar (push)	$O(1)$	$O(1)$
Desempilhar (pop)	$O(1)$	$O(1)$
Topo	$O(1)$	$O(1)$
Vazia	$O(1)$	$O(1)$
Tamanho	$O(1)$	$O(1)$ (com contador)

Aplicação de Pilhas

- Avaliação de expressões aritméticas
 - Notação polonesa reversa (**posfixa**)
 - Operadores sucedem os operandos
 - Dispensa o uso de parênteses
 - **AB * CD/-** = $(A * B) - (C/D)$

Aplicação de Pilhas

- Expressões na notação **posfixa** podem ser avaliadas utilizando uma **pilha**.
 - A expressão é avaliada de esquerda para a direita
 - Os operandos são empilhados
 - Os operadores fazem com que os operandos sejam desempilhados, o cálculo seja realizado e o resultado empilhado

Exercício

- 1) Para a expressão abaixo, indique a sequência de operações necessárias para computar o resultado final usando pilha. Ilustre o estado da pilha após cada operação. efeitos na pilha

6 2 / 3 4 * + 3 - = 6 / 2 + 3 * 4 - 3

- 2) Implemente essa funcionalidade usando o TAD Pilha. Qual implementação mais adequada para esse problema: estática ou dinâmica.

Exercício

- Considere o problema de decidir se uma dada sequência de parênteses e chaves é bem formada. Por exemplo, a sequência abaixo:

(() { () }) é bem-formada,

enquanto a sequência

({ }) é malformada.

- Suponha que a sequência de parênteses e chaves está armazenada em uma cadeia de caracteres **s**. Usando o TAD Pilha, escreva uma função **bem_formada()** que receba a cadeia de caracteres **s** e:
 - devolva **1** se **s** contém uma sequência bem-formada de parênteses e chaves
 - devolva **0** se a sequência está malformada.