

Pilha

SCC-202 – Algoritmos e Estruturas de
Dados I

Pilha

- O que é?
- Para que serve?



Problema: chamada de sub-rotinas

Rotina A

```
1 print "A"  
2 call C  
3 call B  
4 call D  
5 return
```

Rotina B

```
1 call C  
2 print "B"  
3 call D  
4 call C  
5 return
```

Rotina C

```
1 print "C"  
2 call D  
3 return
```

Rotina D

```
1 print "D"  
2 return
```

Qual o resultado da execução da rotina A?

Problema: chamada de sub-rotinas

Rotina A

```
1 print "A"  
2 call C  
3 call B  
4 call D  
5 return
```

Rotina B

```
1 call C  
2 print "B"  
3 call D  
4 call C  
5 return
```

Rotina C

```
1 print "C"  
2 call D  
3 return
```

Rotina D

```
1 print "D"  
2 return
```

Qual o resultado da execução da rotina A?

Qual a dificuldade para se fazer esse cálculo?

Problema: chamada de sub-rotinas

Rotina A

```
1 print "A"  
2 call C  
3 call B  
4 call D  
5 return
```

Rotina B

```
1 call C  
2 print "B"  
3 call D  
4 call C  
5 return
```

Rotina C

```
1 print "C"  
2 call D  
3 return
```

Rotina D

```
1 print "D"  
2 return
```

Qual o resultado da execução da rotina A?

Qual a dificuldade para se fazer esse cálculo?

Possíveis soluções?

Problema: chamada de sub-rotinas

1. Um computador está executando a rotina X e, durante a execução de X, encontra uma chamada à rotina Y
2. A execução de X é pausada e inicia-se a execução de Y
3. Quando se termina a execução de Y, o computador deve saber o que fazer, isto é, onde voltar na rotina X

Problema: chamada de sub-rotinas

■ Dificuldade

- O que estava sendo executado quando uma sub-rotina foi interrompida? Para onde voltar agora que se chegou ao fim de uma sub-rotina?

■ Solução

- A cada chamada de sub-rotina, armazenar o endereço de retorno (rotina e número da linha, por exemplo)
- Como armazenar o endereço de retorno de chamadas sucessivas? Solução → **pilha**

Pilha (*stack*)

■ Definição

- *Estrutura para armazenar um conjunto de elementos que funciona da seguinte forma*
 - Novos elementos sempre entram no “topo” da pilha
 - O único elemento que se pode retirar da pilha em um dado momento é o elemento do topo

■ Para que serve

- Para modelar situações em que é preciso “guardar para mais tarde” vários elementos e “lembrar” sempre do último elemento armazenado

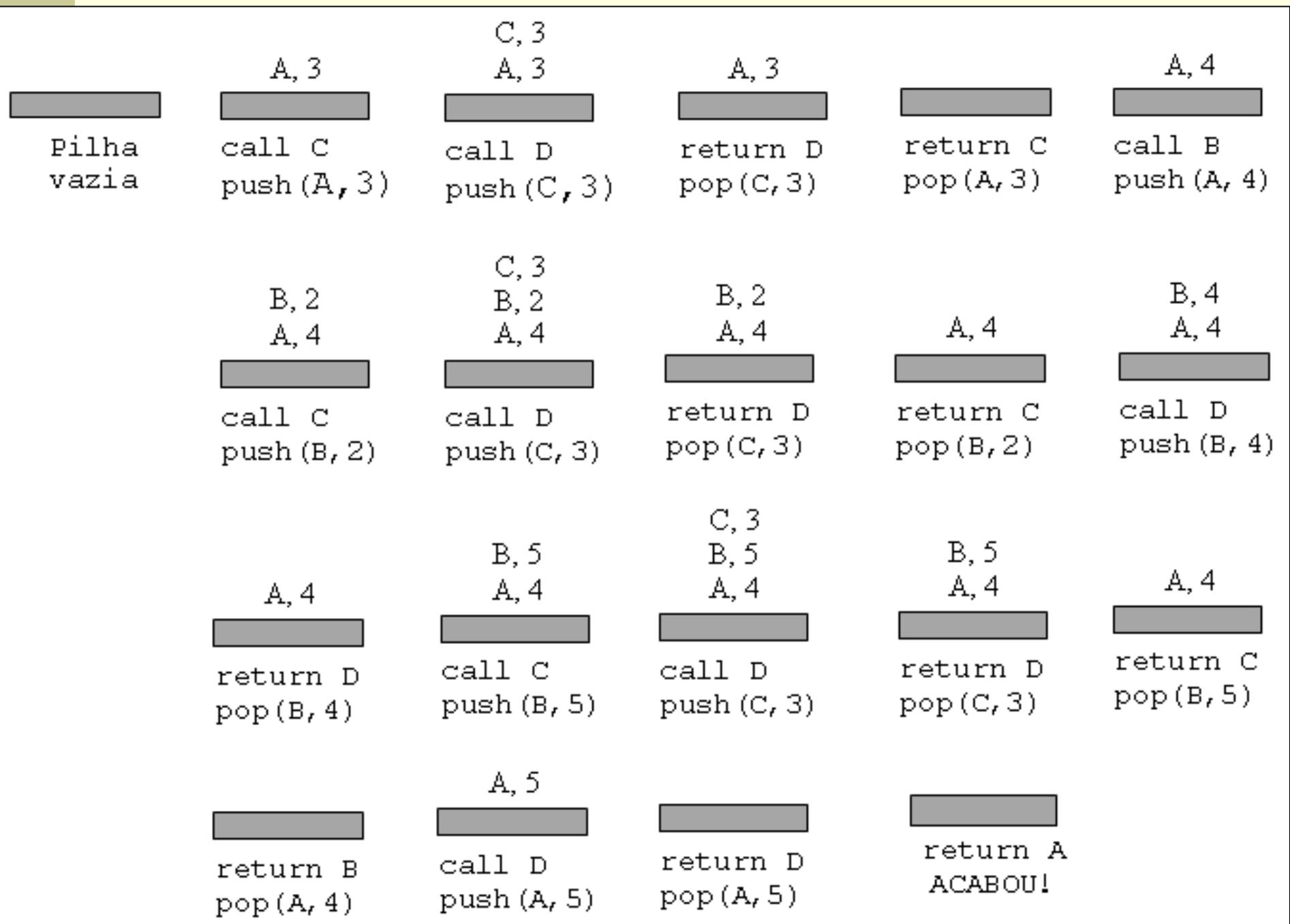
■ L.I.F.O.

- *Last In, First Out*

Problema: chamada de sub-rotinas

- A cada comando call
 - Empilha (*push*) o endereço para retornar depois
 - Passa a executar a nova sub-rotina

- A cada comando return
 - Desempilha (pop) o último endereço armazenado
 - Passa a executar a partir do endereço desempilhado



Problema: chamada de sub-rotinas

- Resultado

- A, C, D, C, D, B, D, C, D, D

Pilha

- Operações usuais
 - $\text{Push}(P,X)$: empilha o valor da variável X na pilha P
 - $\text{Pop}(P,X)$: desempilha P e retorna em X o valor do elemento que estava no topo de P
 - $X=\text{top}(P)$: acessa o valor do elemento do topo de P , sem desempilhar
 - $\text{Create}(P)$: cria uma pilha vazia P
 - $Y=\text{IsEmpty}(P)$: Y recebe *true* se a pilha estiver vazia; *false* caso contrário
 - $\text{Empty}(P)$: esvazia uma pilha P

Exercício

Variáveis

P: pilha

N: inteiro {número a ser convertido}

X: inteiro {resto da divisão}

Início do algoritmo

leia N

create(P)

repita

 X=resto(N,2)

 push(P,X)

 N=quociente(N,2)

até que (N=0)

escreva "o resultado é "

enquanto (IsEmpty(P)=falso) faça

 pop(P,X)

 escreva X

Fim

Implementação da pilha

■ Alocação seqüencial

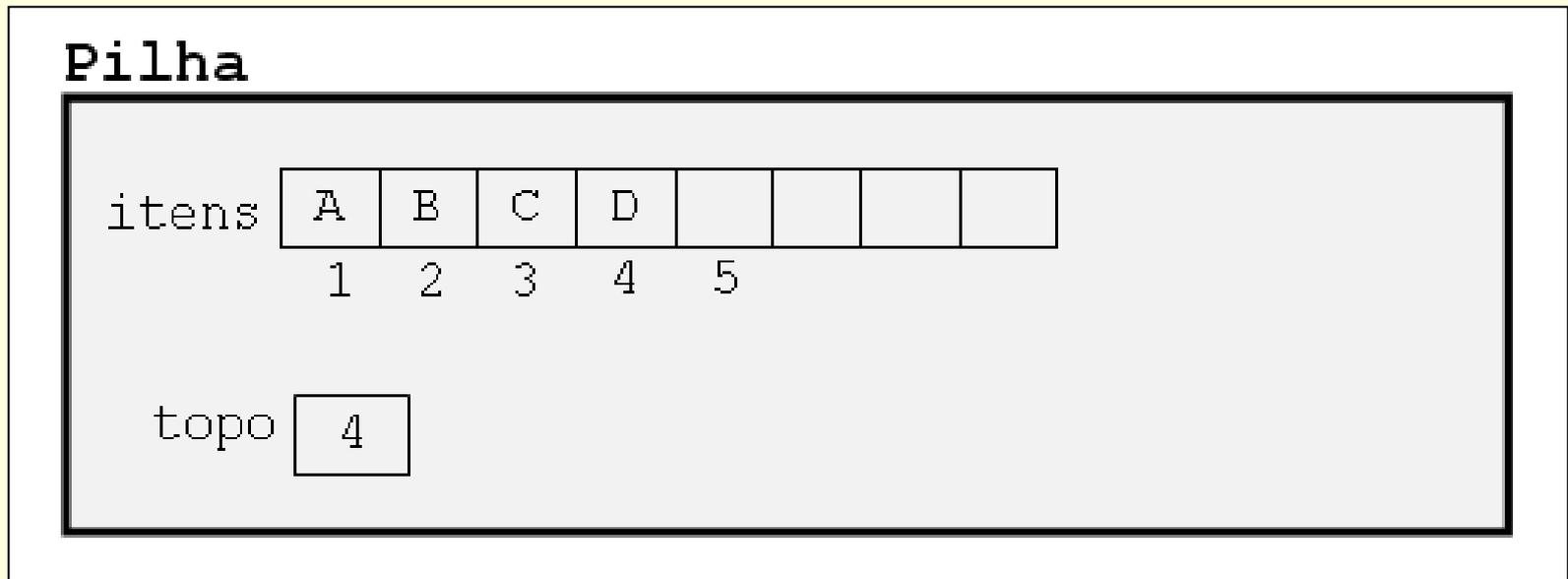
- Os elementos da pilha ficam, necessariamente, em seqüência (um ao lado do outro) na memória

■ Alocação estática

- Todo o espaço de memória a ser utilizado pela pilha é reservado (alocado) em tempo de compilação
- Todo o espaço reservado permanece reservado durante todo o tempo de execução do programa, independentemente de estar sendo efetivamente usado ou não

Implementação da pilha

- Seqüencial e estática



Implementação da pilha

- Declaração em C

?

Implementação da pilha

■ Declaração em C

```
#define TamPilha 100

typedef int elem;

typedef struct {
    int topo;
    elem itens[TamPilha];
} Pilha;

Pilha P;
```

Exercício

- Implementar operações da pilha
 - Create
 - Empty
 - IsEmpty
 - IsFull
 - Push
 - Pop
 - Top

- Atenção: considerações sobre TAD
 - Arquivos .c e .h, parâmetros, mensagens de erro

Exercício

- Adicionar uma rotina ao TAD para verificar se duas pilhas são iguais

Exercício

- Adicionar uma rotina ao TAD para inverter a posição dos elementos de uma pilha

Editor de texto: exercício

- Considere que um editor de texto representa os caracteres digitados como uma pilha, sendo que o último caractere lido fica no topo
- Alguns comandos apagam caracteres. Por exemplo, o *backspace* apaga o último caractere lido
- Alguns comandos apagam tudo o que já foi lido anteriormente
- Considere que, no seu editor, # representa *backspace* e @ indica “apagar tudo”
- Faça um programa que execute essas ações usando o TAD pilha

```

#include <stdio.h>
#include "pilha.h"
int main(void) {
    elem c, x;
    int erro;
    Pilha P;
    Create(&P);
    printf("Digite seu texto: ");
    while ((c=getche())!='\r') {
        if (c=='#') {
            Pop(&P, &x, &erro);
            if (erro) printf("(erro) ");
            else printf("(%c desempilhado) ", x);
        }
        else if (c=='@') {
            Empty(&P);
            printf("(pilha esvaziada) ");
        }
        else {
            Push(&P, &c, &erro);
            if (erro) printf("(erro) ");
        }
    }
}

```

...

...

```

printf("\n\nDesempilhando tudo: ");
while (!IsEmpty(&P)) {
    Pop(&P, &x, &erro);
    if (erro) printf("erro ");
    else printf("%c ", x);
}
system("pause");
return 0;
}

```

POSSÍVEL SOLUÇÃO

Notação posfixa: exercício

- Avaliação de expressões aritméticas
 - Às vezes, na aritmética tradicional, faz-se necessário usar parênteses para dar o significado correto à expressão
 - $A * B - C / D \quad \rightarrow \quad (A * B) - (C / D)$
 - Notação polonesa (prefixa): operadores aparecem antes dos operandos e dispensam parênteses
 - $- * AB / CD$
 - Notação polonesa reversa (posfixa): operadores aparecem depois dos operandos
 - $AB * CD / -$

Notação posfixa: exercício

- Interpretação da notação posfixa usando pilha
 - Empilha operandos até encontrar um operador
 - Retira os operandos, calcula e empilha o resultado
 - Até que se chegue ao final da expressão

Notação posfixa: exercício

■ $AB*CD/-$

A					
A	B				
A	B	*			
$A*B$					
$A*B$	C				
$A*B$	C	D			
$A*B$	C	D	/		
$A*B$	C/D				
$A*B$	C/D	-			
$A*B-C/D$					

Exercício

- Implemente uma função que calcule o valor de uma expressão posfixa passada por parâmetro utilizando o TAD pilha

Exercício – resposta algorítmica

```
função valor(E: expressão): retorna real;
declare x real;
declare P pilha;
início
  Create(P)
  enquanto não acabou(E) faça
    início
      x=proxsimb(E);
      se x é operando então Push(P,x)
      senão início
        remove operandos; {dois pops, em geral}
        calcula o resultado da operação;
        empilhe resultado; {push}
      fim
    fim
  fim
  valor=Top(P);
fim
```

Créditos

- *Material gentilmente cedido pelo Prof. Thiago A. S. Pardo*