Onde estamos ...

1. Pilha

2. Aplicações – Estudo de Casos

Capítulo 03 – Pilhas

Pontos fundamentais a serem cobertos:

1. Contexto e motivação
2. Definição
3. Implementações
4. Exercícios

Introdução

- Uma das estruturas de dados mais simples
- Embora seja uma das estrutura de dados mais utilizadas em programação
- A pilha se *fortalece* quando combinada dentro de outras estruturas
  - Uso de pilhas na sequência de visita a nós de uma árvore
  - Dentro de outras estruturas como filas (depois)
- Há uma metáfora emprestada do mundo real, que a computação utiliza pilhas para resolver muitos problemas de forma simplificada.

Aplicações

Alguns exercícios são clássicos (e devemos implementá-los):

- Balanceamento de símbolos. Exemplo: `([aaa])`
- Conversão da notação infixa para pós-fixa
- Conversão da notação infixa para in-fixa
- Avaliação de uma expressão pós-fixa. Exemplo: `2 3 +`
- Implementações de chamadas de funções (inclusive as chamadas recursivas de funções)
- Armazenamento de páginas visitadas no navegador em uma dada janela (botão **back**)
- Sequência de comandos de um editor de texto, e depois aplique o *undo*, ou `crtl-z`
- Casamento de *tags* in HTML e XML
- As teclas ↑ e ↓ na console ou terminal do Linux, duas pilhas neste caso!

Definição

Definição

Um conjunto ordenado de itens no qual novos itens podem ser inseridos e a partir do qual podem ser eliminados em uma extremidade denominada topo da pilha.

Definição

Definição

Um conjunto ordenado de itens no qual novos itens podem ser inseridos e a partir do qual podem ser eliminados em uma extremidade denominada topo da pilha.

Definição

Uma seqüência de objetos, todos do mesmo tipo, sujeita às seguintes regras de comportamento:

1. Sempre que solicitado a remoção de um elemento, o elemento removido é o último da seqüência.
2. Sempre que solicitado a inserção de um novo elemento, o objeto é inserido no fim da seqüência (topo).

Pilha

- Uma pilha é um objeto dinâmico, constantemente mutável, onde elementos são inseridos e removidos.
- Em uma pilha, cada novo elemento é inserido no topo.
- Os elementos da pilha só podem ser retirado na ordem inversa à ordem em que foram inseridos
  - O primeiro que sai é o último que entrou (*clássico*)
  - Por essa razão, uma pilha é dita uma estrutura do tipo: LIFO(*last-in, first* ou UEPS último a entrar é o primeiro a sair.)

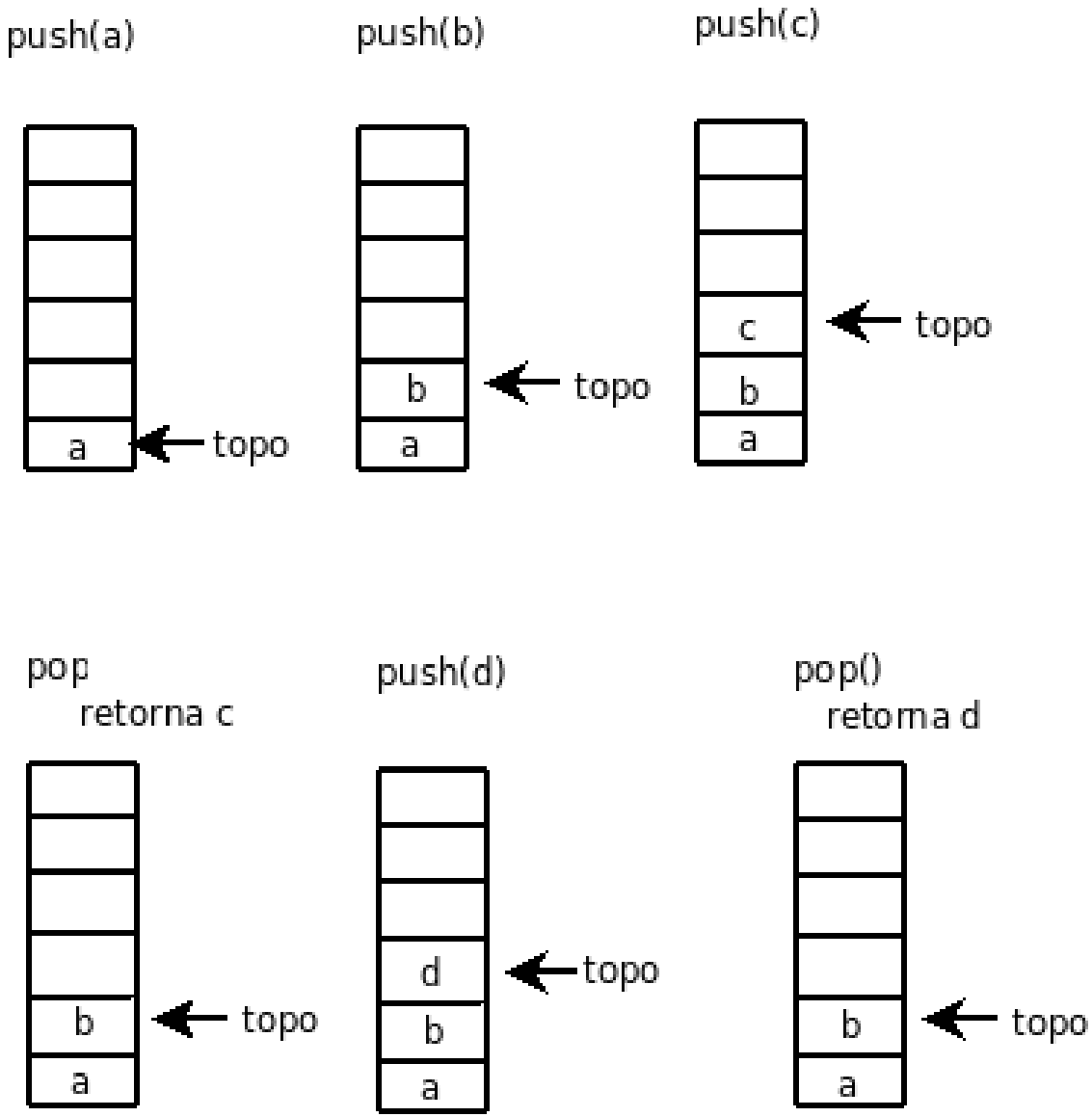
Operações básicas

- As operações básicas que devem ser implementadas em uma estrutura do tipo pilha são:

Operação	**Descrição**
push(p, e)	empilha o elemento e, inserindo-o no topo da pilha p.
pop(p)	desempilha o elemento do topo da pilha p.

Tabela 1: Operações básicas da estrutura de dados pilha.

Exemplo


push(a)
a
topo
push(b)
b
a
topo
push(c)
c
b
a
topo
pop
retorna c
b
a
topo
push(d)
d
b
a
topo
pop()
retorna d
b
a
topo

Operações auxiliares

- Além das operações básicas, temos as operações "*auxiliares*". São elas:

Operação	**Descrição**
create	cria uma pilha vazia.
empty(p)	determina se uma pilha p está ou não vazia.
free(p)	libera o espaço ocupado na memória pela pilha p.

Tabela 2: Operações auxiliares da estrutura de dados pilha.

Interface do Tipo Pilha – Típica

```
1  /* Definicao da estrutura */
2  typedef struct { DEFINA O SEU MODELO AQUI }  Pilha;
3
4  /*Aloca dinamicamente ou estaticamente a estrutura pilha,
5    inicializando seus campos e retorna seu ponteiro.*/
6  Pilha * create(void);
7
8  /*Insere o elemento e na pilha p.*/
9  void push(Pilha *p, int e);
10
11 /*Retira e retorna o elemento do topo da pilha p*/
12 int pop(Pilha *p);
13
14 /*Informa se a pilha p esta ou nao vazia.*/
15 int empty(Pilha *p);
```

Implementações

1. Baseada em um simples vetor
2. Baseada em um vetor dinâmico
3. Baseada em lista encadeada

Implementações

1. Baseada em um simples vetor
2. Baseada em um vetor dinâmico
3. Baseada em lista encadeada
4. mas todas usam ponteiros!

Implementação de Pilha com Vetor

- Normalmente as aplicações que precisam de uma estrutura pilha, é comum saber de antemão o número máximo de elementos que precisam estar armazenados simultaneamente na pilha.
- Essa estrutura de pilha tem um limite conhecido.
- Os elementos são armazenados em um vetor.
- Essa implementação é mais simples.
- Os elementos inseridos ocupam as primeiras posições do vetor.

Implementação de Pilha com Vetor

- Seja p uma pilha armazenada em um vetor VET de N elementos:
  1. O elemento vet[topo] representa o elemento do topo.
  2. A parte ocupada pela pilha é vet[0 .. topo - 1].
  3. A pilha está vazia se topo = -1.
  4. Cheia se topo = N - 1.
  5. Para desempilhar um elemento da pilha, não vazia, basta

$$x = vet[topo - -]$$

(recupera valor do topo e depois decrementa)

  6. Para empilhar um elemento na pilha, em uma pilha não cheia, basta

$$vet[+ + t] = e$$

(soma antes e depois insere)

Implementação de Pilha com Vetor

```
1  #define N 20 /* numero maximo de elementos*/
2  #include <stdio.h>
3  #include "pilha.h"
4
5  /*Define a estrutura da pilha*/
6  struct pilha{
7    int topo;              /* indica o topo da pilha */
8    int elementos[N];    /* elementos da pilha*/
9  };
10
11 Pilha* create(void){
12   Pilha* p = (Pilha * ) malloc(sizeof(Pilha));
13   p->topo = -1;       /* inicializa a pilha com 0 elementos */
14   return p;
15 }
16
```

Implementação de Pilha com Vetor

- Empilha um elemento na pilha

```
1 void push(Pilha *p, int e){
2       if (p->topo == N - 1){ /* capacidade esgotada */
3          printf("A pilha esta cheia");
4          exit(1);
5       }
6       /* insere o elemento na proxima posicao livre */
7       p->elementos[++p->topo] = e;
8 }
```

Implementação de Pilha com Vetor

- Desempilha um elemento da pilha

```
1  int pop(Pilha *p)
2  {
3        int e;
4        if (empty(p)){
5           printf("Pilha vazia.\n");
6           exit(1);
7        }
8
9        /* retira o elemento do topo */
10       e = p->elementos[p->topo--];
11       return e;
12 }
```

Implementação de Pilha com Vetor

```
1 /**
2  * Verifica se a pilha p esta vazia
3  */
4 int empty(Pilha *p)
5 {
6    return (p->t == -1);
7 }
```

Exemplos de Uso

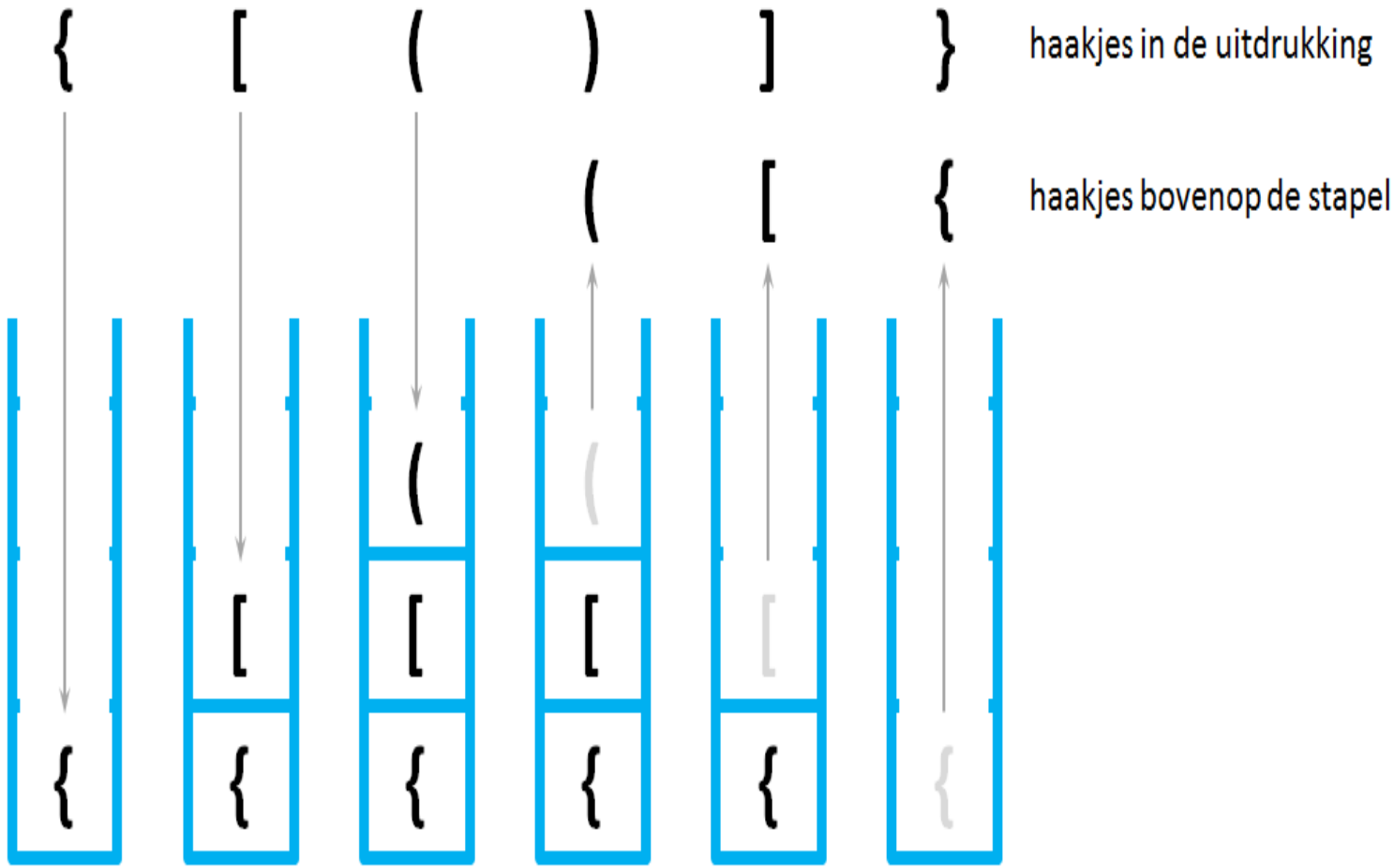
- Na área computacional existem diversas aplicações de pilhas.
- Alguns exemplos são: caminhamento em árvores, chamadas de sub-rotinas por um compilador ou pelo sistema operacional, inversão de uma lista, avaliar expressões, entre outras.
- Uma das aplicações clássicas é a conversão e a avaliação de expressões algébricas. Um exemplo, é o funcionamento das calculadoras da HP, que trabalham com expressões pós-fixadas.

Onde estamos ...

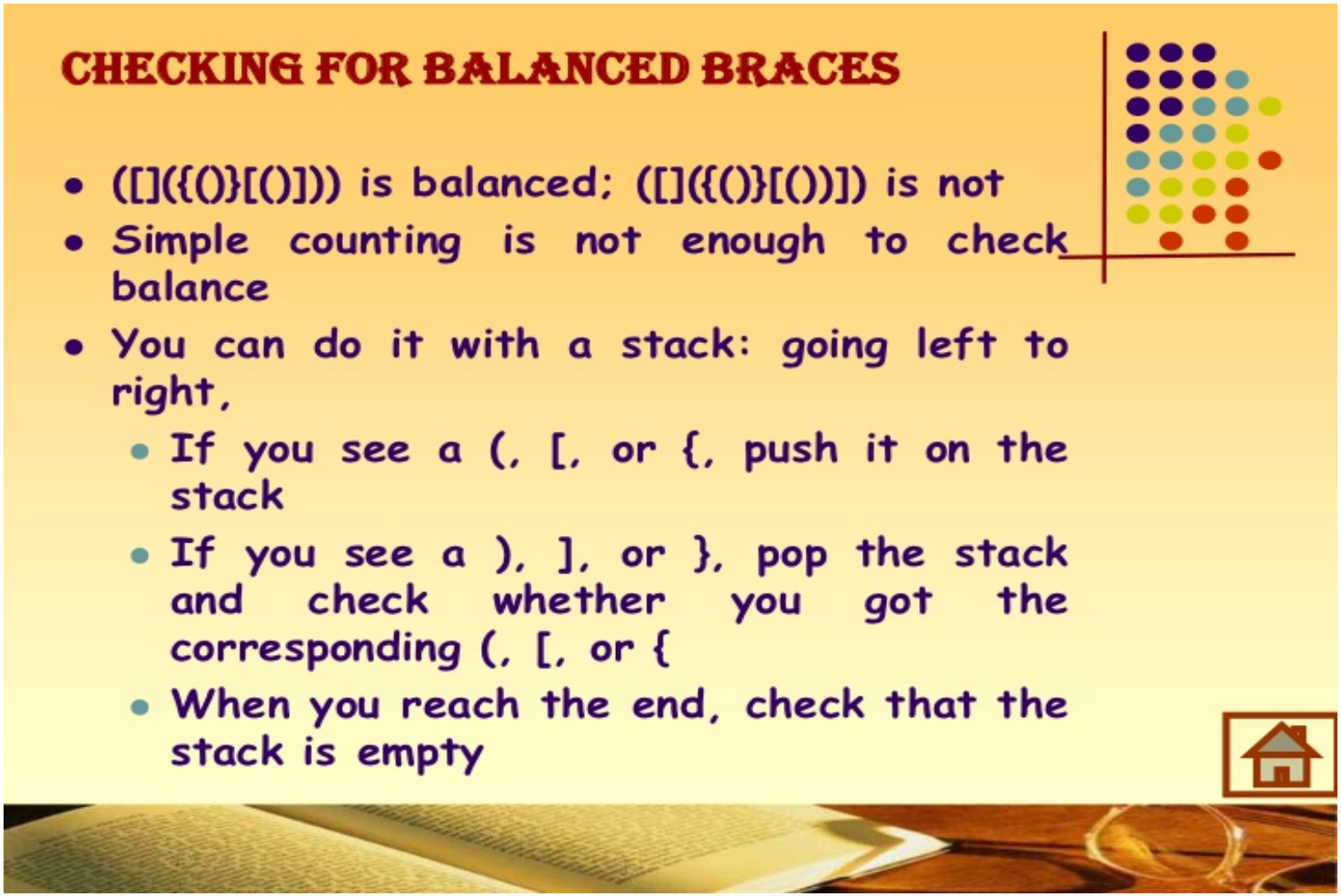
1. Pilha

2. Aplicações – Estudo de Casos

Balanceamento de Parenteses



Balanceamento de Parenteses



Uso imediato: validar expressões Infixas

[X / (Y-Z)+D]	
Symbol	**Stack**
[	[
X	[
/	[
(	[(
Y	[(
-	[(
Z	[(
)	[
+	[
D	[
]	Empty
Valid	

A* (B-C) }+D	
Symbol	**Stack**
A	Empty
*	Empty
(	(
B	(
-	(
C	(
)	Empty
}	Empty
Invalid	

Conversão In-fixa para Pós-fixa – versão 1

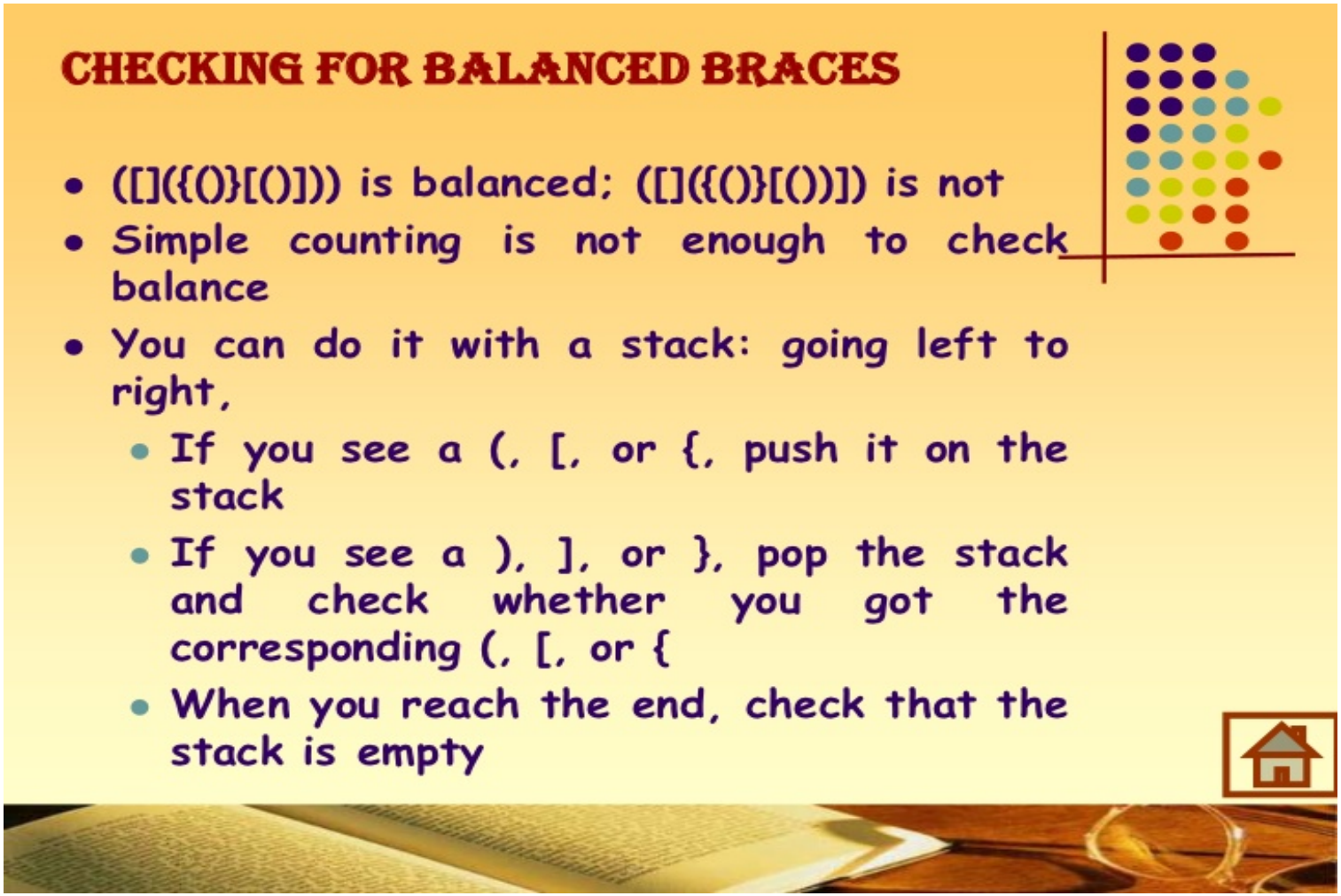
Infix to Postfix Conversion

Algorithm

1. Read the infix expression from left to right one character at a time.
2. Repeat step 3
3. Read symbol
   a. If symbol is operand put into postfix string
   b. If symbol is left parenthesis push into stack
   c. If right parenthesis, pop stack of TOP until left parenthesis occur and make postfix expression
   d. If operator then
      I. If operator has same of less precedence then operators available at top of stack, then pop all such operators and form postfix string
      II. Push scanned /incoming operator to the stack
4. Until last of string encountered
5. Pop all the element from stack to make stack empty

By Prof. Raj Sarode 17

Conversão In-fixa para Pós-fixa – versão 2



Validar Expressões Infixas

[X / (Y-Z)+D]	
Symbol	**Stack**
[	[
X	[
/	[
(	[(
Y	[(
-	[(
Z	[(
)	[
+	[
D	[
]	Empty
Valid	

A* (B-C) }+D	
Symbol	**Stack**
A	Empty
*	Empty
(	(
B	(
-	(
C	(
)	Empty
}	Empty
Invalid	

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
  symb = next input character;
  if (symb is an operand)
       add symb to the postfix string
  else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) )
        {
              topsymb = pop(opstk);
              add topsymb to the postfix string;
         } /* end while */

        push(opstk, symb);
    } /* end else */

} /* end while */

/* output any remaining operators */
while (!empty(opstk) ) {
      topsymb = pop(opstk);
      add topsymb to the postfix string;
} /* end while */
```

Example-2: (A+B)*C

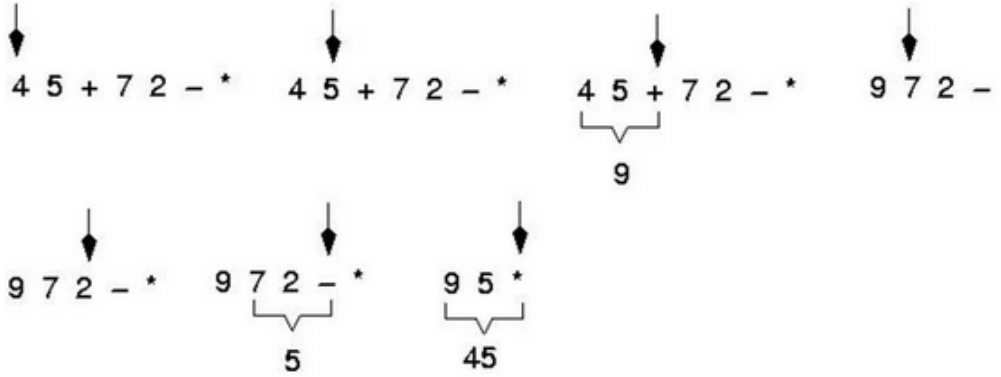
symb	Postfix string	opstk
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	
*	AB+	*
C	AB+C	*
	AB+C*	

Conversão Pré-fixa para Pós-fixa

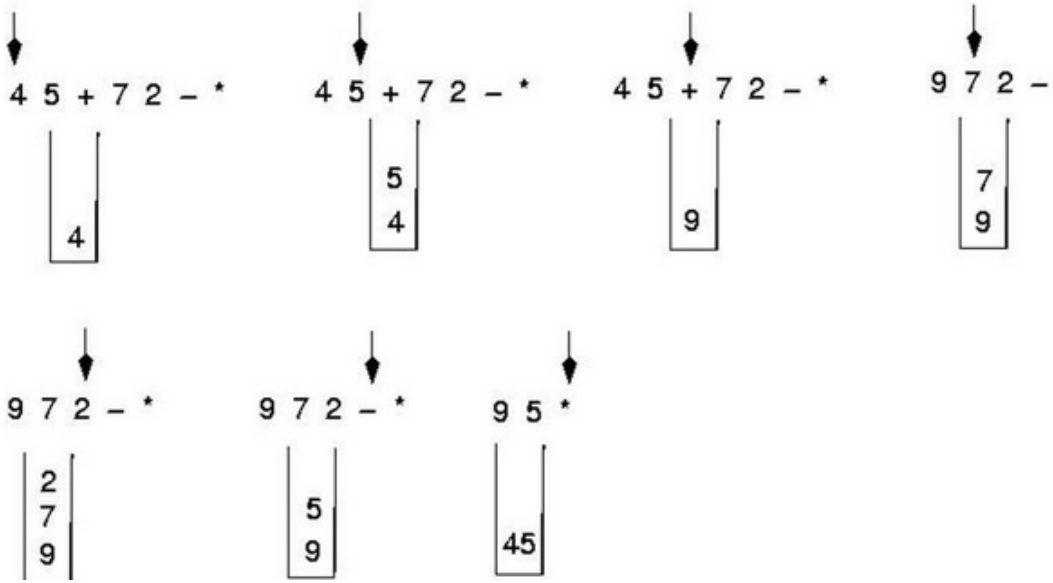
*a - (b + c * d)/e* to

ch	stack (bottom to top)	postfixExp
a		a
−	−	a
(	− (	a
b	− (	ab
+	− (+	ab
c	− (+	abc
*	− (+ *	abc
d	− (+ *	abcd
)	− (+	abcd*
	− (	abcd*+
	−	abcd*+
/	− /	abcd*+
e	− /	abcd*+e
		abcd*+e/−

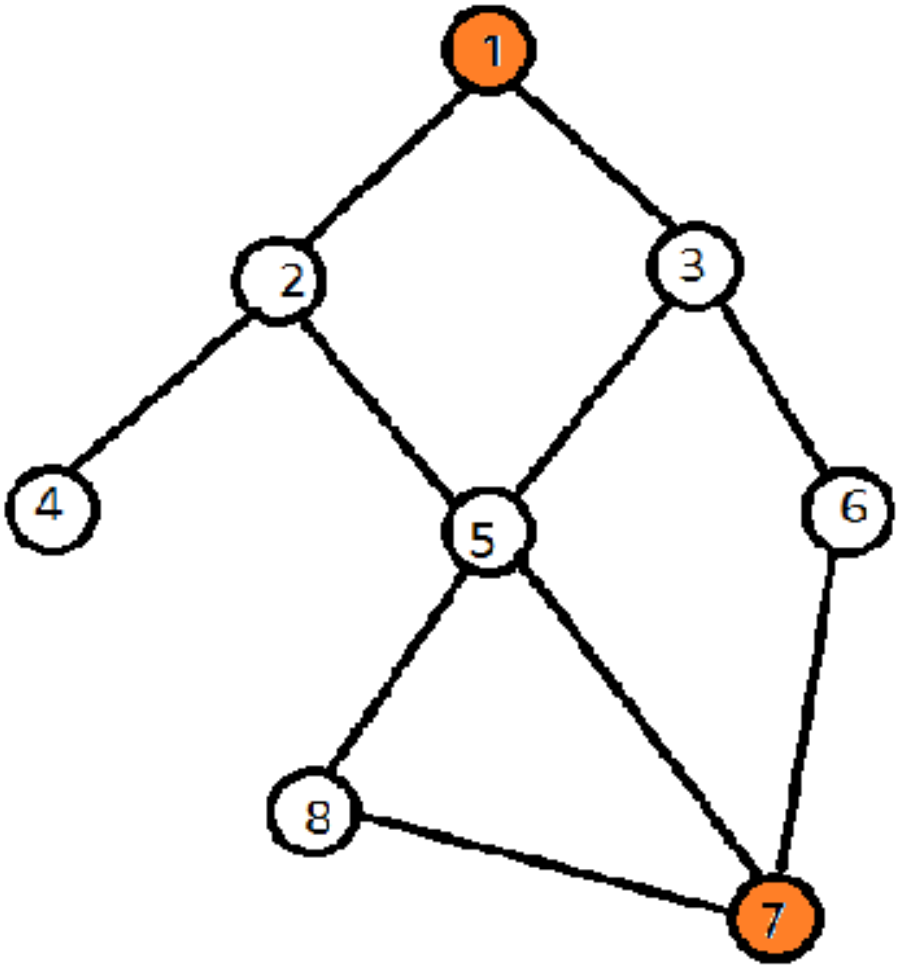
Uso: cálculo de expressões pós-fixas (e pré-fixas)



Picture below depicts *postfix expression using stack*



DFS: Busca em Profundidade



output :

1-2-5-8-7

1-2-5-7

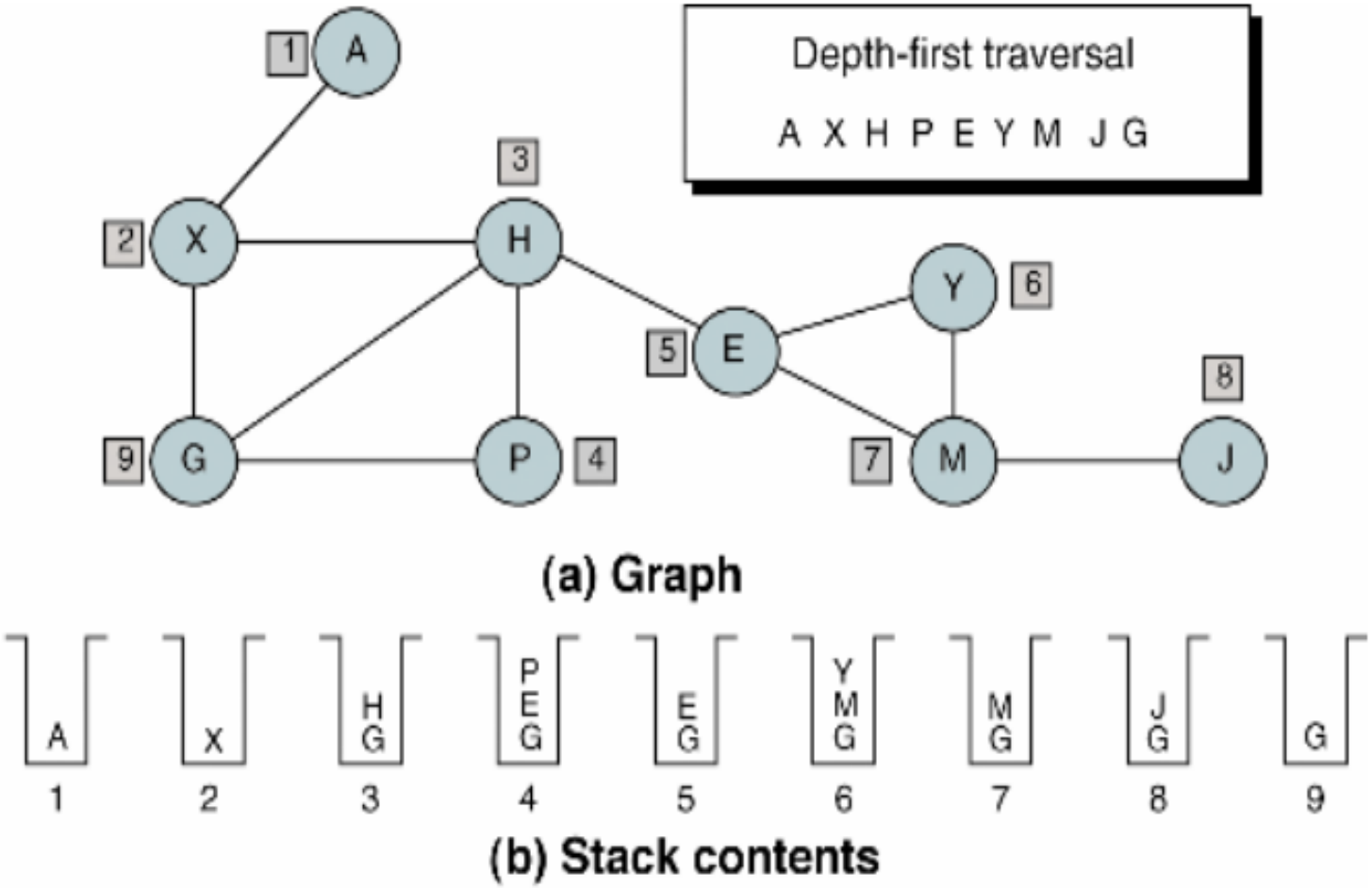
1-2-5-3-6-7

1-3-6-7

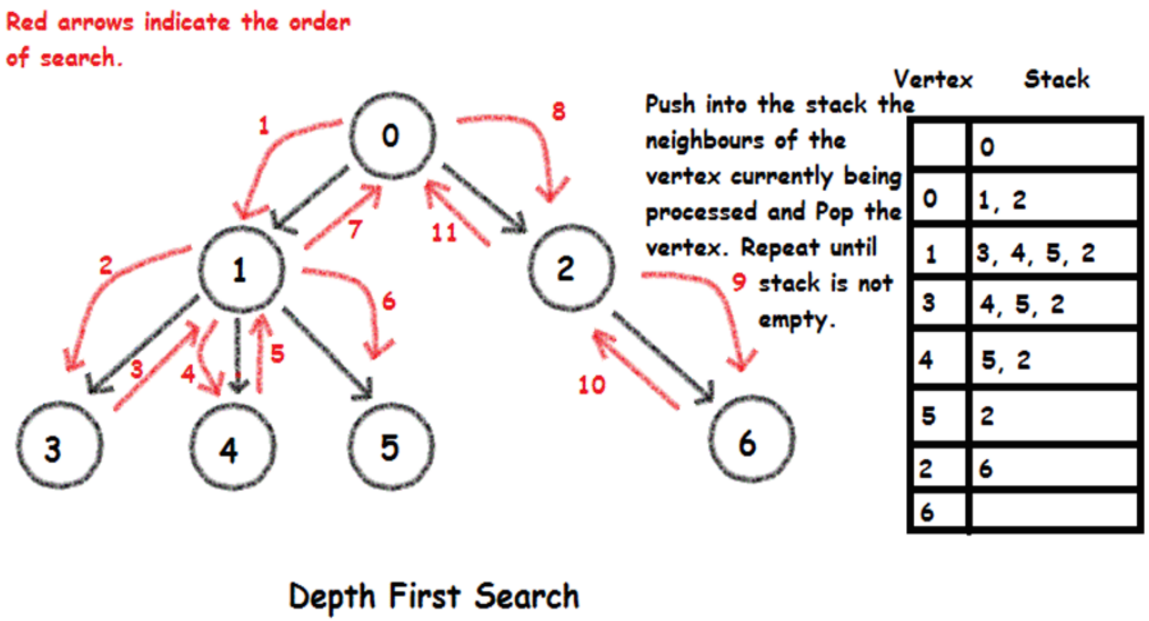
1-3-5-7

1-3-5-8-7

DFS: Busca em Profundidade

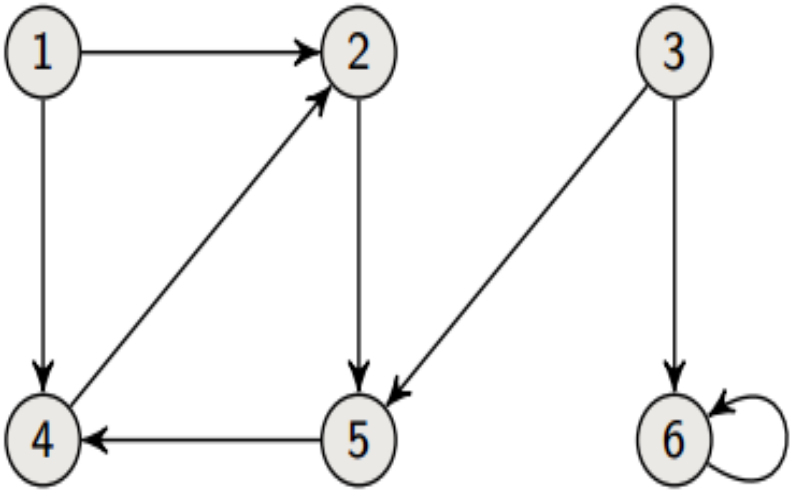


DFS: Busca em Profundidade



Representando Grafos: Matriz de Adjacência

Directed Graph



Adjacency matrix

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

O Algoritmo DFS Iterativo

Busca em Profundidade usando Estrutura de Pilha (P)

pseudo-código by CCS

Pré-requisitos:

- Leitura de uma matriz de adjacência G
- Uma pilha P e seus métodos
- Um vetor de status de n-vértices

```
varredura_do_grafo(G)
{
   vetor_status_nós[i] = aberto; //todos os i-nós marcados como não visitados
   no_inicial = lê_no_inicial( );
   DFS( no_inicial );
 }

DFS( vert )
{
   push( vert, P);                  // vertice = nó inicial da busca
   enquanto ( !vazia(P) )          // enquanto pilha não estiver vazia
    {
       x = pop (P);                 // obtém um vértice x  a ser inspecionado
       se (  vetor_status_nós[x] == aberto )
         {
             imprime_ou_visita(x);
             teste_o_que_quiseres(x); // AQUI se testa x ...
              vetor_status_nós[x] = fechado; // nó visitado, muda status
         };
        para_todos_vértices de i = x até x>= 0  faça
        // aqui muda o sentido da busca … esquerda ou direita
         {
            se (  (vetor_status_nós[i] == aberto ) AND
                     (vertice_vizinho[i] == 1)
              push(i,P);
         };  // fim do empilhamento dos nos adjacentes novos
     } ;  // fim do enquanto
 fim_do_DFS;
} ;
```

Ainda sobre o DFS

- Veloz – pouca memória
- Versátil: muitos tipos de aplicações
- Versão recursiva fica ainda mais simples
- Evita ciclos com os nós visitados
- Veja a implementação usando pilhas

Exercícios

1. Os exercícios propostos no início deste capítulo (slide inicial)
2. Escreva uma função que inverta a ordem das letras de cada palavra de uma sentença, preservando a ordem das palavras. Suponha que as palavras da sentença são separadas por espaços. A aplicação da operação à sentença **AMU MEGASNEM ATERCES**, por exemplo, deve produzir **UMA MENSAGEM SECRETA**.
3. Implemente uma função que receba uma pilha como parâmetro e retorne o valor armazenado em seu topo, restaurando o conteúdo da pilha. Essa função deve obedecer ao protótipo:

   ```
   char topo(Pilha* p);
   ```

4. Implemente uma função que receba duas pilhas, p_1, p_2, e passe todos os elementos da pilha p_2 para o topo da pilha p_1. Essa função deve obedecer ao protótipo:

   ```
   void concatena(Pilha* p1, Pilha* p2);
   ```