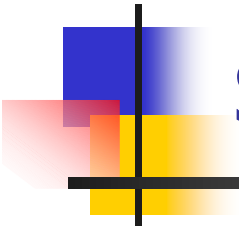


SIN211 Algoritmos e Estruturas de Dados

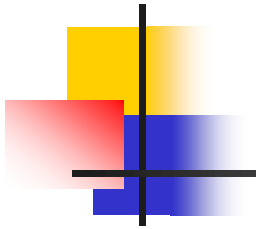
Prof. João Batista Ribeiro

joao42lbatista@gmail.com

UFV

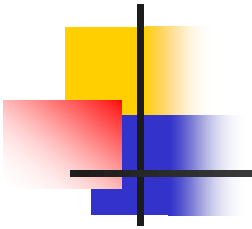
Universidade Federal de Viçosa

Slides baseados no material da Prof.ª Rachel Reis



Aula de Hoje

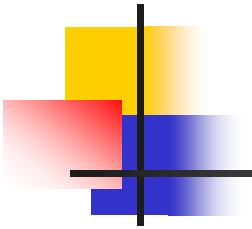
- TAD Pilha



Estrutura de Dados Lista

- Se considerarmos apenas as operações de inserção, remoção, realizadas sobre os extremos da lista, podemos falar em outras estruturas específicas, frequentes na modelagem de problemas computacionais.

- Exemplos dessas estruturas são as **pilhas**, **filas** e **deques**.



Pilha

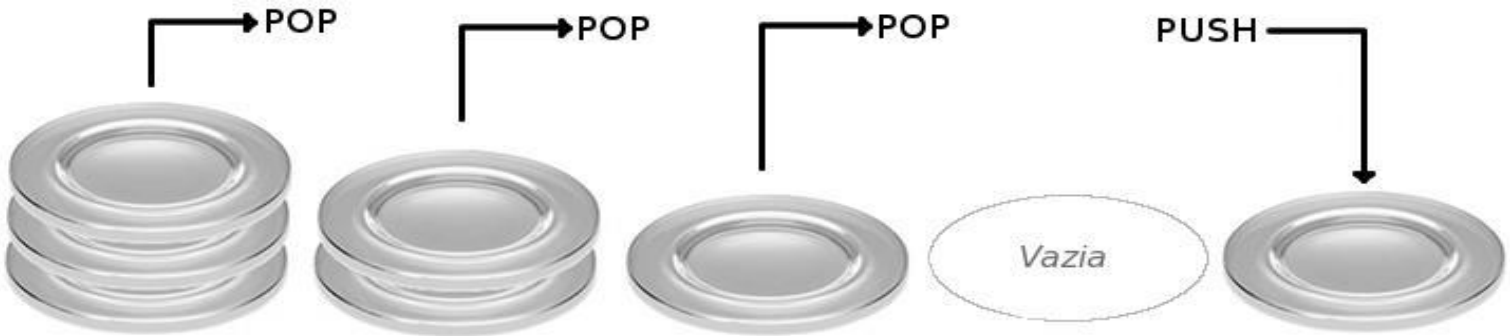
- Definições:

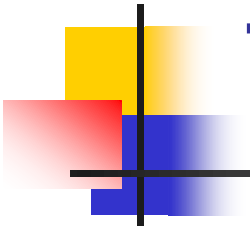
  "Uma **pilha** é um conjunto de itens no qual novos itens podem ser inseridos e a partir do qual podem ser eliminados em uma extremidade chamada **topo** da pilha" (Tenenbaum *et al*, 2005, p. 86).

  "Uma pilha é uma lista linear em que todas as inserções, retiradas e, geralmente, todos os acessos são feitos em apenas um extremo da lista" (Ziviani,2005, p. 72).

Pilha

- O tipo abstrato de dados pilha pode ser representado por uma pilha de pratos em um restaurante, se imaginarmos um prato por vez, os pratos são colocados e retirados apenas do topo da pilha. O último prato colocado será o primeiro a ser retirado da pilha.
- Pode-se pegar um prato apenas se existirem pratos na pilha, e um prato pode ser adicionado à pilha apenas se houver espaço suficiente, ou seja, se a pilha não estiver muito alta (implementação estática).



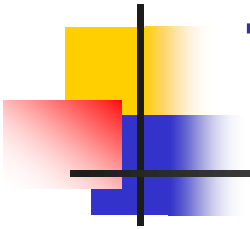


Tipo Abstrato de Dados Pilha

- As estruturas de dados **pilha** adotam o critério LIFO (*Last In First Out* - último elemento a entrar é o primeiro elemento a sair).

- Aplicação:

  1) Pilha de execução de um programa
  2) Casamento de delimitadores (parênteses, chaves, colchetes, etc)
  3) Notação posfixa



Tipo Abstrato de Dados Pilha

- Operações básicas:

  - inserir, também denominada **empilhar**:

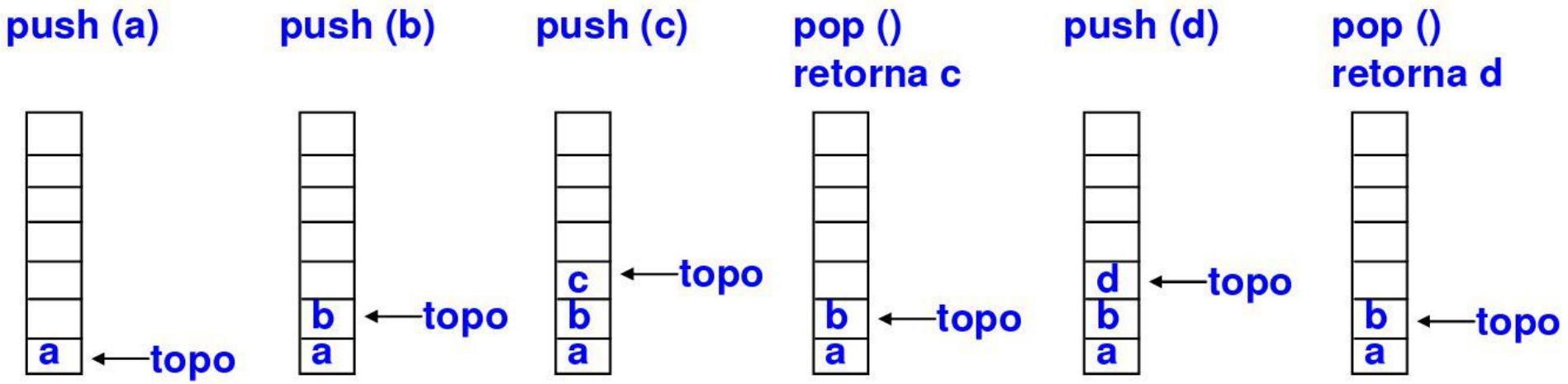
    push (s, x); /* se houver espaço, insere no topo da pilha 's' o elemento 'x'. */

  - remover, também denominada **desempilhar**:

    x = pop (s); /* se houver, remove o item que está no topo da pilha 's', retornando seu valor. */

Tipo Abstrato de Dados Pilha

- Operações push e pop:



Tipo Abstrato de Dados Pilha

- Outras operações:

  - pesquisar, ou acesso ao topo:

```
x = stacktop (s);    /* se houver, retorna o valor que está
                        no  topo da  pilha 's' */
```

  - inicializar:

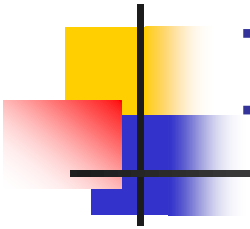
```
init (s);            // inicializa a pilha 's' no estado "vazia"
```

  - verificar se está vazia:

```
empty (s);           // verifica se a pilha 's' está "vazia"
```

  - verificar se está cheia:

```
full (s);            // verifica se a pilha 's' está "cheia"
                     (apenas para implementação estática)
```

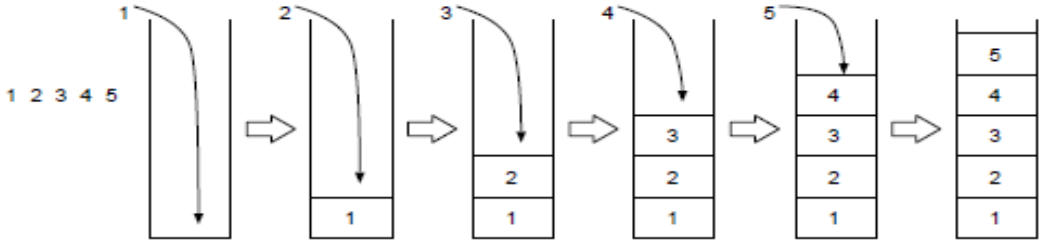


Implementação

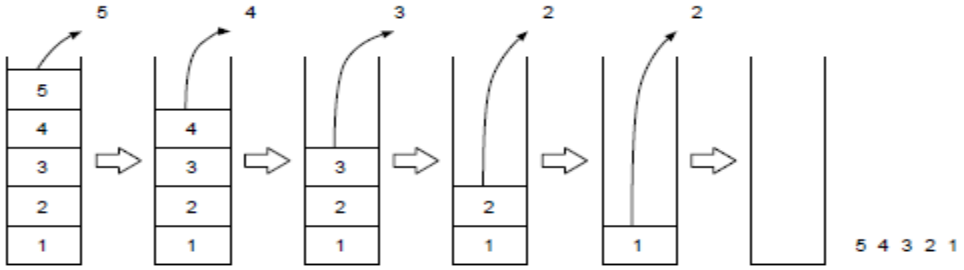
- Como uma pilha pode ser representada em C?
  - Implementação estática (vetor)
  - Implementação usando lista encadeada

Implementação - vetor

- Em uma implementação por meio de vetor os itens são armazenados em posições contíguas de memória
- A operação **push** faz a pilha expandir-se



- A operação **pop** faz a pilha contrair-se



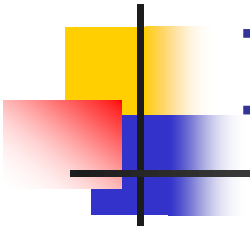
Implementação - vetor

- Definição do tipo Pilha

```
#define TAMMAX 100
typedef struct sPilha{
     <TIPO> itens[TAMMAX];
     int topo;
}Pilha;
```

- Declaração de uma pilha ‘p’

  Pilha p;



Implementação - vetor

- Operações:
  - Inicializa
  - Verificar se a pilha está vazia
  - Verificar se a pilha está cheia
  - Empilhar um item
  - Desempilhar
  - Retornar o elemento que está no topo da pilha sem removê-lo

Implementação - vetor

- Inicializar a pilha no estado “vazia”

```
void init (Pilha *s) {
    s->topo = -1;
}
```

Implementação - vetor

- Verificar se a pilha está vazia

```
int empty (Pilha *s){
    if(s->topo == -1)
        return 1;
    return 0;
}
```

Implementação - vetor

- Verificar se a pilha está cheia

```
int full(Pilha *s) {
    if (s->topo == TAMMAX-1)
        return 1;
    return 0;
}
```

Implementação - vetor

- Empilhar um item

```
void push(Pilha *s, int val) {
    if (full(s)) { // overflow
        printf("ERRO: pilha cheia.\n");
    }else {
        ++s->topo;
        s->itens[s->topo] = val;
    }
}
```

Implementação - vetor

- Desempilhar

```
int pop (Pilha *s) {
    int  aux;
    if ( empty(s)) {  //underflow
        printf ("ERRO: pilha vazia.\n") ;
        return -1;
    } else {
        aux = s->itens[s->topo];
        s->topo--;
        return aux;
    }
}
```

Implementação - vetor

- Retornar o elemento que está no topo da pilha sem removê-lo

```
int stacktop (Pilha *s){
    if(empty(s)){ //underflow
        printf ("ERRO: pilha vazia.\n");
        return -1;
    }
    return s->itens[s->topo];
}
```

Implementação - vetor

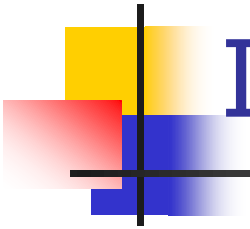
Programa Principal

```
int main(){
    int val;
    Pilha s;
    init(&s);

    push(&s, 7);
    push(&s, 4);

    pop(&s);

    val = stacktop(&s);
    if (val != -1)
        printf("Topo da pilha: %d", val);
}
```



Implementação – Lista Encadeada

- Para implementar uma pilha como uma lista encadeada, precisamos ter acesso ao topo da pilha
  - Para isso será necessário armazenar um ponteiro para o topo da pilha

Implementação – Lista Encadeada

- Um struct que representa um elemento da pilha, contendo as informações necessárias e um ponteiro para o próximo elemento

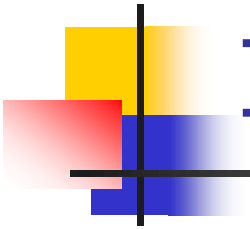
```
typedef struct cell{
   int info;
   struct cell * next;
}CELULA;
```

- Declaração de pilha:

```
CELULA* ptrPilha;
```

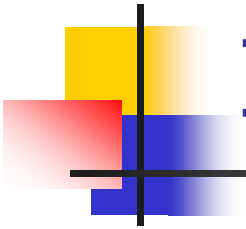
Implementação – Lista Encadeada

- Operações:
  - Inicializar
  - Verificar se a pilha está vazia
  - Acessar topo sem removê-lo
  - Empilhar
  - Desempilhar



Implementação – Lista Encadeada

- **Inicializar**
  - Nome da função: init
  - Tipo de retorno: void
  - Descrição: Inicializa o ponteiro para o topo da pilha com valor NULL
- **Verificar se a pilha está vazia**
  - Nome da função: empty
  - Tipo de retorno: int
  - Retornar 1 se a pilha estiver vazia; 0 caso contrário.



Implementação – Lista Encadeada

- **Empilhar**
  - Nome da função: push
  - Tipo de retorno: void
  - Descrição: função responsável por criar/inicializar/inserir um novo item no topo da pilha
- **Desempilhar**
  - Nome da função: pop
  - Tipo de retorno: <tipo_da_informação>
  - Descrição: função responsável por remover um item do topo da pilha.

Aplicação – Pilha de execução

1) Pilha de execução de um programa:

```
1 void proc1(){
2    printf("1");
3    proc2();
4 }
```

```
1 void proc2(){
2    proc3();
3    proc4();
4 }
```

```
1 void proc3(){
2    proc4();
3    printf("3");
4 }
```

```
1 void proc4(){
2    printf("4");
3 }
```

Pilha de execução

O que será impresso?

Aplicação – Pilha de execução

1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```

proc1()

Pilha de execução

O que será impresso?

Aplicação – Pilha de execução

1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```

O que será impresso?

Aplicação – Pilha de execução

1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```

O que será impresso?

Aplicação – Pilha de execução

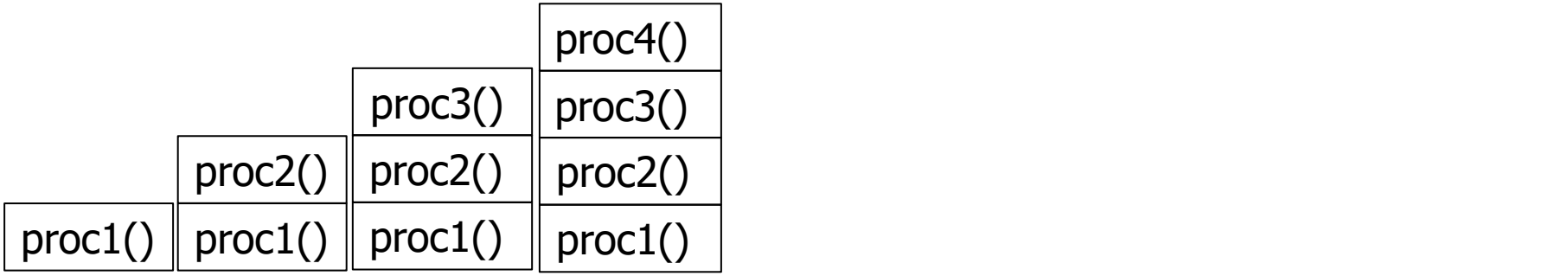
1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```



Pilha de execução

O que será impresso?

Aplicação – Pilha de execução

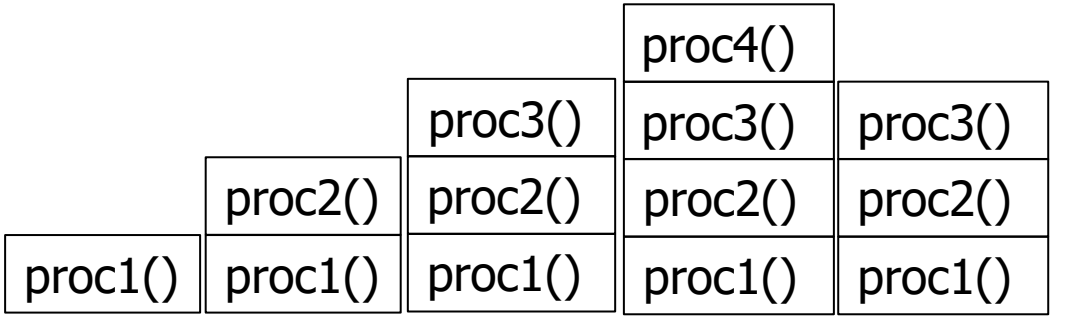
1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```



Pilha de execução

O que será impresso?

Aplicação – Pilha de execução

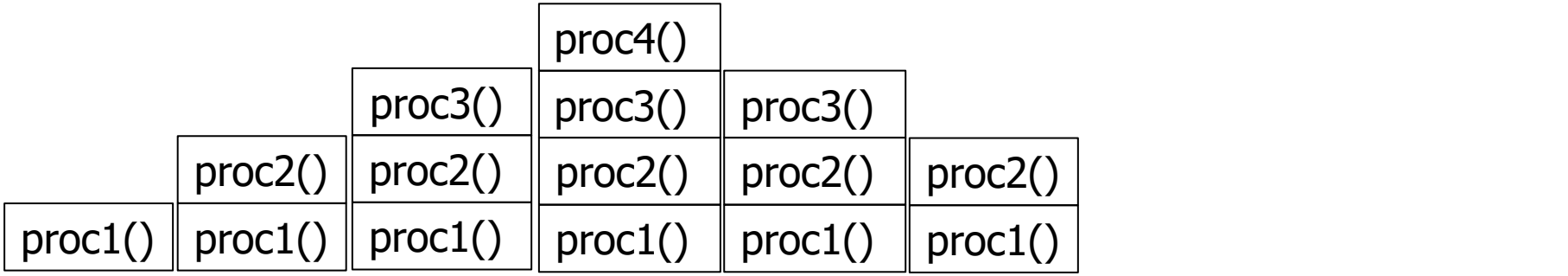
1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```



Pilha de execução

O que será impresso?

Aplicação – Pilha de execução

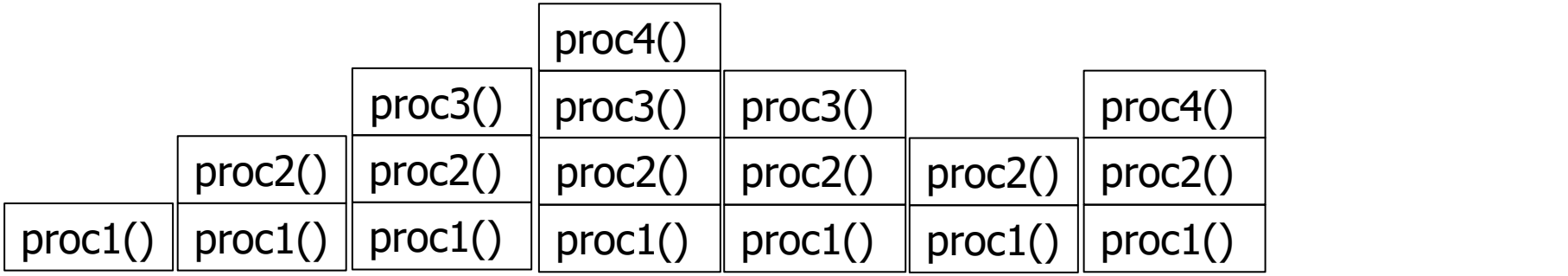
1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```



Pilha de execução

O que será impresso?

Aplicação – Pilha de execução

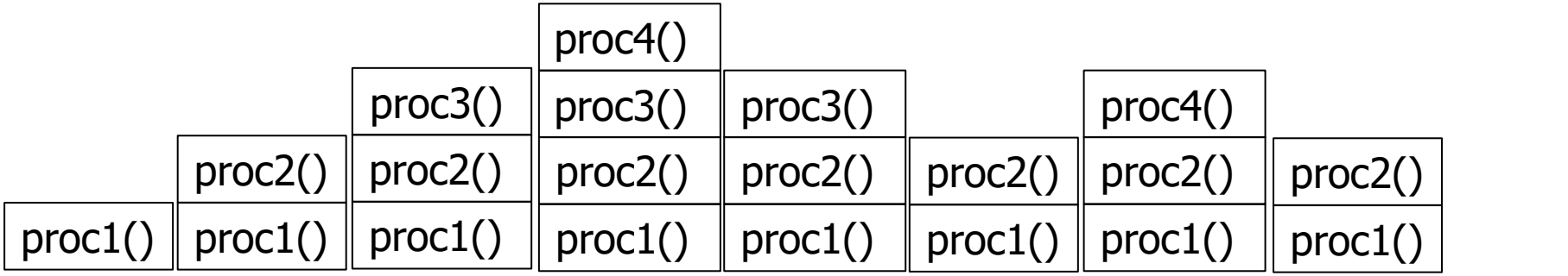
1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```



O que será impresso?

Aplicação – Pilha de execução

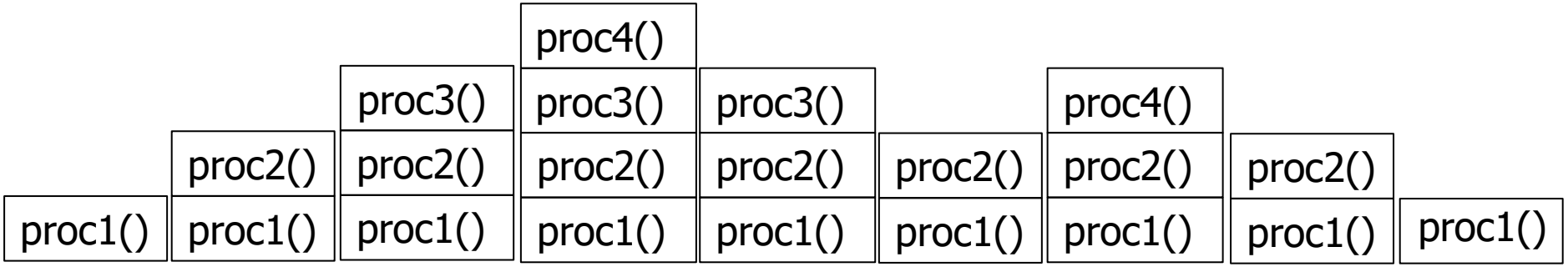
1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```



O que será impresso?

Aplicação – Pilha de execução

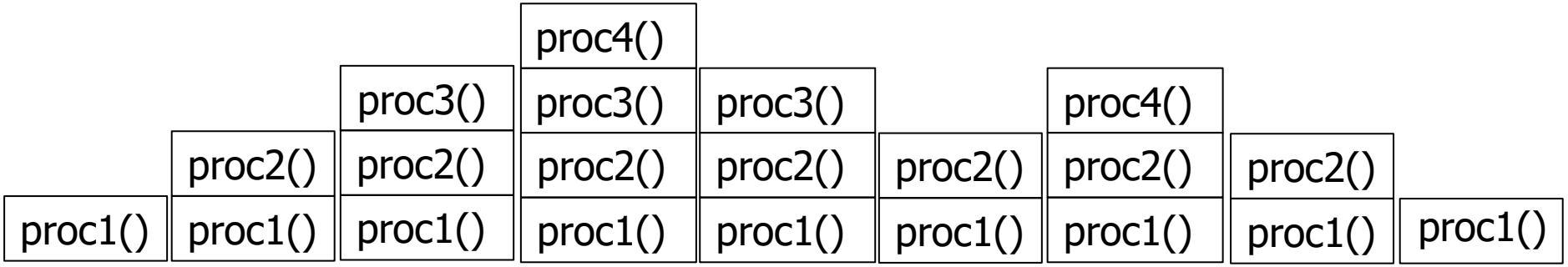
1) Pilha de execução de um programa:

```
1 void proc1(){
2   printf("1");
3   proc2();
4 }
```

```
1 void proc2(){
2   proc3();
3   proc4();
4 }
```

```
1 void proc3(){
2   proc4();
3   printf("3");
4 }
```

```
1 void proc4(){
2   printf("4");
3 }
```

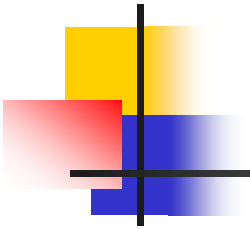


O que será impresso?

Implementação – Lista Encadeada

- **Acessar topo sem removê-lo**
  - Nome da função: stacktop
  - Tipo de retorno: <tipo_da_informação>
  - Descrição: função responsável por retornar o conteúdo do item que está no topo da pilha sem removê-lo.

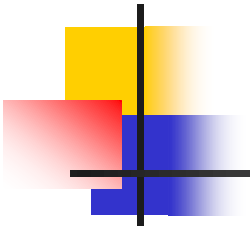
- **Exibir elementos da pilha**
  - Tipo de retorno: void
  - Descrição: função responsável por imprimir o conteúdo do item que está no topo da pilha e em seguida removê-lo. Essa operação deverá ser repetida até que a pilha fique vazia.



Aplicação – Expressões Aritméticas

2) Avaliação de expressões aritméticas

- *Delimiter Matching* (casamento de delimitadores)
- Nos programas em C temos os seguintes delimitadores:
  - parênteses '(' e ')‘
  - colchetes '[' e ']‘
  - chaves '{' e '}’
  - comentários '/*' e '*/'



Aplicação – Expressões Aritméticas

- Exemplo de declarações em C com o uso **NÃO** apropriado de delimitadores:

```
/* Comentário de uma linha de código */
a = a + b (c - d) * (e - f));
g[10] = h[i[9]] + (j + k * l;
while (m < (n[0) + o))
{
  p = 7;
  r = 6;
```

Aplicação – Expressões Aritméticas

- Em algumas declarações há delimitadores aninhados
- Assim, um delimitador só poderá ser casado depois que todos os delimitadores que o seguem também sejam casados.
- Algoritmo para casamento de delimitadores:

→ Lê um caractere a partir de um código fonte

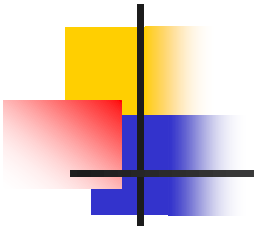
→ Se for um delimitador de abertura, armazena-o numa pilha

→ Se um delimitador de fechamento for encontrado, é comparado com o delimitador no topo da pilha

- se há casamento, o processo continua,
- senão há um erro e o processo pára.

Aplicação – Expressões Aritméticas

```
ler caractere ch do arquivo
enquanto (ch != ';') ou (nao e fim de arquivo) {
    se ch == '(' ou '[' ou '{'
       push(ch)
    senao
       se ch == ')' ou ']' ou '}'
          se ch e pop() nao se casam, erro!
    senao
       se (ch == '/' e o proximo caractere == '*') {
          pule todos os caracteres ate encontrar o '*/'
          erro se nao encontrar '*/' antes de fim de arquivo
       }
    ler proximo caractere ch a partir de arquivo
}
se a pilha é vazia, sucesso
senao, erro
```



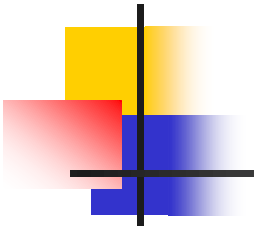
Aplicação - Notação pós-fixa

→ Notação para expressões aritméticas

- infixa = operador entre os operandos (1-2)*(4+5)
- posfixa = operador após operandos 1 2 – 4 5 + *
- prefixa = operador antes dos operandos * - 1 2 + 4 5

→Exemplo:

- calculadora HP científica usa notação posfixa



Aplicação - Notação posfixa

→Avaliação de expressões aritméticas posfixadas:

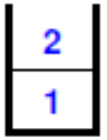
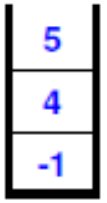
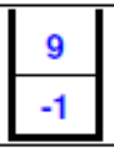
1) Cada operando é empilhado numa pilha de valores

2) Quando se encontra um operador

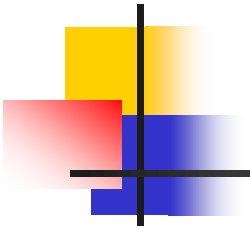
- Desempilha-se o número apropriado de operandos; (dois para operadores binários e um para operadores unários)
- Realiza-se a operação devida;
- Empilha-se o resultado.

→ Exemplo:

- Avaliação da expressão 1 2 – 4 5 + *

empilhe os valores 1 e 2	1 2 – 4 5 + *	
quando aparece o operador "–"	1 2 – 4 5 + *	
desempilhe 1 e 2		
empilhe -1, o resultado da operação (1 - 2)		
empilhe os valores 4 e 5	1 2 – 4 5 + *	
quando aparece o operador "+"	1 2 – 4 5 + *	
desempilhe 4 e 5		
empilhe 9, o resultado da operação (4+5)		
quando aparece o operador "*"	1 2 – 4 5 + *	
desempilhe -1 e 9		
empilhe -9, o resultado da operação (-1*9)		

Fonte: http://www.inf.puc-rio.br/~inf1620/material/slides/capitulo11.PDF



Leituras Recomendadas

- DROZDEK, Adam. Estrutura de Dados e Algoritmos em C++. Editora Pioneira Thomson Learning, 2005.

→Pág. 123 (Pilha)

- TENENBAUM A., LANGSAM Y. e AUGENSTEIN M. J. Estrutura de Dados usando C. Editora Makron, 1995.

→Pág. 86 (Pilha)

- FEOFILOFF, Paulo. Algoritmos em Linguagem C. Editora Campus, 2009.

→Pág. 39 (Pilha)