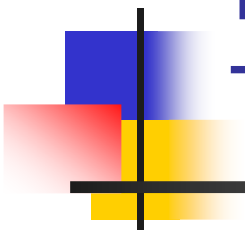
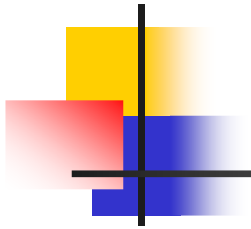


Estruturas de Dados

TAD: Pilha Estática

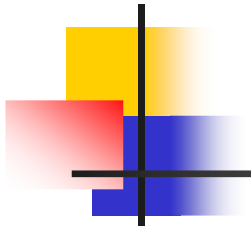
Prof. Marcelo Zorzan

Profª. Melissa Zanatta



Aula de Hoje

- TAD Pilha Estática



Pilha

- Estrutura linear de dados que pode ser acessada somente por uma de suas extremidades para armazenar e recuperar dados.
- Característica: "O último elemento a ser inserido é o primeiro a ser retirado/ removido" (LIFO – *Last in First Out*)
- Analogia: pilha de pratos
- Aplicações:
  - Pilha de execução de um programa/ chamada recursiva de funções
  - Avaliação de expressões aritméticas (casamento de delimitadores)
  - Notação pós-fixa

Aplicação – Pilha de execução

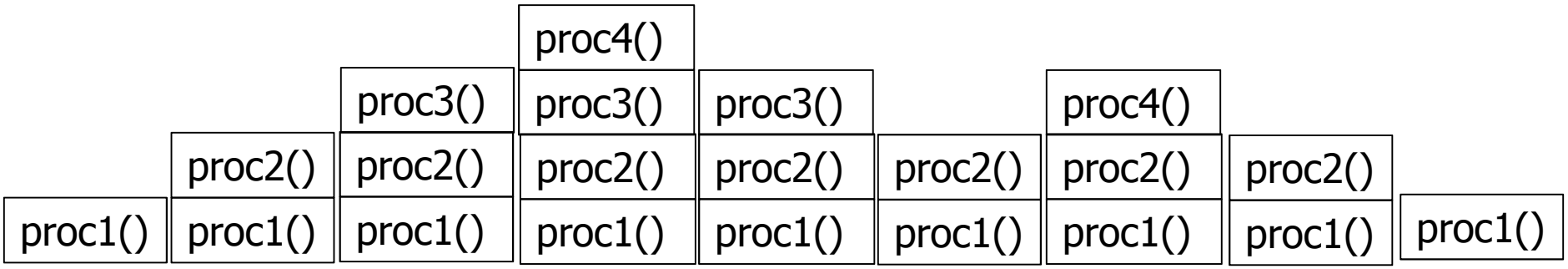
1) Pilha de execução de um programa:

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

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

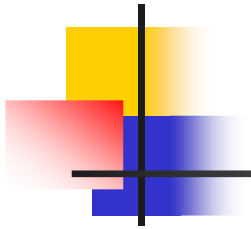
```
1 void proc3 () {
2   proc4();
3   printf ("3");
4 }
```

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



Pilha de execução

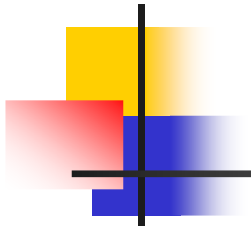
O que será impresso?



Aplicação – Expressões Aritméticas

2) Avaliação de expressões aritméticas

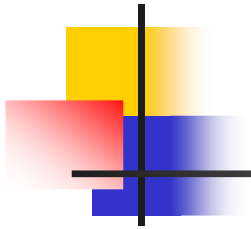
- *Delimiter Matching* (casamento de delimitadores)
- Nos programas em C temos os seguintes delimitadores:
  - parênteses '(' e ')‘
  - colchetes '[' e ']‘
  - chaves '{' e '}’
  - comentários '/*' e '*/'



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

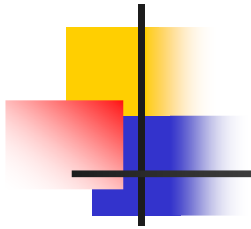
→ Se for um delimitador de abertura, armazena-o numa pilha

→ 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

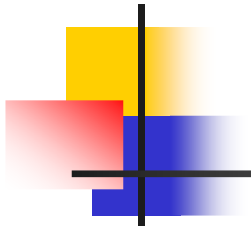
– infixa = operador entre os operandos (1-2)*(4+5)

– pós-fixa = operador após operandos 1 2 – 4 5 + *

– pré-fixa = operador antes dos operandos * - 1 2 + 4 5

→Exemplo:

– calculadora HP científica usa notação pós-fixa



Aplicação - Notação pós-fixa

→Avaliação de expressões aritméticas pós-fixadas:

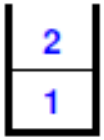
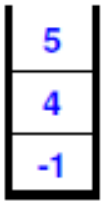
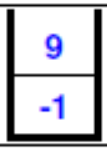
1) Cada operando é empilhado numa pilha de valores

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

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

- Outras operações:

  - pesquisar, ou acesso ao topo:

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

  - inicializar:

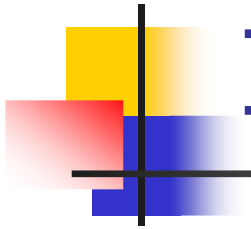
    init (s); // inicializa a pilha ‘s’ no estado “vazia”

  - 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” (implementação estática)

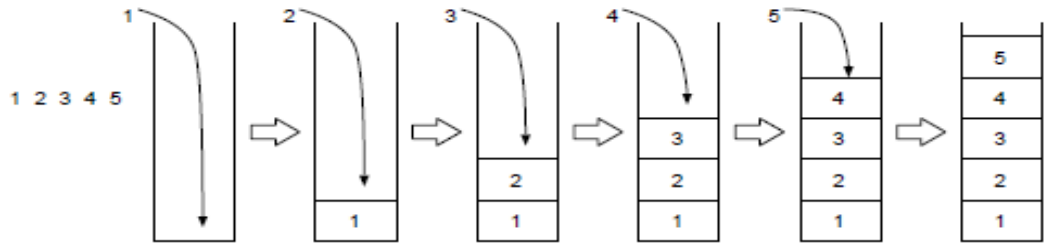


Implementação

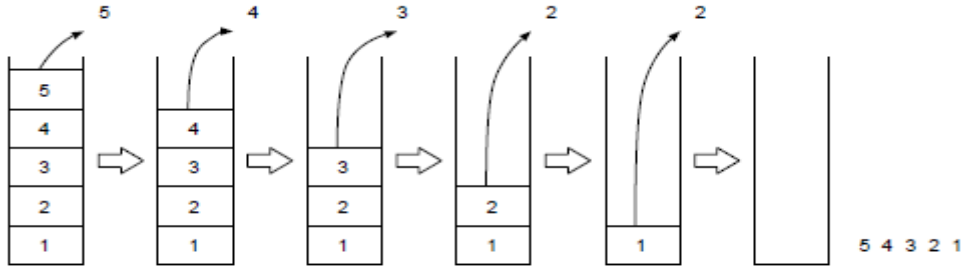
- Como uma pilha pode ser representada em C?
  - Implementação estática (*array)*
  - Implementação usando lista encadeada

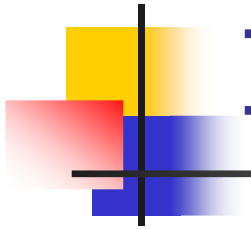
Implementação - *array*

- Em uma implementação por meio de *array* 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





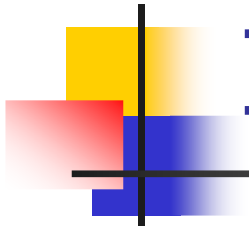
Implementação - *array*

- Definição do tipo Pilha

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

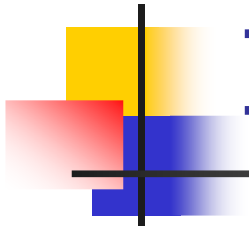
- Declaração de uma pilha ‘s’

  Pilha s;



Implementação - *array*

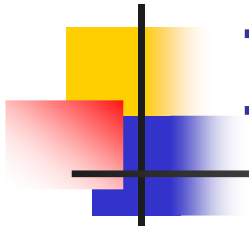
- Operações:
  - Inicializa
  - Verificar se a pilha está vazia
  - Verificar se a pilha está cheia
  - Retornar o elemento que está no topo da pilha sem removê-lo



Implementação - *array*

- Inicializar a pilha no estado “vazia”

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



Implementação - *array*

- Verificar se a pilha está vazia

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

Implementação - *array*

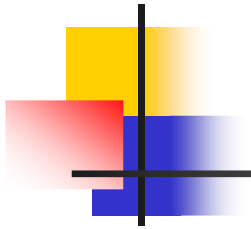
- Verificar se a pilha está cheia

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

Implementação - *array*

- Empilhar um item

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



Implementação - *array*

- Desempilhar

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

Implementação - *array*

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

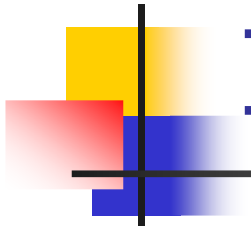
```
int stacktop (Pilha *s) {
    if (empty(s)) {   //underflow
        printf ("ERRO: pilha vazia.\n");
        return -1;
    }
    return s->itens[s->topo];
}
```

Implementação - *array*

Programa Principal

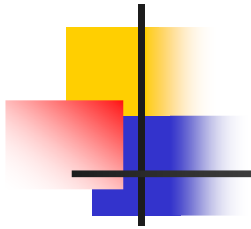
```
int main()
{
    Pilha s;
    inicializa(&s);

    push(&s, 7);
    push(&s, 4);
    pop(&s);
}
```



Implementação - *array*

- Implemente as operações abaixo considerando uma pilha de números inteiros:
  - Inicializar
  - Verificar se a pilha está vazia
  - Acessar topo sem removê-lo
  - Empilhar
  - Desempilhar



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)