



# MAC122 – Princípios de Desenvolvimento de Algoritmos

DCC - IME - USP

Prof.: Dr. Paulo Miranda

[pmiranda@vision.ime.usp.br](mailto:pmiranda@vision.ime.usp.br)

## Lista 02: Pilhas & Filas

Para as questões envolvendo pilhas, assuma a existência das funções:

```
Pilha    CriaPilha();  
void     LiberaPilha(Pilha p);  
bool     PilhaVazia(Pilha p);  
void     Empilha(Pilha p, TipoDado x);  
TipoDado Desempilha(Pilha p);
```

Assuma também a definição:

```
typedef enum boolean {false,true} bool;
```

Você deve definir **TipoDado** usando **typedef** do modo que considerar melhor.

### Questão 1:

Faça uma função que use uma pilha para inverter a ordem das letras de cada palavra de uma string, preservando a ordem das palavras. Por exemplo, dado o texto:

“EXERCICIO MUITO COMPLICADO”

A saída deve ser: “OICICREXE OTIUM ODACILPMOC”.

### Questão 2:

Diremos que uma expressão posfixa simples é uma string cujos caracteres pertencem ao conjunto '+', '\*', '0', '1', ..., '9' (o caractere ' ' não faz parte do conjunto). Uma expressão posfixa simples representa uma expressão aritmética de modo alternativo, em que os argumentos precedem o símbolo da operação. Por exemplo, "123\*+" representa a expressão  $1 + (2 * 3)$ , cujo valor é 7. Note que cada caractere numérico representa um número entre 0 e 9; assim, a sequência "48" representa o número 4 seguido do número 8, e não o número 48. Escreva uma função que receba uma expressão posfixa simples e devolva o valor da correspondente expressão aritmética.

```
int ValorExpressao(char posfixa[]);
```

Exemplo de chamada da função:

```
int main() {  
  
    char pos[] = "123+*45+*6*";  
    int v;  
  
    v = ValorExpressao(pos);  
    printf("v: %d\n", v); /* Imprime v: 270 */  
  
    return 0;  
}
```

### **Questão 3:**

#### **Item A:**

Faça uma função em C que recebe em `cadeia` uma string não balanceada de parênteses e colchetes, e que gera em `corr` uma versão corrigida balanceada, através da inserção de parênteses e colchetes ausentes. O número de caracteres adicionais inseridos deve ser retornado pela função.

```
int CorrigeBalanceamentoFA(char *cadeia, char *corr);
```

Durante o processamento da string `cadeia` (da esquerda para a direita), sempre que for detectado uma falha de balanceamento, a função deve sempre resolver o conflito imediatamente através da inserção de um caractere adicional. Nessa primeira versão, considere que a função sempre dá preferência para inserir um caractere de fechamento para a resolução de conflitos, sendo que caracteres de abertura adicionais só devem ser usados se não houver outra opção imediata.

Dica: Use como base o código visto em aula que verifica o balanceamento de parênteses e mostre quais modificações são necessárias para a resolução do novo problema.

#### Exemplos:

Para a `cadeia` = "[ ( ] )" a saída esperada na string `corr` é "[ ( ) ] ( )", e o valor de retorno da função é 2, dado que dois novos caracteres foram inseridos durante a correção (ver setas).

Para "[ ( ] ]" a saída em `corr` é "[ ( ) ] ( ) [ ]" e o valor de retorno é 3.

Para "[ ( ]" a saída em `corr` é "[ ( ) ]" e o valor de retorno da função é 1.

**OBS:** Para a resolução dessa questão, além das funções de pilhas, considere também as seguintes funções auxiliares:

```
char ParDeAbertura(char c){
    if(c=='[') return '[';
    else if(c=='(') return '(';
}

char ParDeFechamento(char c){
    if(c==']') return ']';
    else if(c==')') return ')';
}
```

#### **Item B:**

Considere agora uma segunda versão da função que sempre dá preferência para inserir um caractere de abertura para a resolução de conflitos, sendo que caracteres de fechamento adicionais só devem ser usados se não houver outra opção imediata.

```
int CorrigeBalanceamentoAF(char *cadeia, char *corr);
```

#### Exemplos:

Para "[ ( ] )" a saída em `corr` é "[ ( [ ] ) ]" e o valor de retorno é 2.

Para "[ ( ] ]" a saída em `corr` é "[ ( [ ] ) ]" e o valor de retorno é 1.

Para "[ ( ]" a saída em `corr` é "[ ( [ ] ) ]" e o valor de retorno é 3.

#### **Questão 4:**

Na notação pré-fixa o operador precede os dois operandos. Exemplos:

<b>infixa</b>	<b>pré-fixa</b>
a+b	+ab
a+b*c	+a*bc
(a+b)*c	*+abc

Considerando os operadores '+', '-', '\*' e '/', faça uma função usando pilhas que converte uma expressão em notação infix para pré-fixa. No caso de operadores de mesmo nível de prioridade, considere a associação à esquerda.

**OBS:** Assim como na notação pós-fixa, a notação pré-fixa também não necessita de parênteses. A própria forma da expressão indica a ordem das operações.

#### **Questão 5:**

(Questão extraída da apostila "Estruturas de Dados e Técnicas de Programação" (2004) - Cláudio L. Lucchesi & Tomasz Kowaltowski)

A notação pós-fixa pode ser estendida para operadores monádicos (de um operando), entretanto eles introduzem ambiguidade. Por exemplo, as expressões infixas distintas " $a - (-b)$ " e " $-(a - b)$ " seriam traduzidas para a mesma expressão pós-fixa " $ab--$ ". A fim de evitar este problema, os operadores devem utilizar símbolos distintos na tradução. Adotando-se o símbolo  $\sim$  para o operador unário  $-$ , teríamos:

$a - (-b)$   $\rightarrow$   $ab\sim\sim$

$-(a - b)$   $\rightarrow$   $ab--\sim$

Complete a implementação da função de conversão de notação infix para pós-fixa usando pilhas vista em aula para incluir o operador monádico '-'.

#### **Questão 6:**

(Questão adaptada da apostila "Estruturas de Dados e Técnicas de Programação" (2004) - Cláudio L. Lucchesi & Tomasz Kowaltowski)

Em algumas aplicações é interessante utilizar o conceito de fila dupla. Nesta estrutura, as inserções e remoções podem ser feitas em qualquer uma das duas extremidades da sequência. Proponha implementações ligada e sequencial para esta estrutura e escreva as rotinas básicas para manipulá-la:

```
FilaDupla CriaFilaDupla();
```

```
void LiberaFilaDupla(FilaDupla p);
```

```
bool FilaDuplaVazia(FilaDupla p);
```

```
void InsereFrenteFilaDupla(FilaDupla *p, TipoDado x);
```

```
void InsereFimFilaDupla(FilaDupla *p, TipoDado x);
```

```
TipoDado RemoveFrenteFilaDupla(FilaDupla *p);
```

```
TipoDado RemoveFimFilaDupla(FilaDupla *p);
```

### Questão 7:

Dada a estrutura a seguir que representa uma imagem com  $m$  linhas e  $n$  colunas em tons de cinza (cores do preto ao branco como valores inteiros entre 0 e 255), em que o apontador  $M$  corresponde a uma matriz  $m \times n$  alocada dinamicamente,

```
typedef struct _Imagem {
    int **M;
    int m;
    int n;
} Imagem;
```

faça uma função em C que troque a cor de uma área contínua de mesma cor para uma nova cor fornecida, usando uma pilha auxiliar. A função em questão recebe como entrada um ponto definido por índices  $(x,y)$  no interior da região a ser pintada e a cor de preenchimento *novaCor*. O algoritmo inicialmente detecta a cor *corAnterior* no ponto  $(x,y)$  e começa pintando tal posição com a cor *novaCor* e faz sucessivamente o mesmo para os vizinhos de cima, de baixo, da esquerda e da direita caso estejam dentro da imagem e possuam cor igual a *corAnterior*, isto é, igual à cor a ser substituída.

Observe o exemplo abaixo:

Imagem de entrada

		→ x							
		1	1	3	3	3	255	255	255
		1	1	3	3	3	255	1	1
		1	255	3	3	3	1	1	1
	↓ y	3	255	3	3	3	1	1	1
		3	255	1	3	3	3	1	1
		1	255	1	3	3	1	3	3
		1	1	1	3	1	1	3	3

Após a chamada da função para essa imagem, com  $x = 2$ ,  $y = 3$  (que corresponde ao ponto destacado na imagem de entrada) e *novaCor* = 255 o resultado é:

Imagem Resultante

		1	1	255	255	255	255	255	255
		1	1	255	255	255	255	1	1
		1	255	255	255	255	1	1	1
		3	255	255	255	255	1	1	1
		3	255	1	255	255	255	1	1
		1	255	1	255	255	1	3	3
		1	1	1	255	1	1	3	3

O protótipo da função a ser construída é:

```
void trocaCor (Imagem *img, int x, int y, int novaCor);
```

#### Observações:

- Essa ferramenta é conhecida em programas de edição de imagem como "lata de tinta".
- Esse problema pode ser resolvido usando o algoritmo de preenchimento por inundação (popularmente conhecido pelo termo inglês floodfill).