

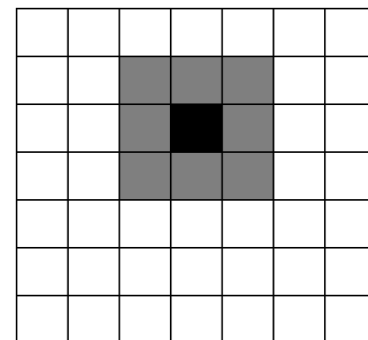
Algoritmos e Estruturas de Dados I

Prof. Lúcio Mauro Pereira

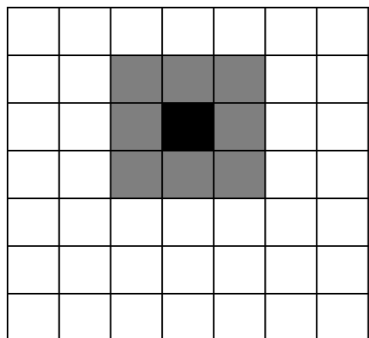
11/11/2024

Algumas questões em laboratório

Elementos vizinhos em uma matriz



Construa uma função que receba uma matriz de reais e dois valores inteiros relativos à posição de um elemento da matriz: linha e coluna. A função deverá gerar a média dos valores dele e seus vizinhos, como apresentado abaixo.



Teste sua solução, a partir da função principal, lembrando de validar potenciais posições extremas da matriz, como ponto chave de pesquisa posicionado na primeira ou última linha, na primeira ou última coluna.

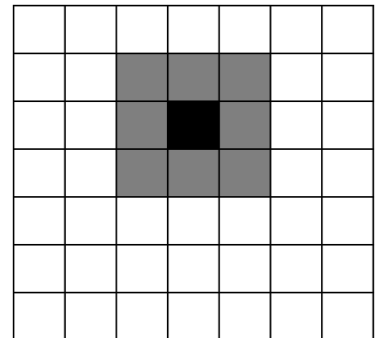
Identificadores globais

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define MAX_LIN 7
```

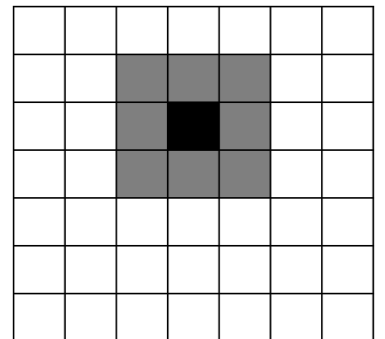
```
#define MAX_COL 7
```



Media quadrado

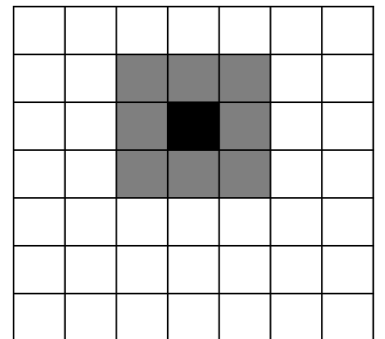
```
float mediaQuadrado(float M[][MAX_COL], int linha, int coluna){
```

```
} //fim mediaQuadrado()
```



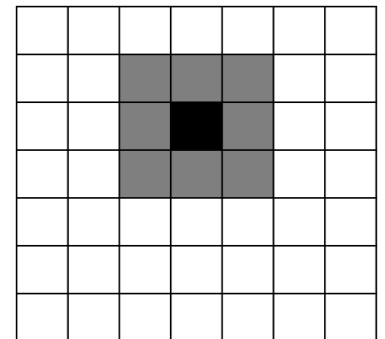
Media quadrado

```
float mediaQuadrado(float M[][MAX_COL], int linha, int coluna){  
    float soma=0;  
    int     c=0;  
  
    return soma/c;  
} //fim mediaQuadrado()
```



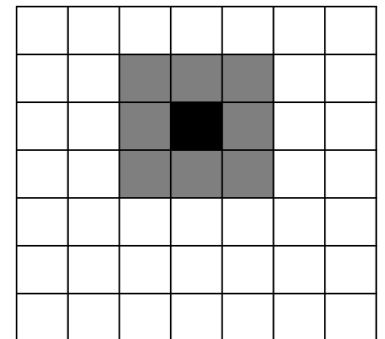
Media quadrado

```
float mediaQuadrado(float M[][MAX_COL], int linha, int coluna){  
    float soma=0;  
    int    c=0;  
    for(int i=linha-1 ; i<=linha+1 ; i++){ //varrendo as linhas  
  
    } //fim for(i)  
    return soma/c;  
} //fim mediaQuadrado()
```



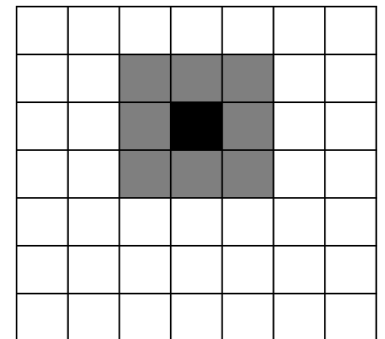
Media quadrado

```
float mediaQuadrado(float M[][MAX_COL], int linha, int coluna){  
    float soma=0;  
    int    c=0;  
    for(int i=linha-1 ; i<=linha+1 ; i++){ //varrendo as linhas  
        for(int j=coluna-1 ; j<=coluna+1 ; j++){//varrendo as colunas  
  
            } //fim for(j)  
        } //fim for(i)  
    return soma/c;  
} //fim mediaQuadrado()
```



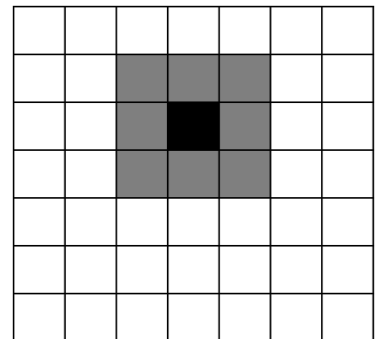
Media quadrado

```
float mediaQuadrado(float M[][MAX_COL], int linha, int coluna){  
    float soma=0;  
    int c=0;  
    for(int i=linha-1 ; i<=linha+1 ; i++){ //varrendo as linhas  
        for(int j=coluna-1 ; j<=coluna+1 ; j++){//varrendo as colunas  
  
            soma=soma+M[i][j];  
            c++;  
  
        } //fim for(j)  
    } //fim for(i)  
    return soma/c;  
} //fim mediaQuadrado()
```



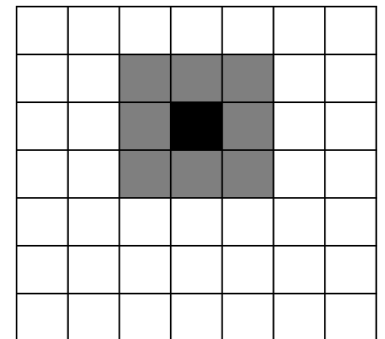
Media quadrado

```
float mediaQuadrado(float M[][MAX_COL], int linha, int coluna){
    float soma=0;
    int    c=0;
    for(int i=linha-1 ; i<=linha+1 ; i++){ //varrendo as linhas
        for(int j=coluna-1 ; j<=coluna+1 ; j++){//varrendo as colunas
            if( i>=0 && i<MAX_LIN && j>=0 && j<MAX_COL ){
                soma=soma+M[i][j];
                c++;
            } //fim if()
        } //fim for(j)
    } //fim for(i)
    return soma/c;
} //fim mediaQuadrado()
```



Media quadrado

```
float mediaQuadrado(float M[][MAX_COL], int linha, int coluna){
    float media=0;
    if( linha>=0 && linha<MAX_LIN && coluna>=0 && coluna<MAX_COL ){
        float soma=0;
        int c=0;
        for(int i=linha-1 ; i<=linha+1 ; i++){ //varrendo as linhas
            for(int j=coluna-1 ; j<=coluna+1 ; j++){//varrendo as colunas
                if( i>=0 && i<MAX_LIN && j>=0 && j<MAX_COL ){
                    soma=soma+M[i][j];
                    c++;
                }//fim if()
            } //fim for(j)
        } //fim for(i)
        media= soma/c;
    } //fim if()
    return media;
} //fim mediaQuadrado()
```



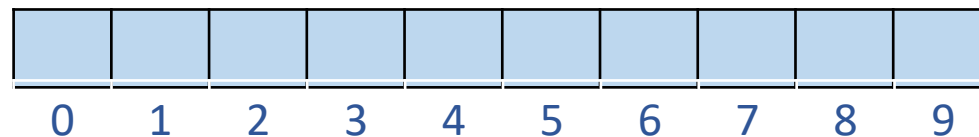
Inserindo em um vetor

Construa uma função que receba um vetor de reais e um valor x a ser inserido na posição i do vetor. Para que o valor x possa ser inserido sem impacto, antes da inserção os valores a partir daquela posição deverão ser deslocados uma posição à direita ($shift + 1$).

Considere o tamanho lógico do vetor definido na variável global **TAM**.

Uma função deverá implementar unicamente a funcionalidade do deslocamento à direita ($shift + 1$). O arranjo e a posição inicial do deslocamento devem ser parametrizados. Construa duas versões para esta função:

- a) Abordagem iterativa
- b) Abordagem recursiva

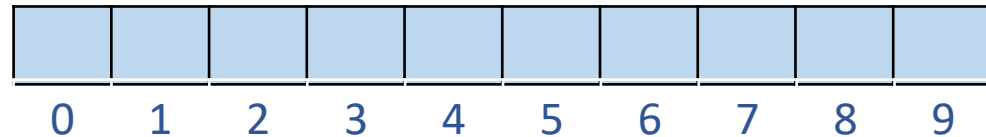


Identificadores globais

```
#include <stdio.h>  
#include <stdlib.h>  
#include <stdbool.h>
```

```
#define MAX 10
```

```
int TAM = 0;
```



Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){
```

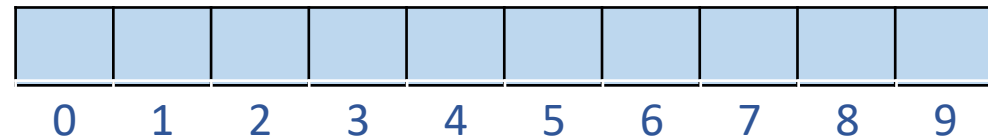
```
} //fim insira()
```



Inserindo em um vetor de reais

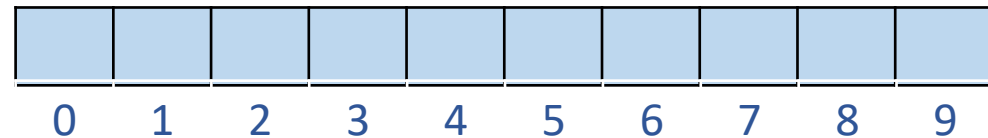
```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
```

```
    return sucesso;
} //fim insira()
```



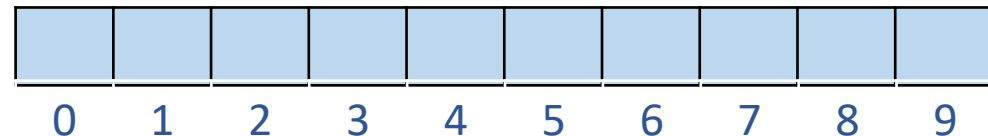
Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){  
    bool sucesso = false;  
    if( TAM < MAX ){  
  
        sucesso = true;  
    } //fim if()  
    return sucesso;  
} //fim insira()
```



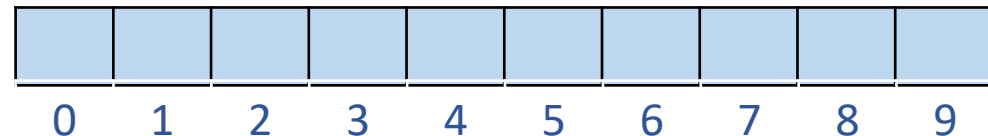
Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){  
    bool sucesso = false;  
    if( TAM < MAX ){  
        shiftMais1(A, posicao);  
  
        sucesso = true;  
    } //fim if()  
    return sucesso;  
} //fim insira()
```



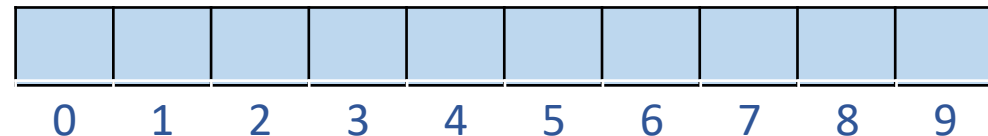
Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){  
    bool sucesso = false;  
    if( TAM < MAX ){  
        shiftMais1(A, posicao);  
        A[posicao] = X;  
  
        sucesso = true;  
    } //fim if()  
    return sucesso;  
} //fim insira()
```



Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){  
    bool sucesso = false;  
    if( TAM < MAX ){  
        shiftMais1(A, posicao);  
        A[posicao] = X;  
        TAM++;  
        sucesso = true;  
    } //fim if()  
    return sucesso;  
} //fim insira()
```

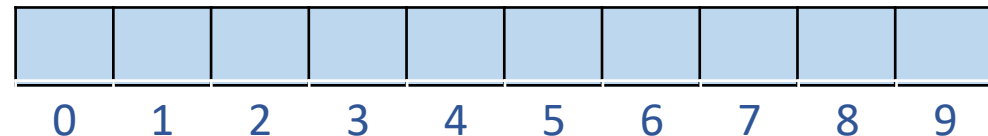


Shift à direita - Iterativo

Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){  
    bool sucesso = false;  
    if( TAM < MAX ){  
        shiftMais1(A, posicao);  
        A[posicao] = X;  
        TAM++;  
        sucesso = true;  
    } //fim if()  
    return sucesso;  
} //fim insira()
```

```
void shiftMais1(float A[], int posicao){  
  
  
  
  
  
} //fim shiftMais1()
```



Inserindo em um vetor de reais

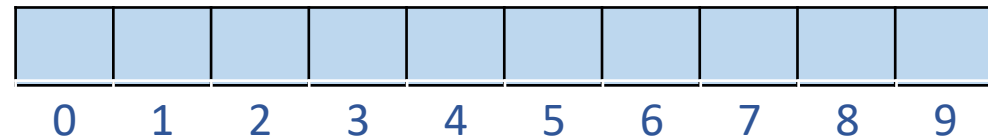
```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
    if( TAM < MAX ){
        shiftMais1(A, posicao);
        A[posicao] = X;
        TAM++;
        sucesso = true;
    } //fim if()
    return sucesso;
} //fim insira()
```

```
void shiftMais1(float A[], int posicao){

    for(int i=TAM; i>posicao; i--){

    } //fim for(i)

} //fim shiftMais1()
```



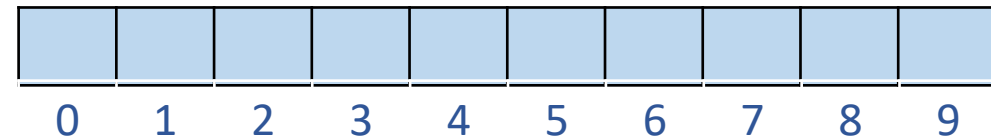
Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
    if( TAM < MAX ){
        shiftMais1(A, posicao);
        A[posicao] = X;
        TAM++;
        sucesso = true;
    } //fim if()
    return sucesso;
} //fim insira()
```

```
void shiftMais1(float A[], int posicao){

    for(int i=TAM; i>posicao; i--){
        A[i] = A[i-1];
    } //fim for(i)

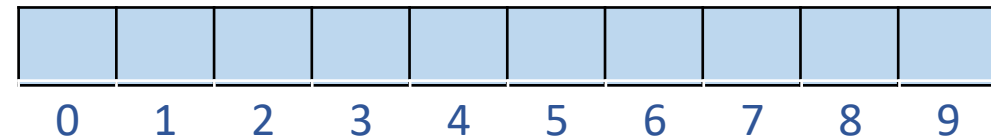
} //fim shiftMais1()
```



Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
    if( TAM < MAX ){
        shiftMais1(A, posicao);
        A[posicao] = X;
        TAM++;
        sucesso = true;
    } //fim if()
    return sucesso;
} //fim insira()
```

```
void shiftMais1(float A[], int posicao){
    if(posicao >= 0 && posicao <= TAM && TAM < MAX){
        for(int i=TAM; i>posicao; i--){
            A[i] = A[i-1];
        } //fim for(i)
    } //fim if()
} //fim shiftMais1()
```



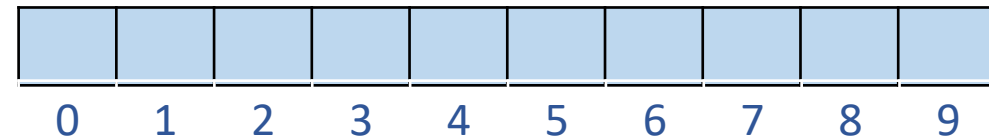
Shift à direita - Recursivo

Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
    if( TAM < MAX ){
        shiftMais1(A, posicao);
        A[posicao] = X;
        TAM++;
        sucesso = true;
    } //fim if()
    return sucesso;
} //fim insira()
```

```
void shiftMais1Rec(float A[], int posicao    )}
```

```
} //fim shiftMais1Rec()
```

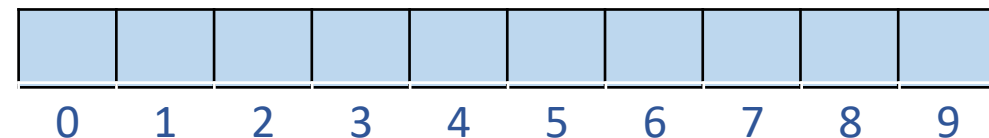


Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
    if( TAM < MAX ){
        shiftMais1(A, posicao);
        A[posicao] = X;
        TAM++;
        sucesso = true;
    } //fim if()
    return sucesso;
} //fim insira()
```

```
void shiftMais1Rec(float A[], int posicao, int frente){
```

```
} //jim shiftMais1Rec()
```



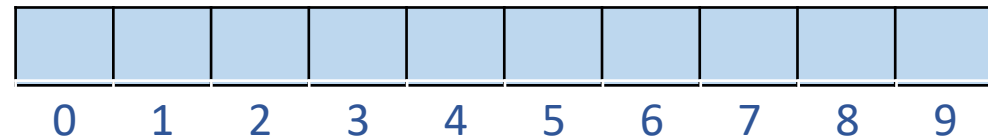
Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){  
    bool sucesso = false;  
    if( TAM < MAX ){  
        shiftMais1(A, posicao);  
        A[posicao] = X;  
        TAM++;  
        sucesso = true;  
    } //fim if()  
    return sucesso;  
} //fim insira()
```

```
void shiftMais1(float A[], int posicao){  
  
} //fim shiftMais1()
```

```
void shiftMais1Rec(float A[], int posicao, int frente){
```

```
} //fim shiftMais1Rec()
```



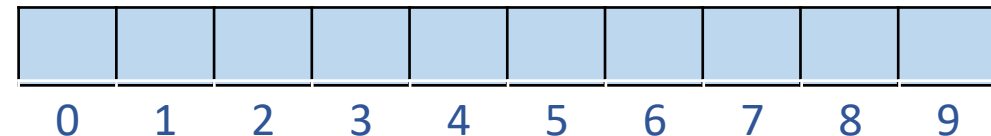
Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){  
    bool sucesso = false;  
    if( TAM < MAX ){  
        shiftMais1(A, posicao);  
        A[posicao] = X;  
        TAM++;  
        sucesso = true;  
    } //fim if()  
    return sucesso;  
} //fim insira()
```

```
void shiftMais1(float A[], int posicao){  
    shiftMais1Rec(A, posicao, TAM );  
} //fim shiftMais1()
```

```
void shiftMais1Rec(float A[], int posicao, int frente){
```

```
} //fim shiftMais1Rec()
```



Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
    if( TAM < MAX ){
        shiftMais1(A, posicao);
        A[posicao] = X;
        TAM++;
        sucesso = true;
    } //fim if()
    return sucesso;
} //fim insira()
```

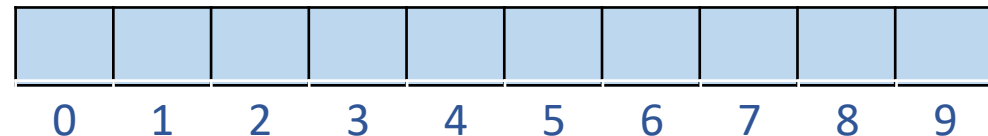
```
void shiftMais1(float A[], int posicao){
    shiftMais1Rec(A, posicao, TAM );
} //fim shiftMais1()

void shiftMais1Rec(float A[], int posicao, int frente){

    if(frente > posicao){

    } //fim if()

} //fim shiftMais1Rec()
```



Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
    if( TAM < MAX ){
        shiftMais1(A, posicao);
        A[posicao] = X;
        TAM++;
        sucesso = true;
    } //fim if()
    return sucesso;
} //fim insira()
```

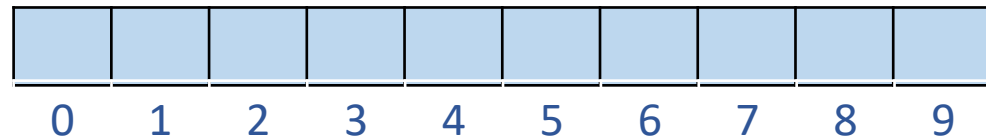
```
void shiftMais1(float A[], int posicao){
    shiftMais1Rec(A, posicao, TAM );
} //fim shiftMais1()

void shiftMais1Rec(float A[], int posicao, int frente){

    if(frente > posicao){
        A[frente] = A[frente-1];

    } //fim if()

} //fim shiftMais1Rec()
```



Inserindo em um vetor de reais

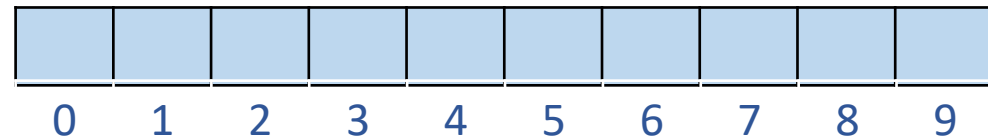
```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
    if( TAM < MAX ){
        shiftMais1(A, posicao);
        A[posicao] = X;
        TAM++;
        sucesso = true;
    } //fim if()
    return sucesso;
} //fim insira()
```

```
void shiftMais1(float A[], int posicao){
    shiftMais1Rec(A, posicao, TAM );
} //fim shiftMais1()

void shiftMais1Rec(float A[], int posicao, int frente){

    if(frente > posicao){
        A[frente] = A[frente-1];
        shiftMais1Rec(A, posicao, frente-1);
    } //fim if()

} //fim shiftMais1Rec()
```

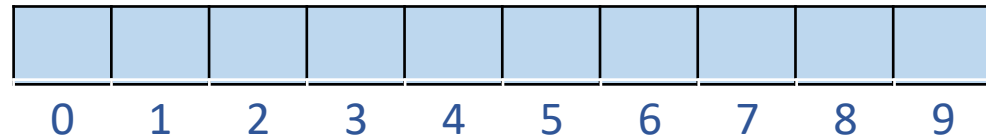


Inserindo em um vetor de reais

```
bool insira(float A[], float X, int posicao){
    bool sucesso = false;
    if( TAM < MAX ){
        shiftMais1(A, posicao);
        A[posicao] = X;
        TAM++;
        sucesso = true;
    } //fim if()
    return sucesso;
} //fim insira()
```

```
void shiftMais1(float A[], int posicao){
    shiftMais1Rec(A, posicao, TAM );
} //fim shiftMais1()
```

```
void shiftMais1Rec(float A[], int posicao, int frente){
    if(posicao >= 0 && posicao <= TAM && TAM < MAX && frente > 0 && frente <= TAM+1){
        if(frente > posicao){
            A[frente] = A[frente-1];
            shiftMais1Rec(A, posicao, frente-1);
        } //fim if()
    } //fim if()
} //fim shiftMais1Rec()
```



Listando invertido um vetor de estruturas

Considere uma Pessoa descrita pelos seguintes atributos:

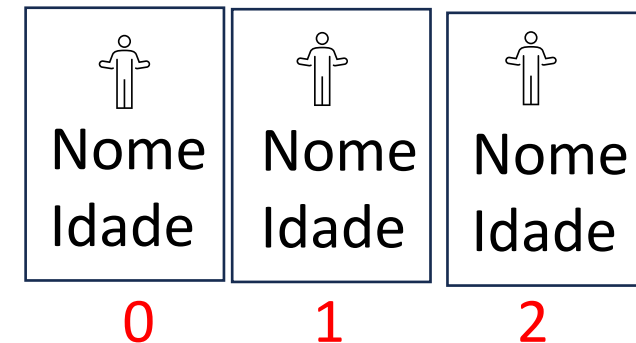
Nome da pessoa

Idade

Construa uma função que receba um arranjo de pessoas e as liste de forma invertida em relação ao arranjo: da última pessoa até à primeira.

a) Abordagem iterativa

b) Abordagem recursiva



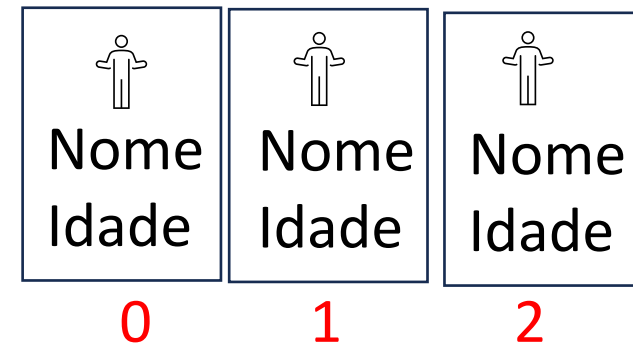
Considere uma Pessoa descrita pelos seguintes atributos:

Nome da pessoa

Idade

Construa uma função que receba um arranjo de pessoas e as liste de forma invertida em relação ao arranjo: da última pessoa até à primeira.

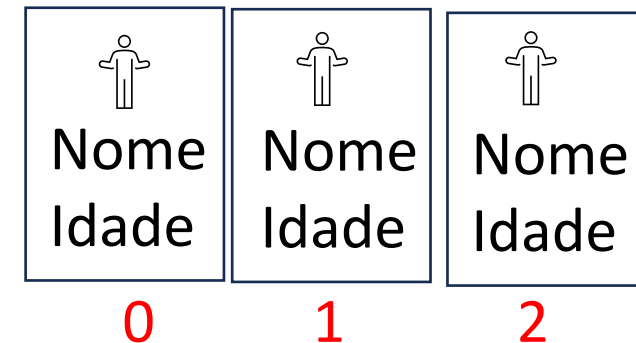
```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```



Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){  
  
  
} //fim listaInvertido()
```

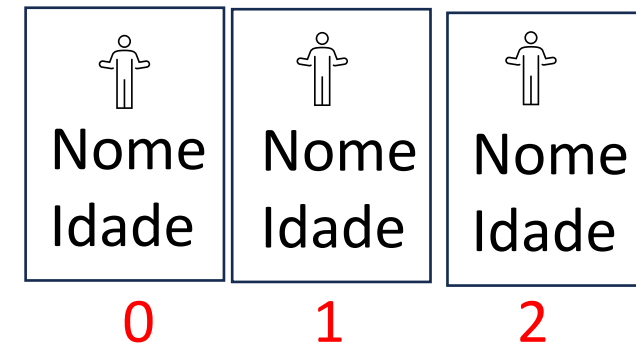
```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```



Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){  
    for(int i=TAM-1 ; i>=0 ; i--) {  
  
        } //fim for(i)  
    } //fim listaInvertido()
```

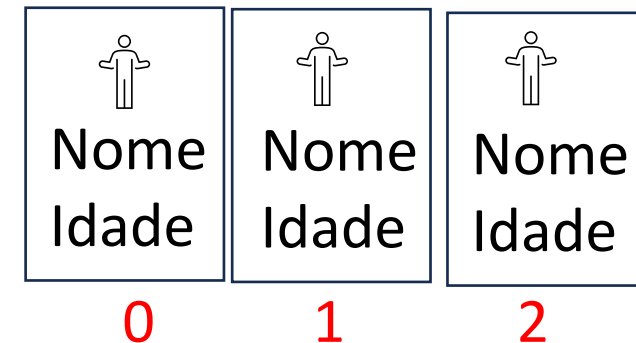
```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```



Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){  
    for(int i=TAM-1 ; i>=0 ; i--) {  
        escrevaPessoa(pessoas[i]);  
    } //fim for(i)  
} //fim listaInvertido()
```

```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```



Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){  
    for(int i=TAM-1 ; i>=0 ; i--) {  
        escrevaPessoa(pessoas[i]);  
    } //fim for(i)  
} //fim listaInvertido()
```

```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){
```

```
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){  
    for(int i=TAM-1 ; i>=0 ; i--) {  
        escrevaPessoa(pessoas[i]);  
    } //fim for(i)  
} //fim listaInvertido()
```

```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){  
    printf("\nNome: %s", PESSOA.nome );  
    printf("\nIdade: %d ", PESSOA.idade);  
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Abordagem recursiva

Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){  
    listaInvertidoRec(pessoas, TAM);  
} //fim listaInvertido()
```

```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){  
    printf("\nNome: %s", PESSOA.nome );  
    printf("\nIdade: %d ", PESSOA.idade);  
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){
    listaInvertidoRec(pessoas, TAM);
} //fim listaInvertido()

void listaInvertidoRec(Pessoa pessoas[], int N){
    if(N>0){

    }
} //fim listaInvertidoRec()
```

```
typedef struct{
    char nome[50];
    int idade;
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){
    printf("\nNome: %s", PESSOA.nome );
    printf("\nIdade: %d ", PESSOA.idade);
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){
    listaInvertidoRec(pessoas, TAM);
} //fim listaInvertido()

void listaInvertidoRec(Pessoa pessoas[], int N){
    if(N>0){
        escrevaPessoa(pessoas[N-1]);
    }
} //fim listaInvertidoRec()
```

```
typedef struct{
    char nome[50];
    int idade;
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){
    printf("\nNome: %s", PESSOA.nome );
    printf("\nIdade: %d ", PESSOA.idade);
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){
    listaInvertidoRec(pessoas, TAM);
} //fim listaInvertido()

void listaInvertidoRec(Pessoa pessoas[], int N){
    if(N>0){
        escrevaPessoa(pessoas[N-1]);
        listaInvertidoRec(pessoas, N-1);
    }
} //fim listaInvertidoRec()
```

```
typedef struct{
    char nome[50];
    int idade;
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){
    printf("\nNome: %s", PESSOA.nome );
    printf("\nIdade: %d ", PESSOA.idade);
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Listando em ordem

Listando pessoas de forma invertida

```
void listaInvertido(Pessoa pessoas[]){
    listaInvertidoRec(pessoas, TAM);
} //fim listaInvertido()

void listaInvertidoRec(Pessoa pessoas[], int N){
    if(N>0){
        escrevaPessoa(pessoas[N-1]);
        listaInvertidoRec(pessoas, N-1);
    }
} //fim listaInvertidoRec()
```

```
typedef struct{
    char nome[50];
    int idade;
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){
    printf("\nNome: %s", PESSOA.nome );
    printf("\nIdade: %d ", PESSOA.idade);
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Listando pessoas em ordem

```
void listaInvertido(Pessoa pessoas[]){
    listaInvertidoRec(pessoas, TAM);
} //fim listaInvertido()

void listaInvertidoRec(Pessoa pessoas[], int N){
    if(N>0){
        listaInvertidoRec(pessoas, N-1);
        escrevaPessoa(pessoas[N-1]);
    }
} //fim listaInvertidoRec()
```

```
typedef struct{
    char nome[50];
    int idade;
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){
    printf("\nNome: %s", PESSOA.nome );
    printf("\nIdade: %d ", PESSOA.idade);
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Chave de pesquisa: nome

Listando todas as pessoas

```
void listaPessoas(Pessoa pessoas[]){  
    for(int i=0; i<TAM ; i++) {  
  
        escrevaPessoa(pessoas[i]);  
  
    } //fim for(i)  
} //fim listaPessoas()
```

```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){  
    printf("\nNome: %s", PESSOA.nome );  
    printf("\nIdade: %d ", PESSOA.idade);  
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Chave de pesquisa: nome

```
void listaPessoas(Pessoa pessoas[], char* nome){  
    for(int i=0; i<TAM ; i++) {  
        if( strcmp(pessoas[i].nome, nome) ){  
            escrevaPessoa(pessoas[i]);  
        } //fim if()  
    } //fim for(i)  
} //fim listaPessoas()
```

```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```

```
void escrevaPessoa(Pessoa PESSOA){  
    printf("\nNome: %s", PESSOA.nome );  
    printf("\nIdade: %d ", PESSOA.idade);  
} //fim escrevaPessoa()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Chave de pesquisa: nome

```
void listaPessoas(Pessoa pessoas[], char* nome){  
    for(int i=0; i<TAM ; i++) {  
        if( strIguals(pessoas[i].nome, nome) ){  
            escrevaPessoa(pessoas[i]);  
        } //fim if()  
    } //fim for(i)  
} //fim listaPessoas()
```

```
bool strIguals(char str1[], char str2[]) {  
    bool iguais=true;  
    int i=0;  
    while( iguais && i<maxStr  
           && str1[i]!='\0' && str2[i]!='\0') {  
        if( toupper(str1[i]) != toupper(str2[i])){  
            iguais=false;  
        }  
        i++;  
    }  
    return iguais;  
} //fim strIguals()
```

Chave de pesquisa: nome
Abordagem recursiva

Listando todas as pessoas

```
void listaPessoas(Pessoa pessoas[]){  
    listaPessoasRec(pessoas, TAM);  
} //fim listaPessoas()  
  
void listaPessoasRec(Pessoa pessoas[], int N){  
    if(N>0){  
  
        listaPessoasRec(pessoas, N-1);  
  
        escrevaPessoa(pessoas[N-1]);  
    }  
} //fim listaPessoasRec()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Listando pessoas: chave de pesquisa nome

```
void listaPessoasNome(Pessoa pessoas[], char* nome){
    listaPessoasNomeRec(pessoas, nome, TAM);
} //fim listaPessoasNome()

void listaPessoasNomeRec(Pessoa pessoas[], char* nome, int N){
    if(N>0){
        listaPessoasNomeRec(pessoas, N-1);
        if( strcmp(pessoas[i].nome, nome) ){
            escrevaPessoa(pessoas[N-1]);
        }
    }
} //fim listaPessoasNomeRec()
```

Nome Idade	Nome Idade	Nome Idade
0	1	2

Questões em listas de exercícios

Palíndromo - Iterativo

Construa uma função que verifique se uma *string* é um palíndromo ou não. A função deverá retornar *verdadeiro*, se o for, ou *falso*, caso contrário. Um palíndromo corresponde a sequências que podem ser lidas da mesma forma em ambas as direções – da esquerda para a direita e também da direita para esquerda. Por exemplo, são palíndromos: ARARA, 1001, REVIVER, RADAR.

- A *string* deverá ser um argumento da função (recebida por parâmetro)

a) Versão iterativa

b) Versão recursiva

Buscando padrão de comportamento

1001

RADAR

ARARA

ANILINA

REVIVER

R	E	V	I	V	E	R	\0		
0	1	2	3	4	5	6	7	8	9

Buscando padrão de comportamento

1001


RADAR

ARARA

ANILINA

REVIVER

R	E	V	I	V	E	R	\0		
0	1	2	3	4	5	6	7	8	9



Buscando padrão de comportamento

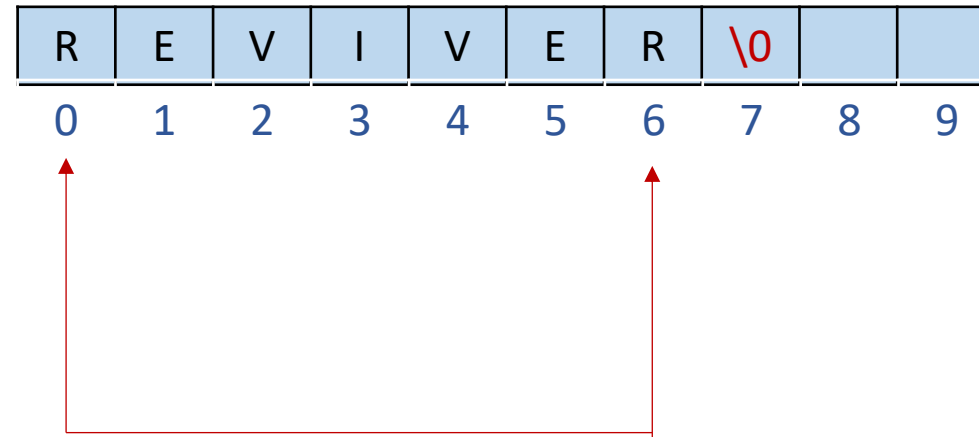
1001

RADAR

ARARA

ANILINA

REVIVER



Buscando padrão de comportamento

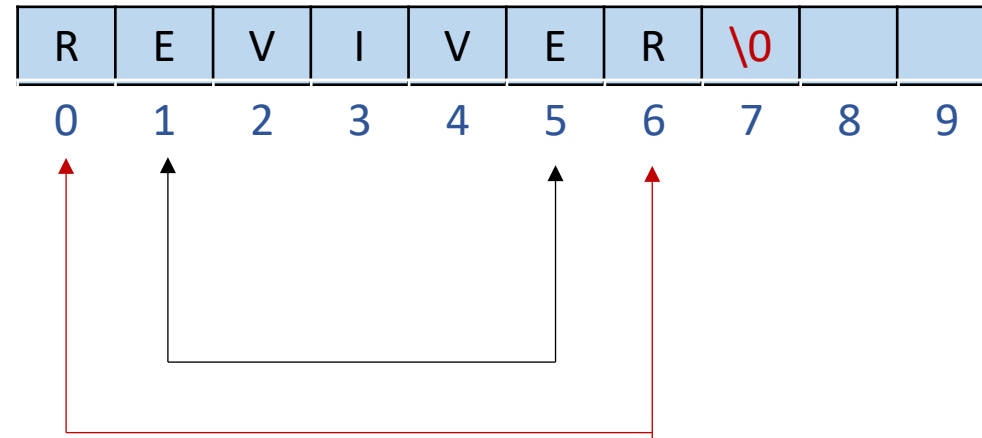
1001

RADAR

ARARA

ANILINA

REVIVER



Buscando padrão de comportamento

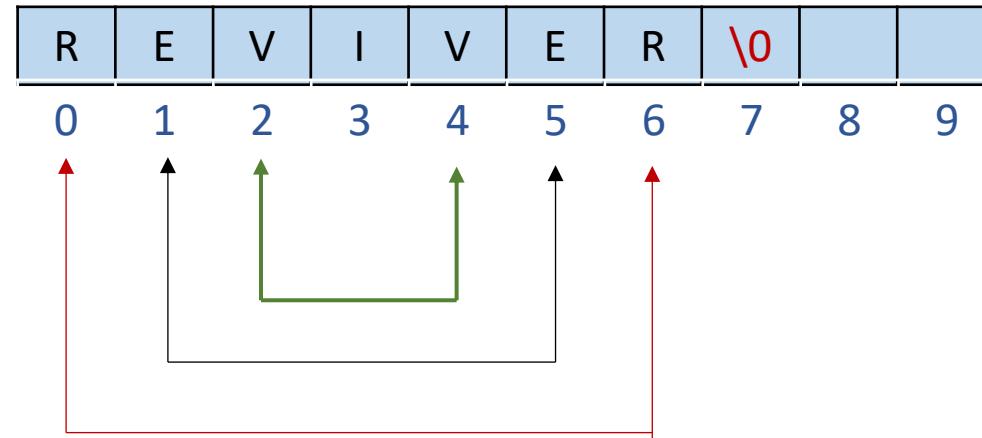
1001

RADAR

ARARA

ANILINA

REVIVER



Buscando padrão de comportamento

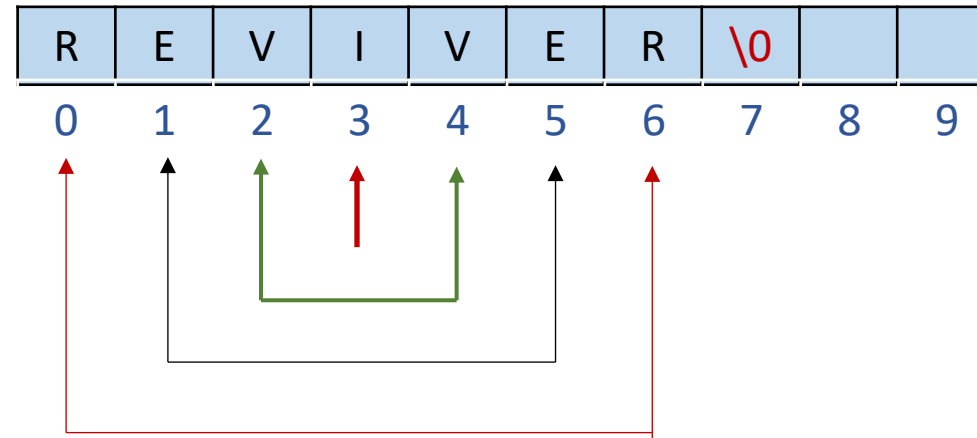
1001

RADAR

ARARA

ANILINA

REVIVER



Buscando padrão de comportamento

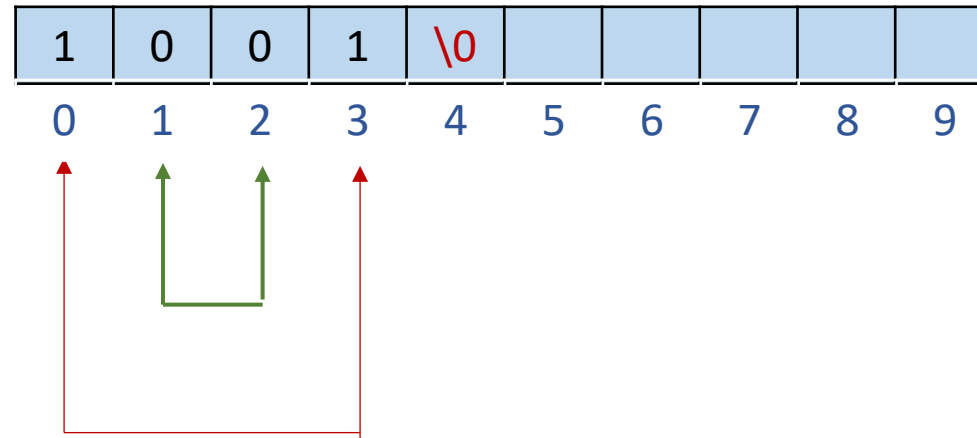
1001

RADAR

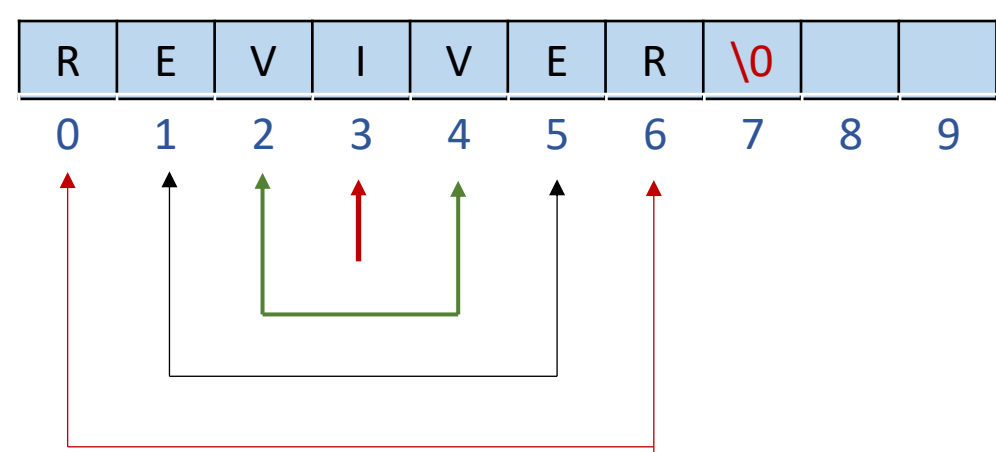
ARARA

ANILINA

REVIVER



Versão Iterativa



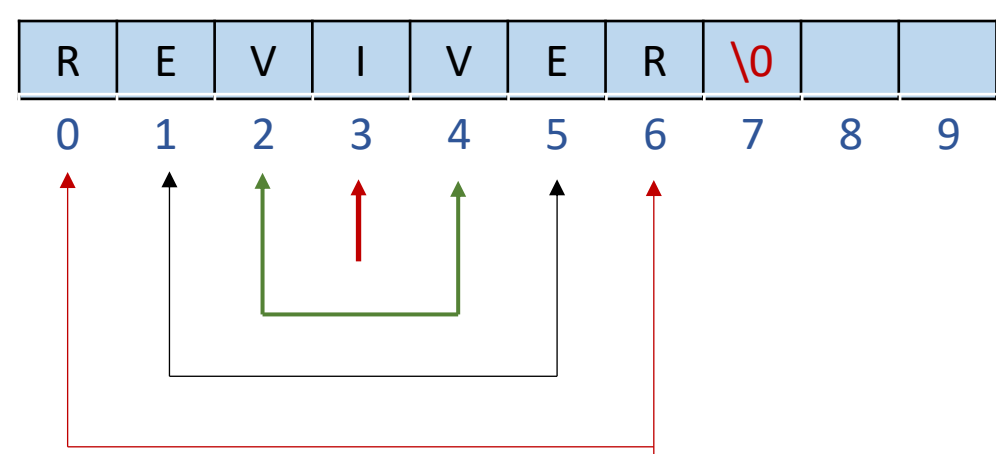
```
bool ehPalindromo(char* str){
```

```
    bool palindromo= true;
```

```
        return palindromo;
```

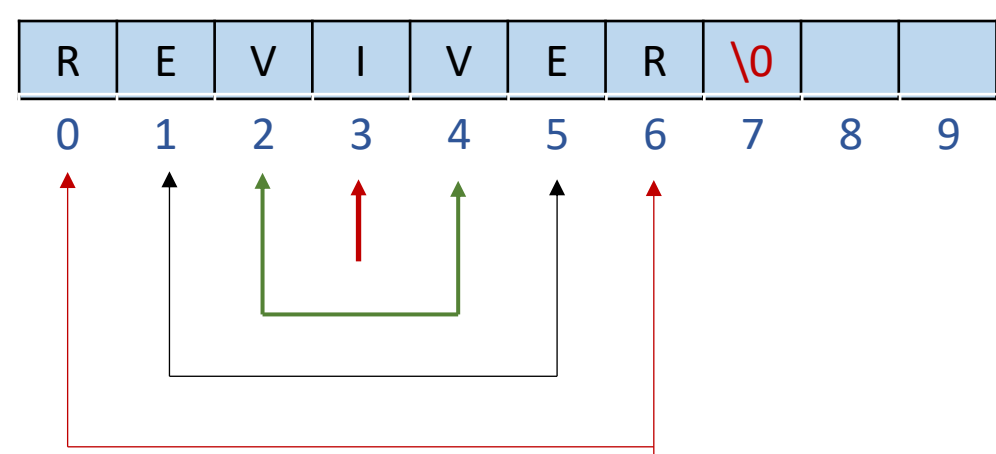
```
    } //fim ehPalindromo()
```

Versão Iterativa



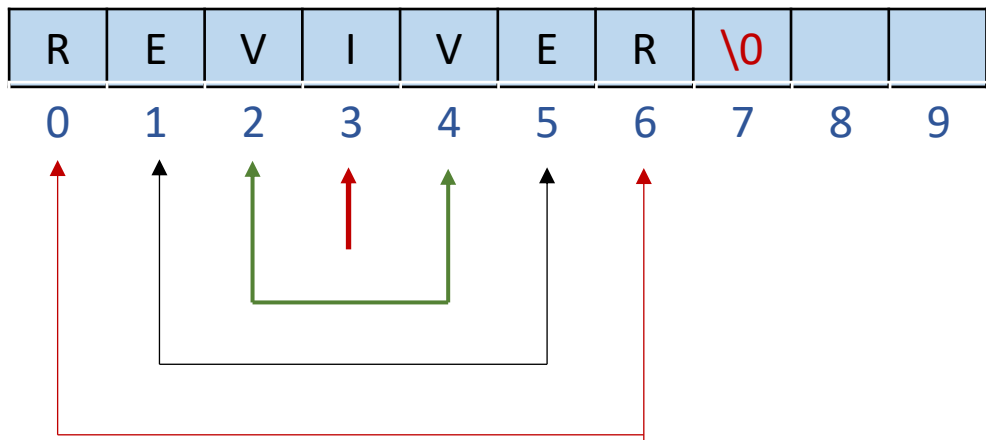
```
bool ehPalindromo(char* str){  
    bool palindromo= true;  
    int i=0;  
    int j=strlen(str)-1;  
  
    return palindromo;  
} //fim ehPalindromo()
```

Versão Iterativa



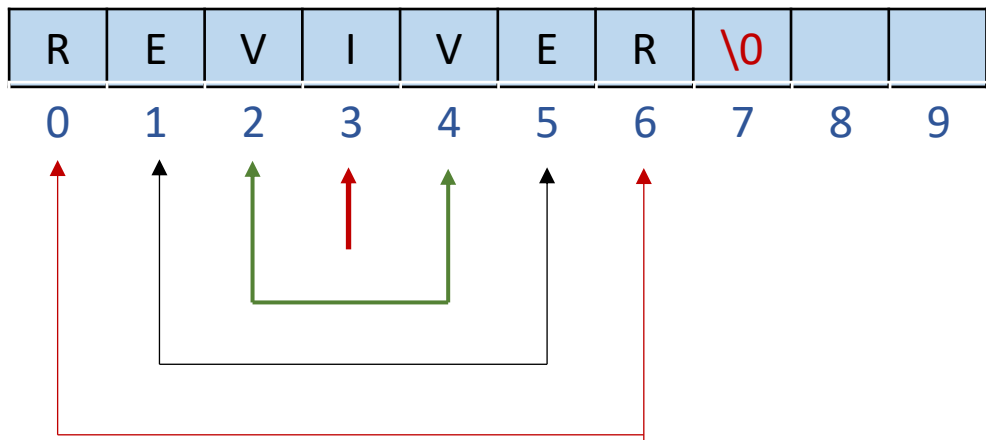
```
bool ehPalindromo(char* str){  
    bool palindromo= true;  
    int i=0;  
    int j=strlen(str)-1;  
    while(palindromo && i<j){  
  
        } //fim while()  
    return palindromo;  
} //fim ehPalindromo()
```

Versão Iterativa



```
bool ehPalindromo(char* str){  
    bool palindromo= true;  
    int i=0;  
    int j=strlen(str)-1;  
    while(palindromo && i<j){  
        if(str[i]!=str[j])palindromo= false;  
  
        } //fim while()  
    return palindromo;  
} //fim ehPalindromo()
```

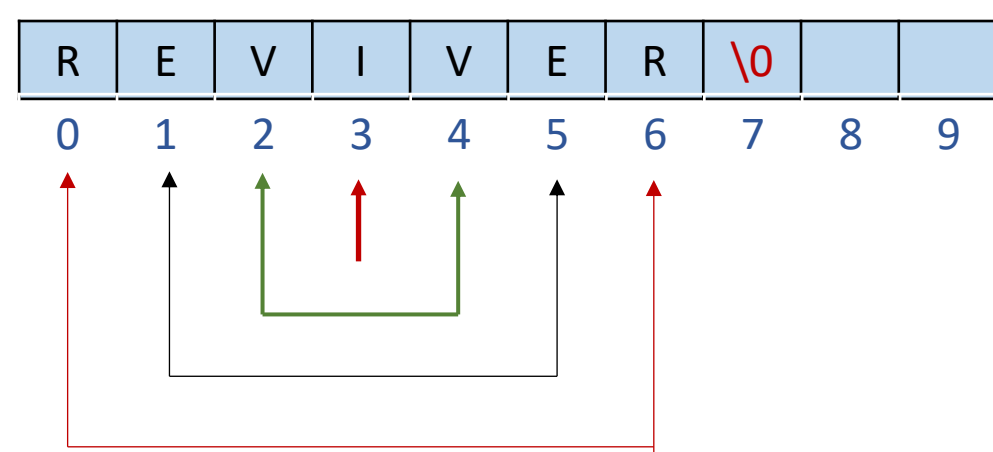
Versão Iterativa



```
bool ehPalindromo(char* str){  
    bool palindromo= true;  
    int i=0;  
    int j=strlen(str)-1;  
    while(palindromo && i<j){  
        if(str[i]!=str[j])palindromo= false;  
        i++; j--;  
    } //fim while()  
    return palindromo;  
} //fim ehPalindromo()
```

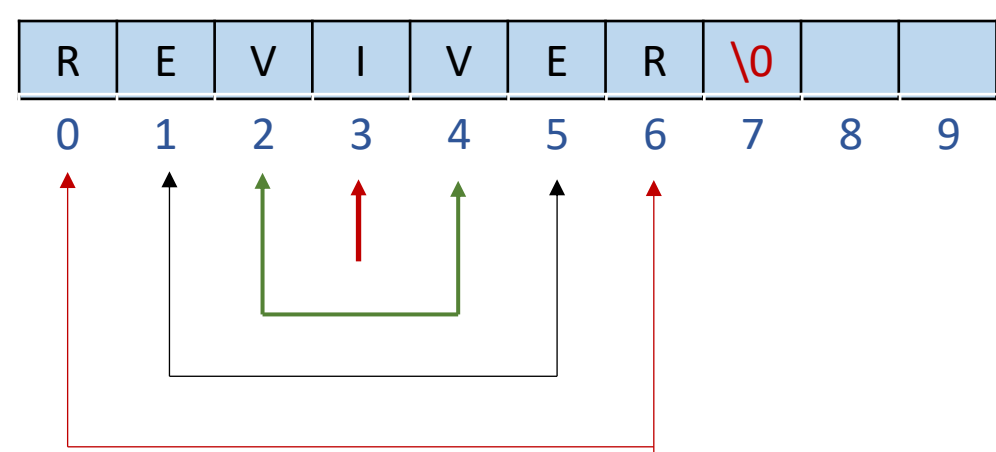

Versão recursiva

Versão Iterativa



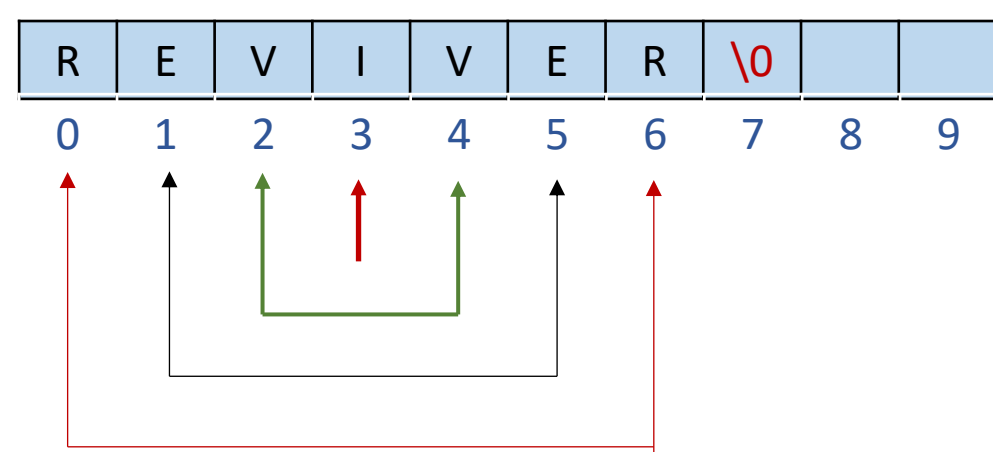
```
bool ehPalindromoRec(char* str){  
    return ehPalindromoRec2(str,    ,    );  
} //fim ehPalindromo()
```

Versão Iterativa



```
bool ehPalindromoRec(char* str){  
    return ehPalindromoRec2(str, 0, strlen(str)-1 );  
} //fim ehPalindromo()
```

Versão Iterativa

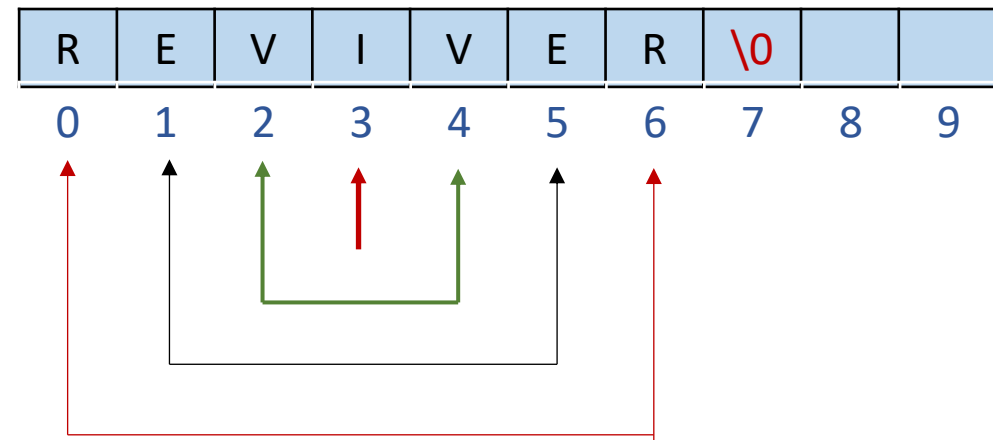


```
bool ehPalindromoRec(char* str){  
    return ehPalindromoRec2(str, 0, strlen(str)-1 );  
} //fim ehPalindromo()
```

```
bool ehPalindromoRec2(char* str, int i, int j){
```

```
} //fim ehPalindromo()
```

Versão Iterativa

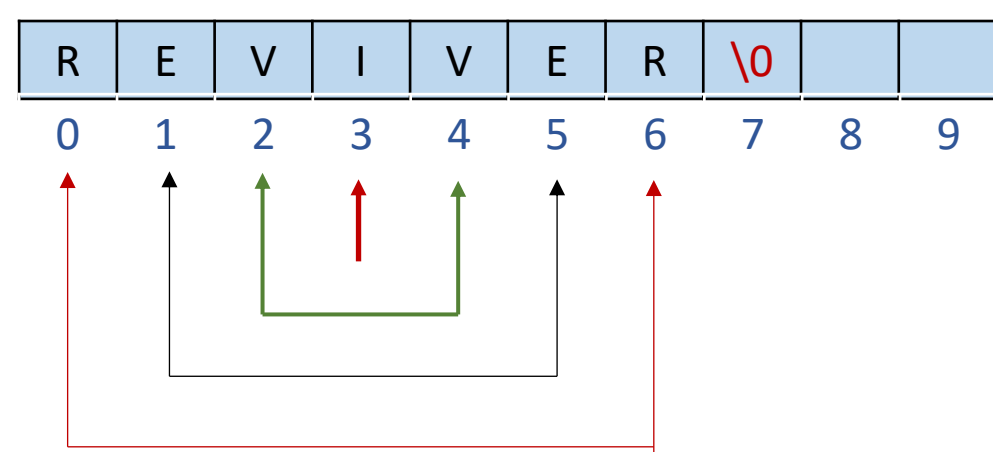


```
bool ehPalindromoRec(char* str){  
    return ehPalindromoRec2(str, 0, strlen(str)-1 );  
} //fim ehPalindromo()
```

```
bool ehPalindromoRec2(char* str, int i, int j){  
    bool palindromo= true;
```

```
    return palindromo;  
} //fim ehPalindromo()
```

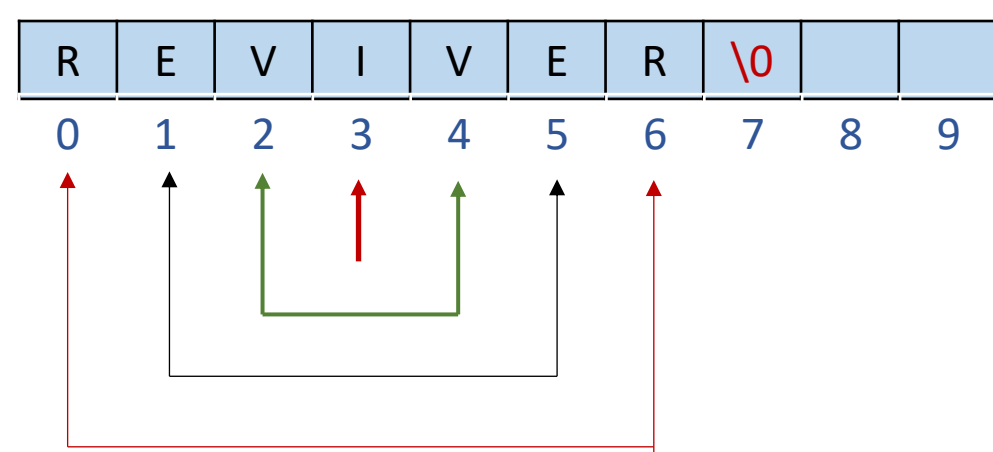
Versão Iterativa



```
bool ehPalindromoRec(char* str){  
    return ehPalindromoRec2(str, 0, strlen(str)-1 );  
} //fim ehPalindromo()
```

```
bool ehPalindromoRec2(char* str, int i, int j){  
    bool palindromo= true;  
    if( i< j ){  
  
    }  
    return palindromo;  
} //fim ehPalindromo()
```

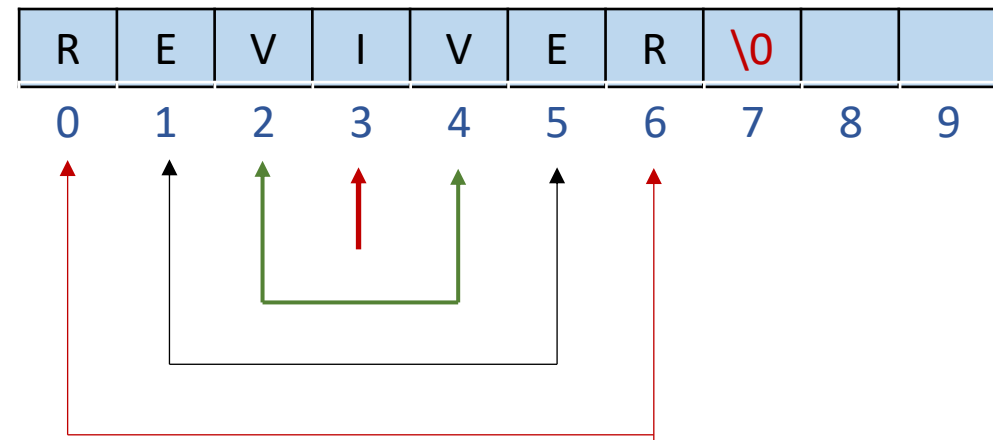
Versão Iterativa



```
bool ehPalindromoRec(char* str){  
    return ehPalindromoRec2(str, 0, strlen(str)-1 );  
} //fim ehPalindromo()
```

```
bool ehPalindromoRec2(char* str, int i, int j){  
    bool palindromo= true;  
    if( i< j ){  
        if(str[i]!=str[j])palindromo= false;  
        else  
        }  
    return palindromo;  
} //fim ehPalindromo()
```

Versão Iterativa



```
bool ehPalindromoRec(char* str){  
    return ehPalindromoRec2(str, 0, strlen(str)-1 );  
} //fim ehPalindromo()
```

```
bool ehPalindromoRec2(char* str, int i, int j){  
    bool palindromo= true;  
    if( i< j ){  
        if(str[i]!=str[j])palindromo= false;  
        else                palindromo= ehPalindromoRec2(str, i+1, j-1);  
    }  
    return palindromo;  
} //fim ehPalindromo()
```


Questão

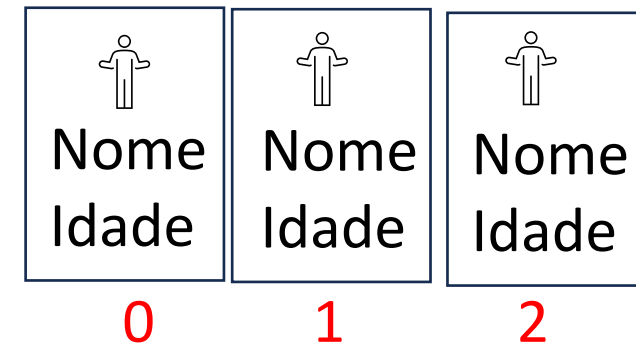
Inserindo Pessoa

Construa uma função que receba um vetor de pessoas, uma estrutura de pessoa e a posição do vetor em que a estrutura deverá ser inserida. Se a posição estiver ocupada, todos os elementos daquela posição em diante deverão ser deslocados uma posição à direita para permitir a inserção.

O deslocamento deverá ser implementado utilizando a abordagem recursiva.

Considere o tamanho físico do arranjo definido em MAX e o tamanho lógico armazenado na variável TAM

```
typedef struct{  
    char nome[50];  
    int  idade;  
} Pessoa;
```



Verde – Turma Teórica (Final 00)

Questões

Dois desafios foram postados no Verde.

Postar solução até o início da próxima aula:

Até quarta-feira, dia 13, às 8h50

Valor: **1 ponto**

Para a prova, certifique de estar em dia com todas as listas de exercícios apresentadas, em especial, com o Trabalho Prático Final.