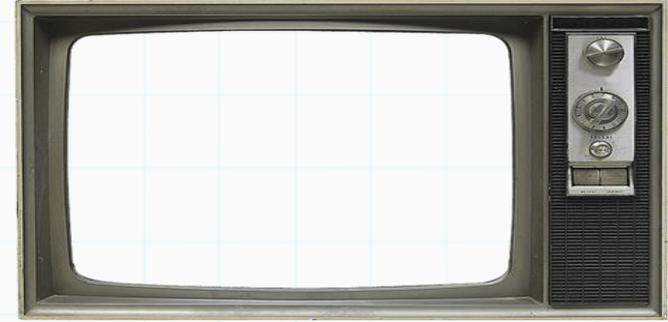


Programação Estruturada

Professor : Yuri Frota

yuri@ic.uff.br



Funções

- Funções são unidades modularizadas de código com o objetivo de realizar uma tarefa
- Permitem que trechos de código sejam reutilizados
- Até agora, criamos programas apenas com a função principal `main()`

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

```
int main()
```

```
{
```

```
    return 0;
```

```
}
```



Funções

- Funções são unidades modularizadas de código com o objetivo de realizar uma tarefa
- Permitem que trechos de código sejam reutilizados
- Até agora, criamos programas apenas com a função principal `main()`
- Em C, a definição de uma função tem a seguinte forma

```
tipo_retorno nome_função (parâmetros)
{
    Declarações e comandos
}
```

- Os parâmetros são declarados como:
`tipo1 nome1, tipo2 nome2, tipo3 nome3 ...`
- A comunicação entre as funções é feita através da passagem de parâmetros ou através de variáveis globais



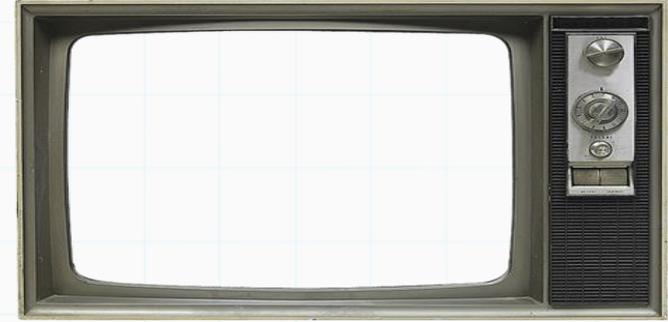
Funções

Uma função pode retornar uma informação ou pode não retornar nada.

Exemplo:

```
void escreve_menu()  
{  
    printf("MENU \n");  
    printf("1 - somar\n");  
    printf("2 - subtrair\n");  
    printf("3 - fim\n");  
}
```

Não retorna nada



Funções

Uma função pode retornar uma informação ou pode não retornar nada.

Exemplo:

```
void escreve_menu()  
{  
    printf("MENU \n");  
    printf("1 - somar\n");  
    printf("2 - subtrair\n");  
    printf("3 - fim\n");  
}
```

Não retorna nada



Para que a função retorne um valor para sua função chamadora é usado o comando **return**
Se o tipo da função for omitido, o default é o tipo **int**

```
int soma(int A, int B)  
{  
    int C;  
    C = A + B;  
    return(C);  
}
```

ou

```
int soma(int A, int B) {  
    return(A + B);  
}
```

Funções

- O tipo da função sempre indica o tipo do valor que será retornado
- Uma função pode possuir mais de um **return**

```
int funcao(int A, int B, int flag)
{
    if (flag == 1)
        return(A + B);
    if (flag == 2)
        return (A - B);

    return (A * B);
}
```



Funções

- O tipo da função sempre indica o tipo do valor que será retornado
- Uma função pode possuir mais de um **return**
- A função termina sua execução quando encontrar um **return** ou quando for encontrado o seu fim

```
void funcao(int n)
{
    for (int i=0; i<n; i++)
        printf("*");
}
```

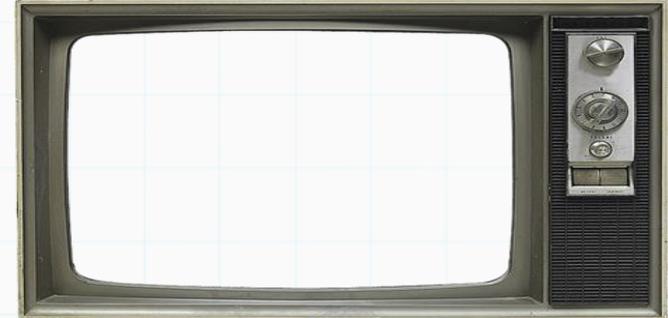
Retorno sem
comando return



Funções

- O tipo da função sempre indica o tipo do valor que será retornado
- Uma função pode possuir mais de um **return**
- A função termina sua execução quando encontrar um **return** ou quando for encontrado o seu fim
- Uma função pode ser declarada antes ou depois da função **main** (por padrão sempre antes)

```
int soma(int A, int B) {  
    return(A + B);  
}  
  
int main() {  
    int n1, n2, resultado;  
    printf("Digite dois valores inteiros: ");  
    scanf("%d %d", &n1, &n2);  
    resultado = soma(n1,n2);  
    printf("O resultado é: %d", resultado);  
    return 0;  
}
```



Funções

- Exemplo:

```
int par(int N) {
    if (N % 2 == 0)
        return 1;
    else
        return 0;
}

int main() {
    int N;
    printf("Digite um valor inteiro: ");
    scanf("%d", &N);
    if (par(N))
        printf("O número é PAR");
    else
        printf("O número é ÍMPAR");
    return 0;
}
```



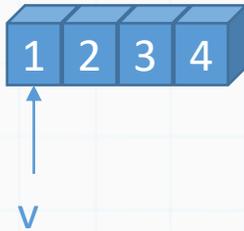
Funções

Vetores como parâmetros :

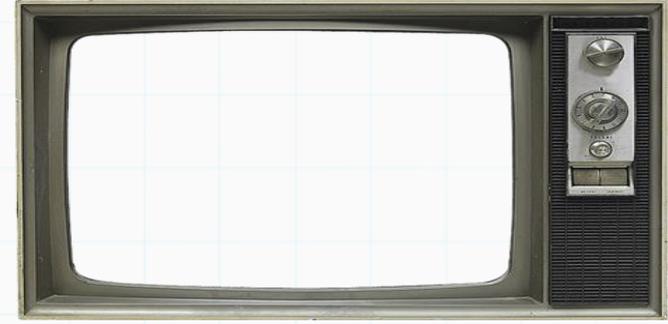
- Vetores em C são na verdade ponteiros para a primeira posição do vetor. Veremos o conceito de ponteiros em aulas futuras
- Exemplo:

```
int v[4] = {1, 2, 3, 4};
```

- Na memória



v é um ponteiro para o primeiro inteiro do vetor



Funções

Vetores como parâmetros :

- Ao se passar um vetor (estático) como parâmetro para uma função, devemos passar seu tamanho para o compilador saber que tamanho de vetor esta recebendo.



```
int main (void) {
    int n=10;
    int i, vetor[n];
    float media;

    // lendo os valores do vetor
    for (i=0; i<n; i++) {
        printf("Digite um número: ");
        scanf("%d", &vetor[i]);
    }
    media = calculaMedia(n, vetor);

    printf("A média é: %.2f", media);
    return 0;
}
```

```
float calculaMedia(int n, int
v[n]) {

    int i, soma=0;
    for (i=0; i<n; i++)
        soma += v[i];

    return (soma/(n*1.0));
}
```

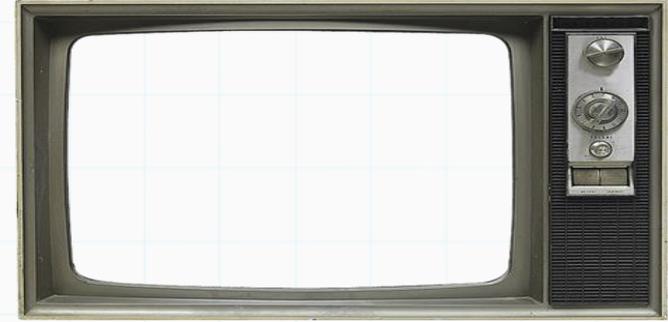
parâmetro n deve vir
antes do parâmetro
v[n] para o
compilador saber
quem é n



Funções

Vetores como parâmetros :

- Ao se passar um vetor (estático) como parâmetro para uma função, devemos passar seu tamanho para o compilador saber que tamanho de vetor esta recebendo.



```
int main (void) {
    int n=10;
    int i, vetor[n];
    float media;

    // lendo os valores do vetor
    for (i=0; i<n; i++) {
        printf("Digite um número: ");
        scanf("%d", &vetor[i]);
    }
    media = calculaMedia(n, vetor);

    printf("A média é: %.2f", media);
    return 0;
}
```

- Mas lembre-se, estamos passando uma cópia do ponteiro para a primeira posição do vetor, por isso as alterações no vetor dentro da função são feitas no vetor original !

```
float calculaMedia(int n, int
v[n]) {

    int i, soma=0;
    for (i=0; i<n; i++)
        soma += v[i];

    return (soma/(n*1.0));
}
```

parâmetro n deve vir antes do parâmetro v[n] para o compilador saber quem é n



Funções

Vetores como parâmetros :

- Ao se passar um vetor (estático) como parâmetro para uma função, devemos passar seu tamanho para o compilador saber que tamanho de vetor esta recebendo. **Compiladores modernos aceitam passar os vetores sem a definição de tamanho (na duvida, sempre coloque o tamanho)**

```
int main (void) {
    int n=10;
    int i, vetor[n];
    float media;

    // lendo os valores do vetor
    for (i=0; i<n; i++) {
        printf("Digite um número: ");
        scanf("%d", &vetor[i]);
    }
    media = calculaMedia(n, vetor);

    printf("A média é: %.2f", media);
    return 0;
}
```

- Mas lembre-se, estamos passando uma cópia do ponteiro para a primeira posição do vetor, por isso as alterações no vetor dentro da função são feitas no vetor original !

```
float calculaMedia(int n, int v[])
{
    int i, soma=0;
    for (i=0; i<n; i++)
        soma += v[i];

    return (soma/(n*1.0));
}
```

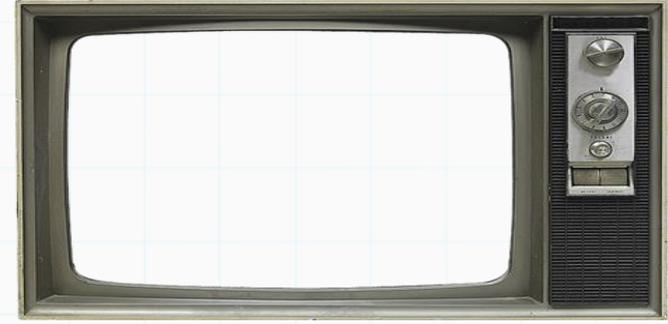
parâmetro n deve vir antes do parâmetro v[n] para o compilador saber quem é n



Funções

Vetores como parâmetros :

- E matrizes (estáticas) ? Matrizes não temos opção, temos que sempre informar seus tamanhos na função.



```
int main (void) {
    int n=10, m=10;
    int i, matriz[n][m];
    float media;

    // lendo os valores do vetor
    for (i=0; i<TAM; i++) {
        printf("Digite um número: ");
        scanf("%d", &vetor[i]);
    }
    media = calculaMedia(n, m, matriz);

    printf("A média é: %.2f", media);
    return 0;
}
```

parâmetros n e m devem vir antes do parâmetro m[n][m] para o compilador saber quem é n e m

```
float calculaMedia(int n, int m, int matriz[n][m])
{
    int i, j, soma=0;
    for (i=0; i<n; i++)
        for (j=0; j<m; j++)
            soma += matriz[i][j];

    return (soma/((n*m)*1.0));
}
```

Funções



```
int A, B, C;
```

```
int teste1(int A) {  
    int D;  
    ...  
}
```

```
int teste2() {  
    int C;  
    ...  
}
```

```
int main() {  
    int B;  
    A = teste1(B);  
    C = teste2();  
    return 0;  
}
```

- Escopo de variáveis:

- Lembrando...

- Variáveis **Globais** são todas as variáveis declaradas fora de qualquer função e podem ser vistas/alteradas por todas as funções do programa.

Funções



```
int A, B, C;
```

```
int teste1(int A) {  
    int D;  
    ...  
}
```

```
int teste2() {  
    int C;  
    ...  
}
```

```
int main() {  
    int B;  
    A = teste1(B);  
    C = teste2();  
    return 0;  
}
```

- Escopo de variáveis:
 - Relembrando...
 - Variáveis **Globais** são todas as variáveis declaradas fora de qualquer função e podem ser vistas/alteradas por todas as funções do programa.
 - Uma variável **Local** só pode ser reconhecida dentro da função/bloco onde ela foi declarada. É criada ao chamar a função e destruída ao sair dela.

Funções



```
int A, B, C;
```

```
int teste1(int A) {  
    int D;  
    ...  
}
```

A e D são locais

B e C são globais

```
int teste2() {  
    int C;  
    ...  
}
```

```
int main() {  
    int B;  
    A = teste1(B);  
    C = teste2();  
    return 0;  
}
```

- Escopo de variáveis:

- Relembrando...

- Variáveis **Globais** são todas as variáveis declaradas fora de qualquer função e podem ser vistas/alteradas por todas as funções do programa.

- Uma variável **Local** só pode ser reconhecida dentro da função/bloco onde ela foi declarada. É criada ao chamar a função e destruída ao sair dela.

Funções



```
int A, B, C;
```

```
int teste1(int A) {  
    int D;  
    ...  
}
```

A e D são locais

B e C são globais

```
int teste2() {  
    int C;  
    ...  
}
```

C é local

A e B são globais

```
int main() {  
    int B;  
    A = teste1(B);  
    C = teste2();  
    return 0;  
}
```

- Escopo de variáveis:

- Relembrando...

- Variáveis **Globais** são todas as variáveis declaradas fora de qualquer função e podem ser vistas/alteradas por todas as funções do programa.

- Uma variável **Local** só pode ser reconhecida dentro da função/bloco onde ela foi declarada. É criada ao chamar a função e destruída ao sair dela.

Funções



```
int A, B, C;
```

```
int teste1(int A) {  
    int D;  
    ...  
}
```

A e D são locais
B e C são globais

```
int teste2() {  
    int C;  
    ...  
}
```

C é local
A e B são globais

```
int main() {  
    int B;  
    A = teste1(B);  
    C = teste2();  
    return 0;  
}
```

B é local
A e C são globais

- Escopo de variáveis:
 - Relembrando...

- Variáveis **Globais** são todas as variáveis declaradas fora de qualquer função e podem ser vistas/alteradas por todas as funções do programa.
- Uma variável **Local** só pode ser reconhecida dentro da função/bloco onde ela foi declarada. É criada ao chamar a função e destruída ao sair dela.

Funções



Fura Olho: O que será escrito ?

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

```
int A, B, C;
```

```
int teste(int A) {
```

```
    int D = 2;
```

```
    A++;
```

```
    C += A;
```

```
    D -= (A+B);
```

```
    return D+1;
```

```
}
```

```
int main() {
```

```
    A = 5;
```

```
    B = 6;
```

```
    B -= A;
```

```
    C = B + A;
```

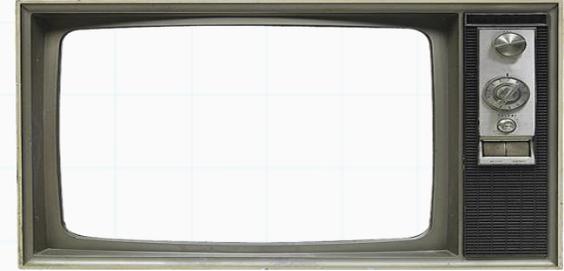
```
    B = B + teste(B);
```

```
    printf("A = %d, B = %d, C = %d", A, B, C);
```

```
    return 0;
```

```
}
```

Funções



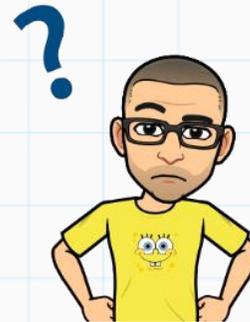
1) Maior elemento: Escreva uma função que receba vetor de inteiros de tamanho $n > 0$. A função deve encontrar o menor elemento e depois diminuir este menor elemento de cada elemento do vetor. Para isso, usa a função a seguir, que encontra e retorna o menor elemento e altera o vetor.

```
int menor_ele(int n, int v[n])
```

Exemplo:

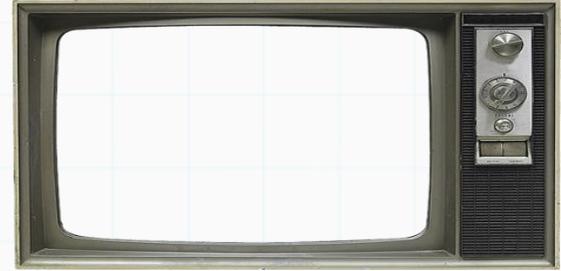
```
vetor: 4 , 8 , 7 , 88 , 12 , 23 ,  
menor = 4  
novo vetor: 0 , 4 , 3 , 84 , 8 , 19 ,
```

```
int soma(int A, int B)  
{  
    int C;  
    C = A + B;  
    return(C);  
}
```



Funções

```
int menor_ele(int n, int v[n]) {  
    int i;  
    float menor=v[0];  
  
    for (i = 1; i < n; i++)  
        if (menor > v[i])  
            menor = v[i];  
  
    for (i = 0; i < n; i++)  
        v[i] = v[i] - menor;  
  
    return menor;  
}
```



Funções

2) Séries: Faça um programa que recebe um valor inteiro e positivo $n > 0$ e retorna o valor de S , calculado pela fórmula

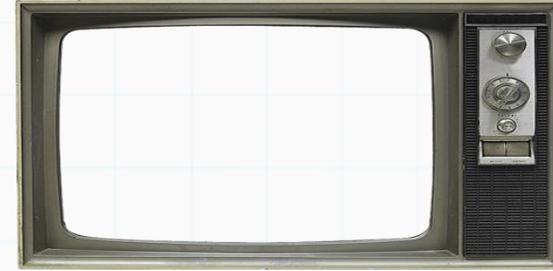
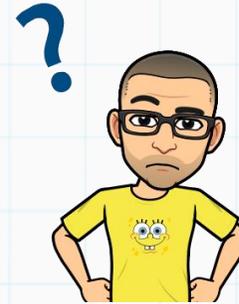
$$S = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/N!$$

Para calcular S , implemente uma função chamada

```
float series(int n)
```

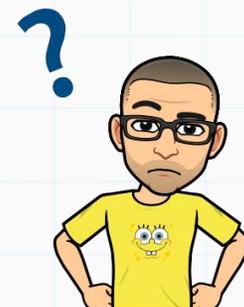
E essa função deve utilizar uma segunda função chamada

```
int fat(int f)
```



```
int soma(int A, int B)
{
    int C;
    C = A + B;
    return(C);
}
```

Funções



```
#include <stdio.h>

int fat(int f)
{
    int resp = 1;

    for (int i=1; i<=f; i++)
        resp = resp * i;
    return resp;
}

float series(int n)
{
    float s = 1;

    for (int i=1; i<=n; i++)
        s = s + (1.0/fat(i));

    return s;
}

int main() {
    int n;
    printf("n =");
    scanf("%d", &n);
    printf("s = %f", series(n));

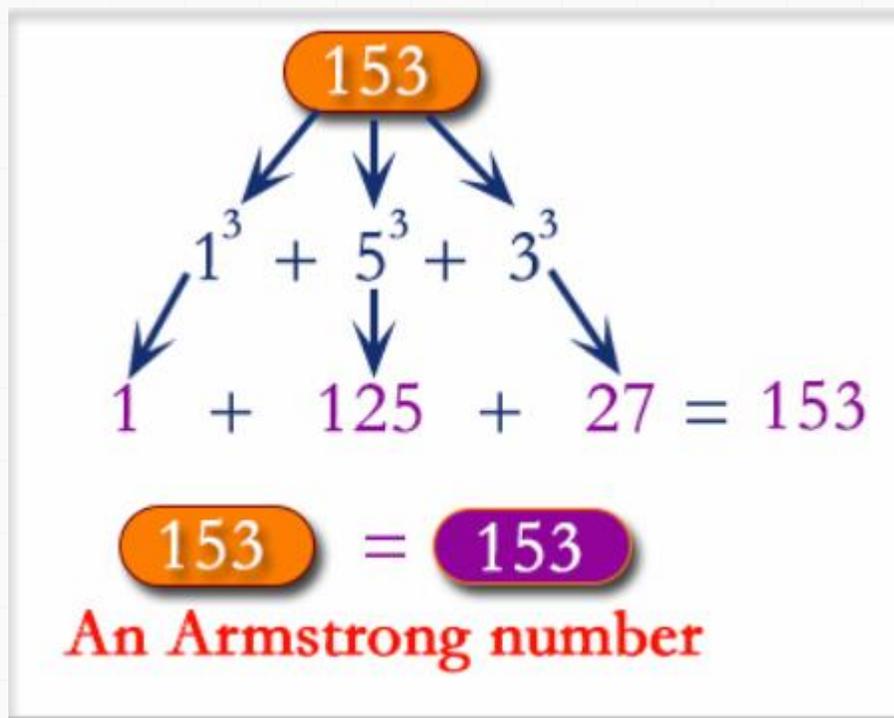
    return 0;
}
```



Funções



3) Armstrong: Faça um programa que recebe um valor inteiro de 3 dígitos e positivo $999 > n > 100$ e checa se ele é um número de Armstrong



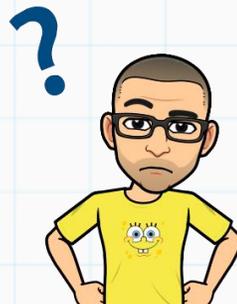
- Sabendo que dado um inteiro x , temos que:
 - $x \% 10$ retorna o último dígito de x
 - $x / 10$ retorna o número x sem o último dígito.

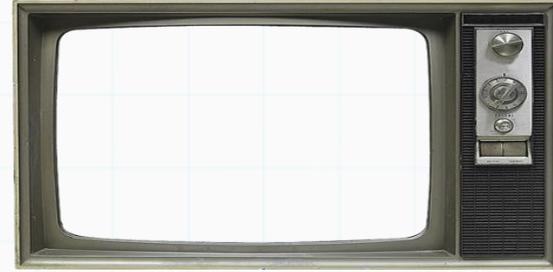
```
int soma(int A, int B)
{
    int C;
    C = A + B;
    return(C);
}
```

implemente uma função chamada

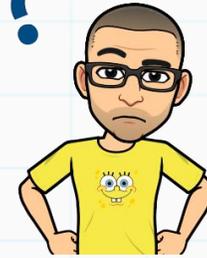
`int armstrong(int n)`

Para checar o número





Funções



```
#include <stdio.h>
int armstrong(int n1)
{
    int digito, sum, num;
    sum = 0;
    num = n1;
    while(num!=0)
    {
        digito = num % 10;
        sum += digito * digito * digito;
        num = num/10;
    }
    return (n1 == sum);
}
int main() {
    int n1;

    printf("n: ");
    scanf("%d", &n1);

    if(armstrong(n1))
        printf("%d é numero de armstrong.\n", n1);
    else
        printf("%d não é numero de armstrong\n", n1);

    return 0;
}
```



Funções



4) Subsequência: Escreva uma função que receba 2 vetores, v1 de tamanho $t1 > 0$ e v2 de tamanho $t2 > 0$, onde $t1 > t2$. A função deve retornar 1 se v2 é uma subsequência de v1, e 0 caso contrário.

```
int checa(int t1, int v1[t1], int t2, int v2[t2])
```

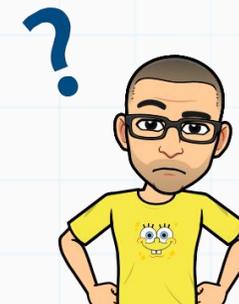
Exemplo:

v1 = { 4, -5, 23, 12, 66, 83, 90, 2} é
v2 = { 12, 66, 83}

v1 = { 4, -5, 23, 12, 66, 83, 90, 2} é
v2 = { 12}

v1 = { 4, -5, 23, 12, 66, 83, 90, 2} é
v2 = { 4, -5}

v1 = { 4, -5, 23, 12, 66, 83, 90, 2} não é
v2 = { 90, -3}



```
int soma(int A, int B)
{
    int C;
    C = A + B;
    return(C);
}
```

```
int checa(int t1, int v1[t1], int t2, int v2[t2]) {
    int i, j, achou;

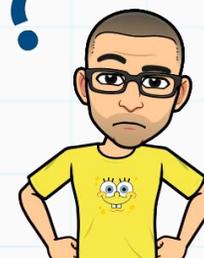
    for (i = 0; i < t1; i++)
    {
        achou = 1;
        for (j = 0; j < t2; j++)
            if ((i < t1) && (v1[i] == v2[j]))
            {
                i++;
            }
            else
            {
                achou = 0;
                break;
            }

        if (achou == 1)
            return 1;
    }
    return 0;
}
```

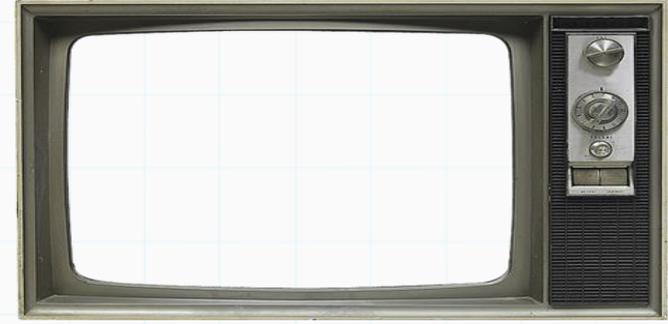


Funções

?



Até a próxima



Slides baseados no curso de Aline Nascimento

