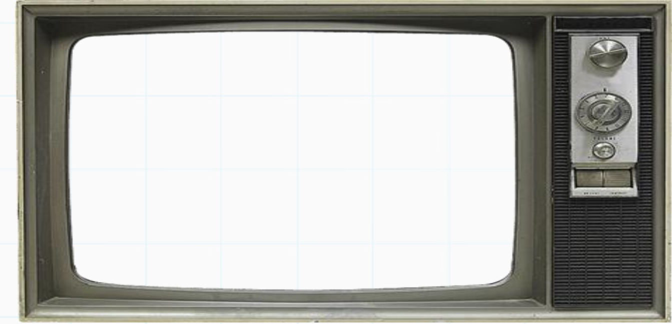
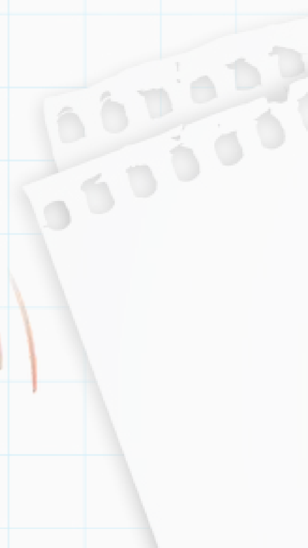
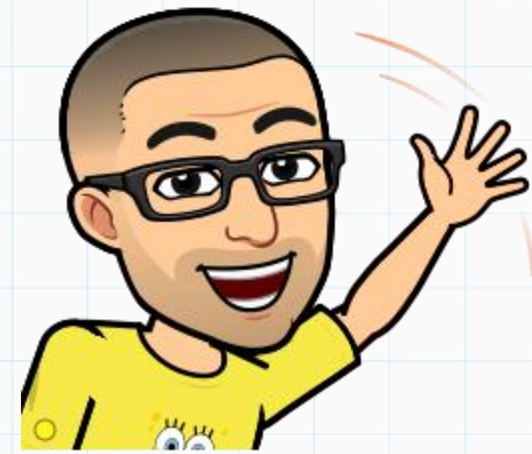
Programação Estruturada



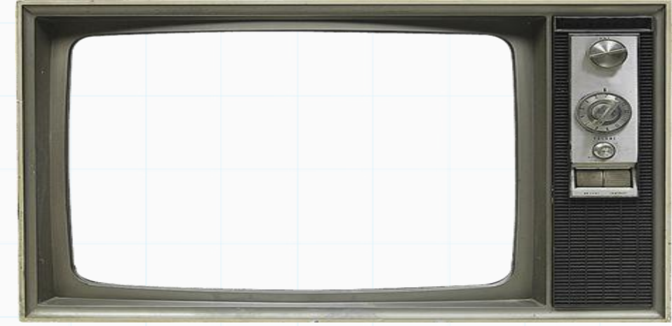
Professor : Yuri Frota

yuri@ic.uff.br





Recursividade



- Quando uma função chama a si própria direta ou indiretamente é caracterizada a recursividade.

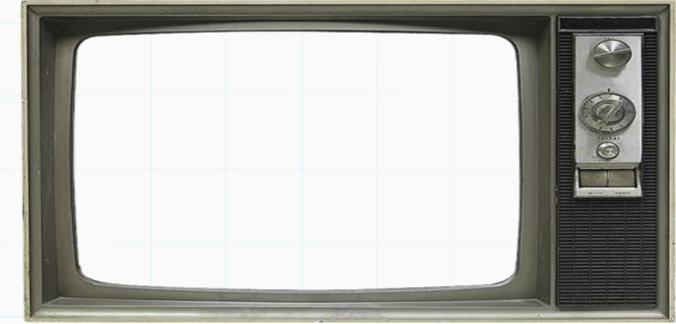
RECURSÃO DIRETA

```
int f1()
{
   f1();
}
```

RECURSÃO INDIRETA

```
int f1()
{
   f2();
}

int f2()
{
   f1();
}
```

Recursividade



- Quando uma função chama a si própria direta ou indiretamente é caracterizada a recursividade.

RECURSÃO DIRETA

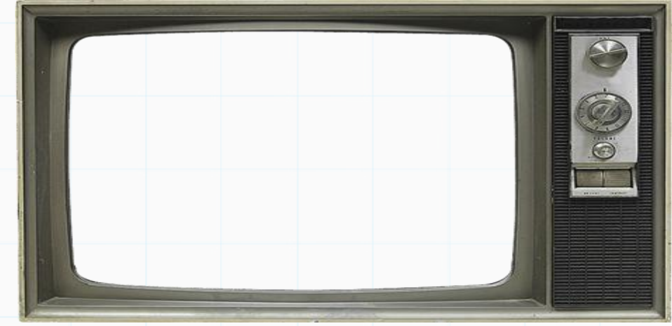
```
int f1()
{
  if () f1();
}
```

RECURSÃO INDIRETA

```
int f1()
{
  f2();
}

int f2()
{
  if () f1();
}
```

- Todo algoritmo deve ser finito
- para garantir que uma chamada recursiva não crie um loop infinito é necessário que ela esteja subordinada a uma expressão lógica que, em algum instante, se tornará falsa.
- Isto permitirá que a execução termine.

Recursividade



- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

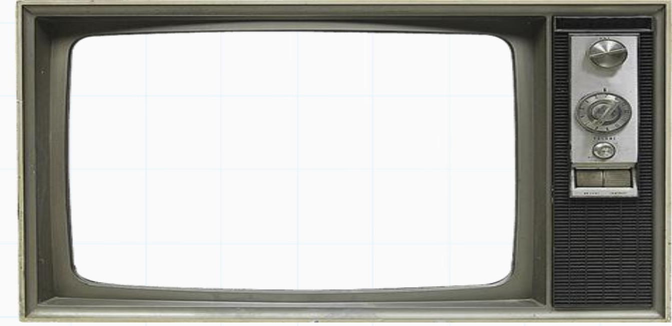
Vamos supor que o usuário digite N = 7

A resposta será: $1 \times 3 \times 5 \times 7 = 105$

```
int main (void)
{
    int num;
    scanf("%d", &num);
    printf("O produto é: %d", produto(num));
    return 0;
}
```

Vamos fazer a função produto

Recursividade



- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

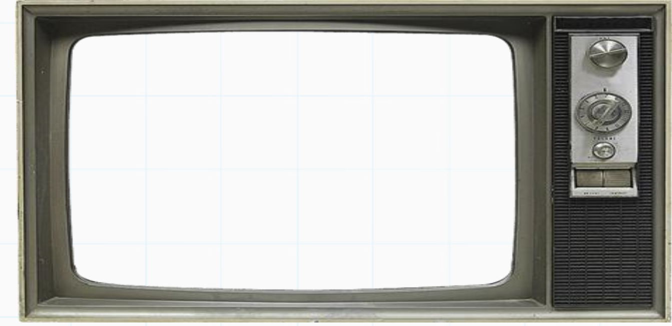
Vamos supor que o usuário digite N = 7

A resposta será: 1 × 3 × 5 × 7 = 105

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
   int num;
   scanf("%d", &num);
   printf("O produto é: %d", produto(num));
   return 0;
}
```

Vamos fazer a função produto

Recursividade

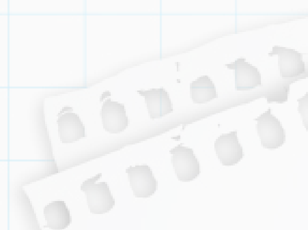


- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

Vamos supor que o usuário digite N = 7

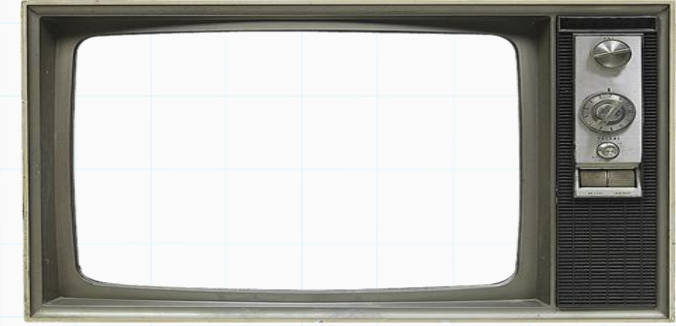
A resposta será: 1 × 3 × 5 × 7 = 105

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
   int num;
   scanf("%d", &num);
   printf("O produto é: %d", produto(num));
   return 0;
}
```



main → printf("%d",produto(7));

Recursividade

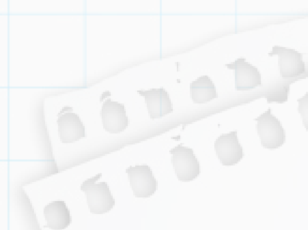


- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

Vamos supor que o usuário digite N = 7

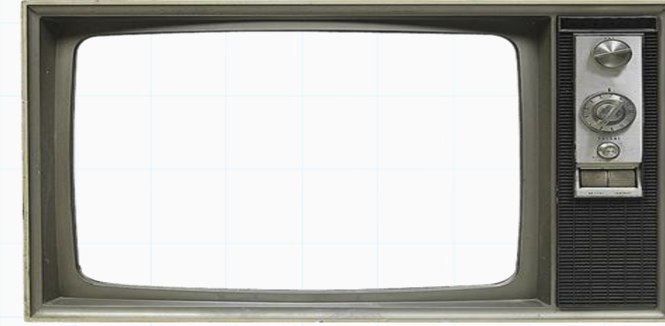
A resposta será: 1 × 3 × 5 × 7 = 105

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
   int num;
   scanf("%d", &num);
   printf("O produto é: %d", produto(num));
   return 0;
}
```



produto(7) → return(produto(5)*7);
main → printf("%d",produto(7));

Recursividade



- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

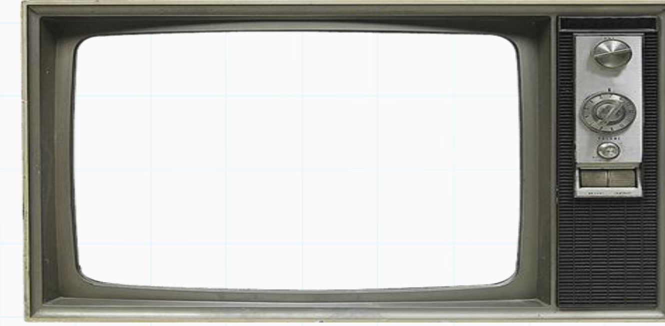
Vamos supor que o usuário digite N = 7

A resposta será: 1 × 3 × 5 × 7 = 105

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
   int num;
   scanf("%d", &num);
   printf("O produto é: %d", produto(num));
   return 0;
}
```

produto(5) → return(produto(3)*5);
produto(7) → return(produto(5)*7);
main → printf("%d",produto(7));

Recursividade



- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

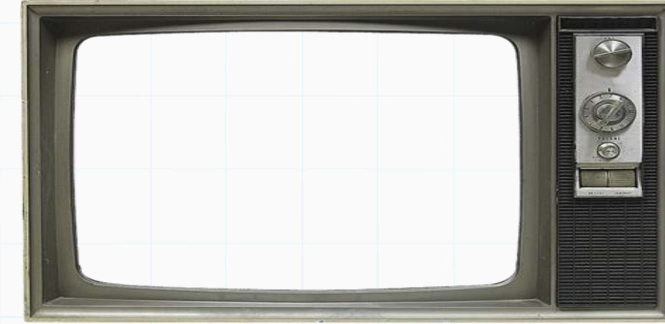
Vamos supor que o usuário digite N = 7

A resposta será: 1 × 3 × 5 × 7 = 105

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
    int num;
    scanf("%d", &num);
    printf("O produto é: %d", produto(num));
    return 0;
}
```

produto(3) → return(produto(1)*3);
produto(5) → return(produto(3)*5);
produto(7) → return(produto(5)*7);
main → printf("%d",produto(7));

Recursividade

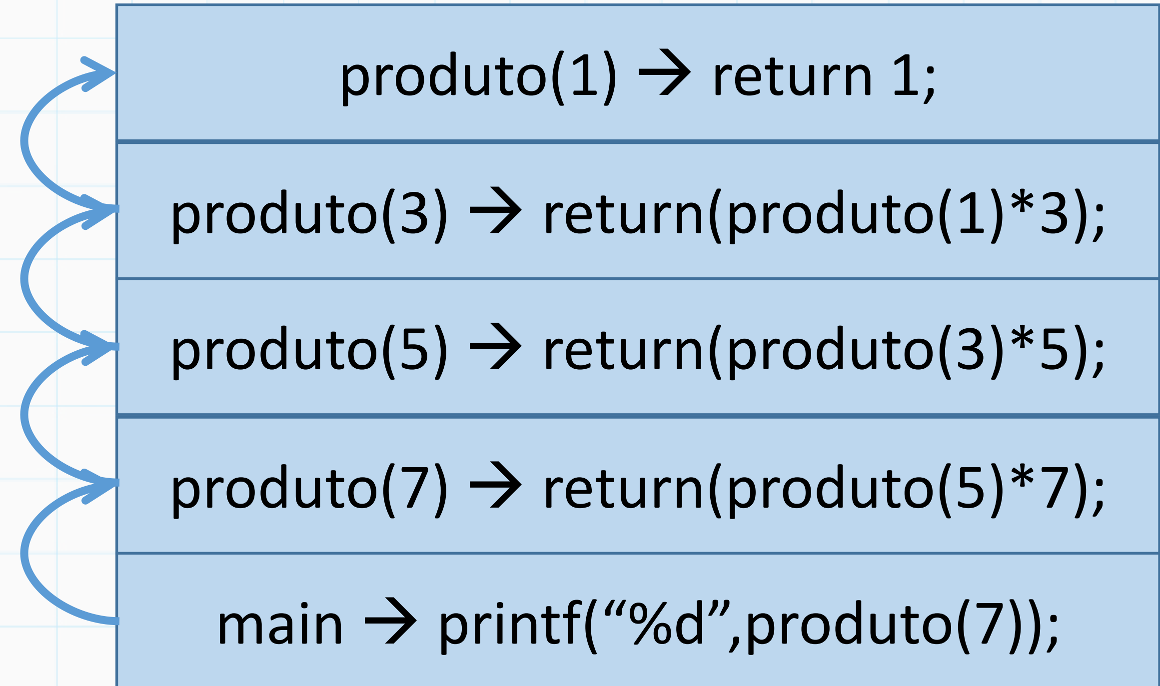


- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

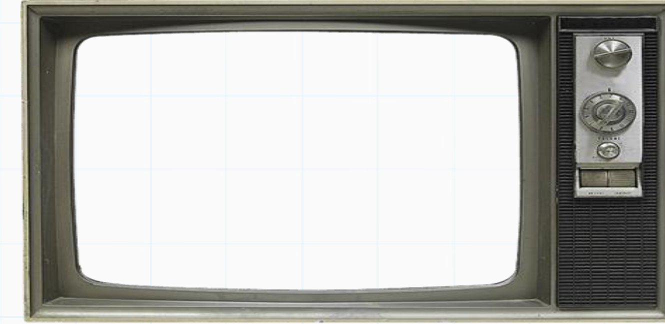
Vamos supor que o usuário digite N = 7

A resposta será: 1 × 3 × 5 × 7 = 105

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
    int num;
    scanf("%d", &num);
    printf("O produto é: %d", produto(num));
    return 0;
}
```



Recursividade

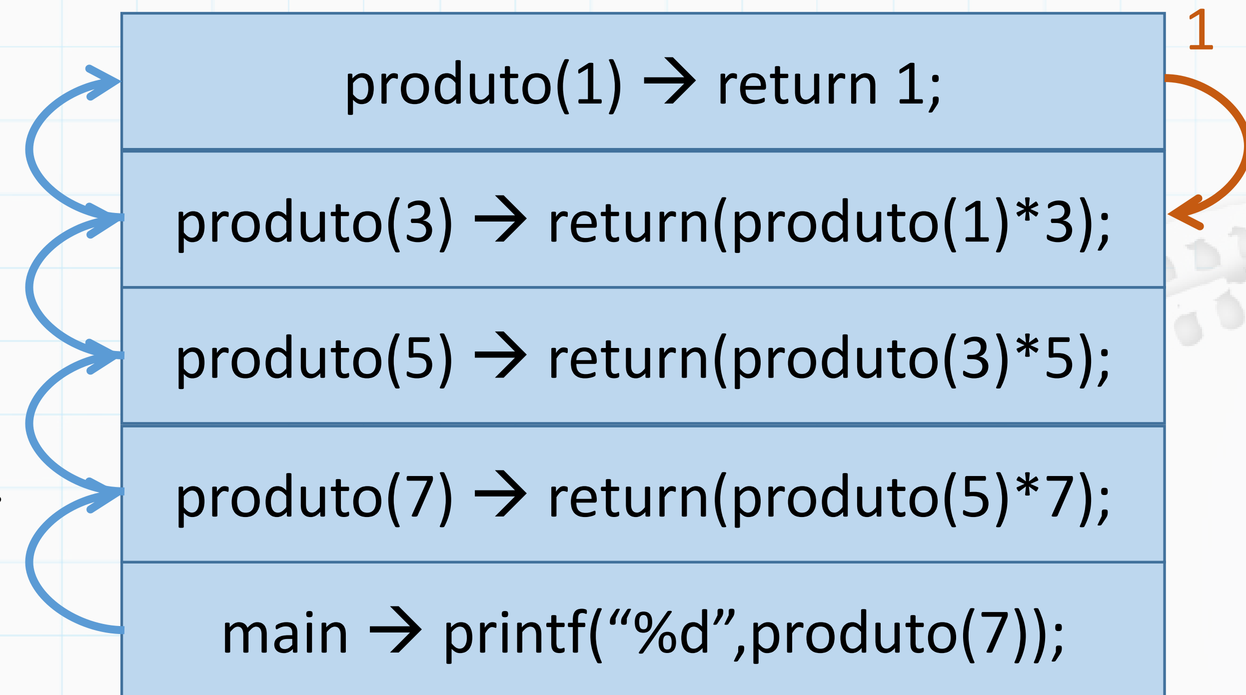


- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

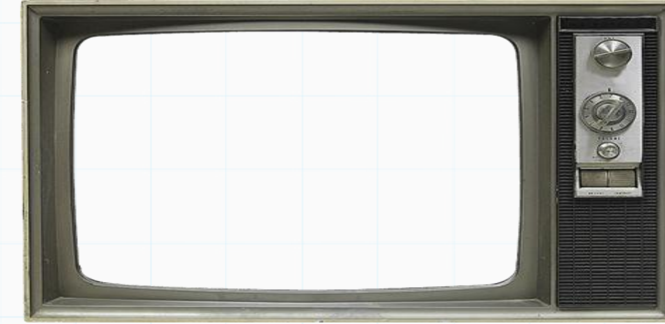
Vamos supor que o usuário digite N = 7

A resposta será: 1 × 3 × 5 × 7 = 105

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
    int num;
    scanf("%d", &num);
    printf("O produto é: %d", produto(num));
    return 0;
}
```



Recursividade

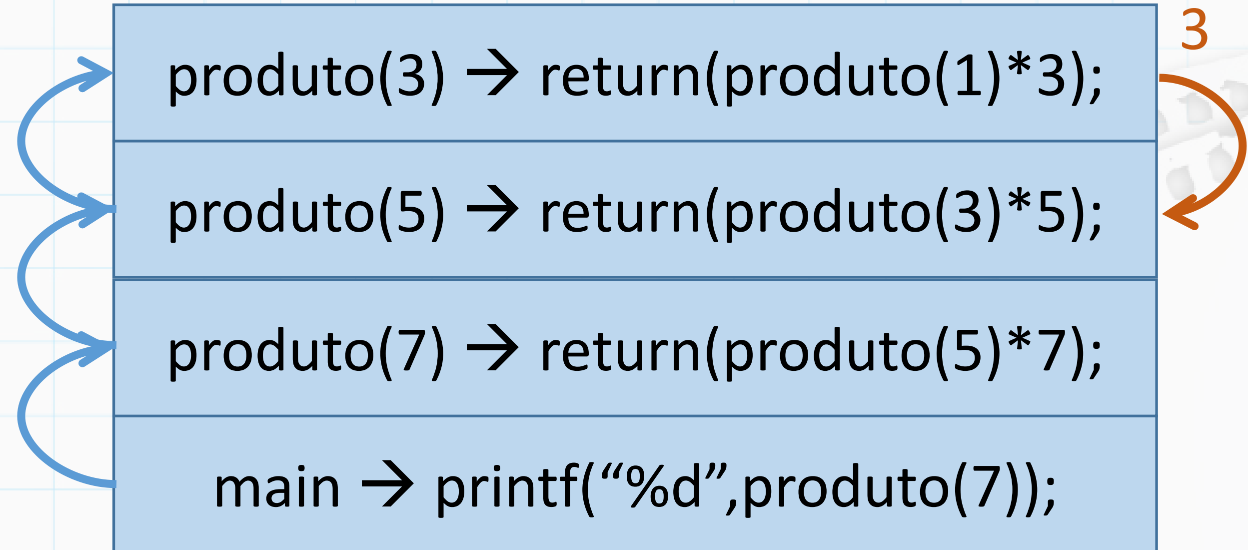


- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

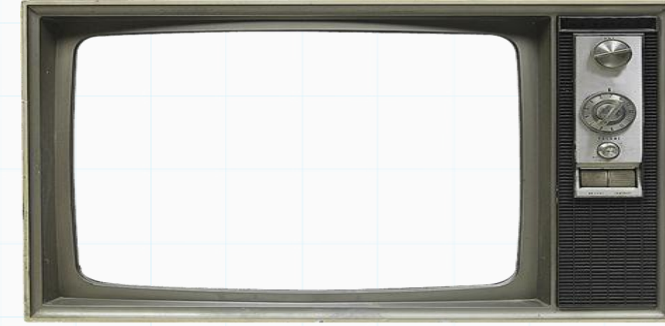
Vamos supor que o usuário digite N = 7

A resposta será: 1 × 3 × 5 × 7 = 105

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
    int num;
    scanf("%d", &num);
    printf("O produto é: %d", produto(num));
    return 0;
}
```



Recursividade

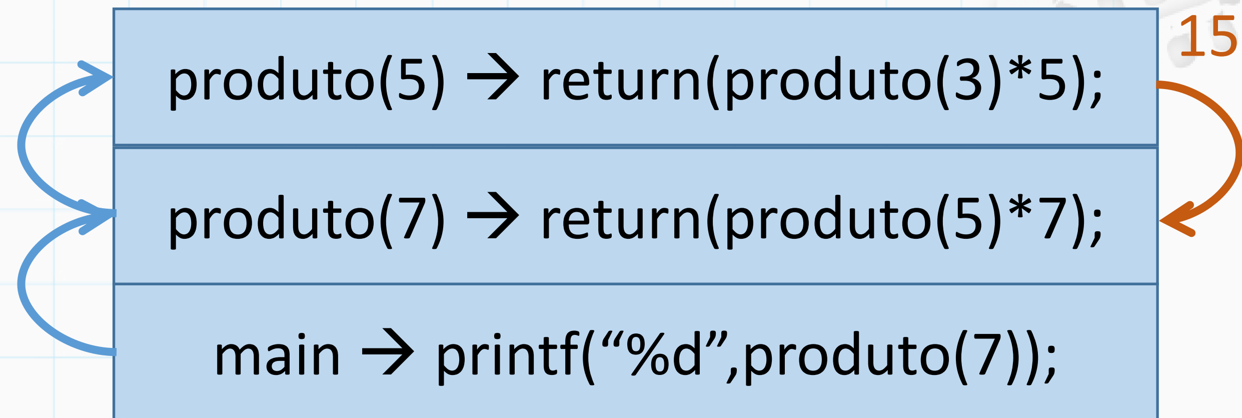


- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

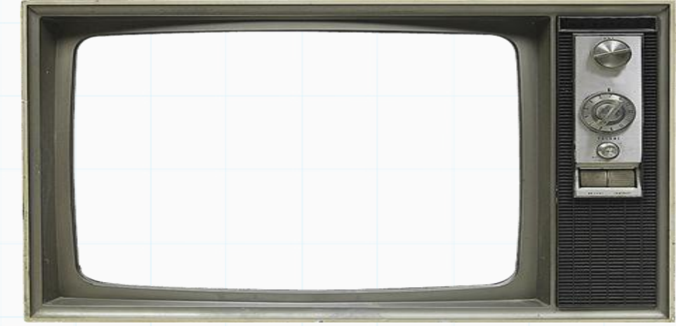
Vamos supor que o usuário digite N = 7

A resposta será: 1 × 3 × 5 × 7 = 105

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
    int num;
    scanf("%d", &num);
    printf("O produto é: %d", produto(num));
    return 0;
}
```



Recursividade

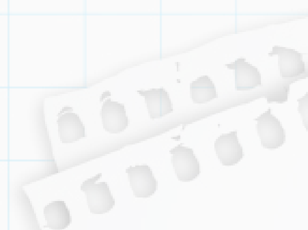


- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

Vamos supor que o usuário digite N = 7

A resposta será: 1 × 3 × 5 × 7 = 105

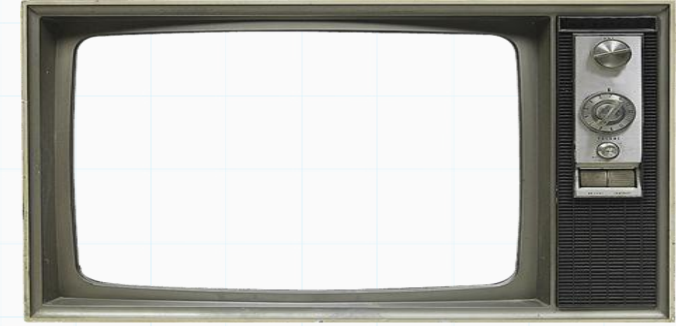
```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
    int num;
    scanf("%d", &num);
    printf("O produto é: %d", produto(num));
    return 0;
}
```



produto(7) → return(produto(5)*7);
main → printf("%d",produto(7));

105

Recursividade

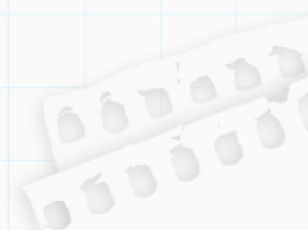


- Exemplo: Dado um número N impar, como calcular o produto dos números inteiros positivos ímpares menores ou iguais a N de forma recursiva.

Vamos supor que o usuário digite N = 7

A resposta será: 1 × 3 × 5 × 7 = 105

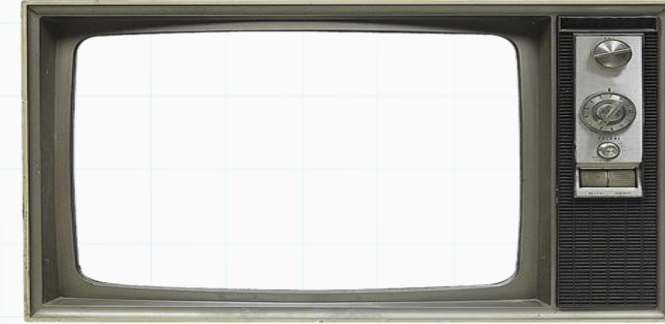
```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
int main (void)
{
    int num;
    scanf("%d", &num);
    printf("O produto é: %d", produto(num));
    return 0;
}
```



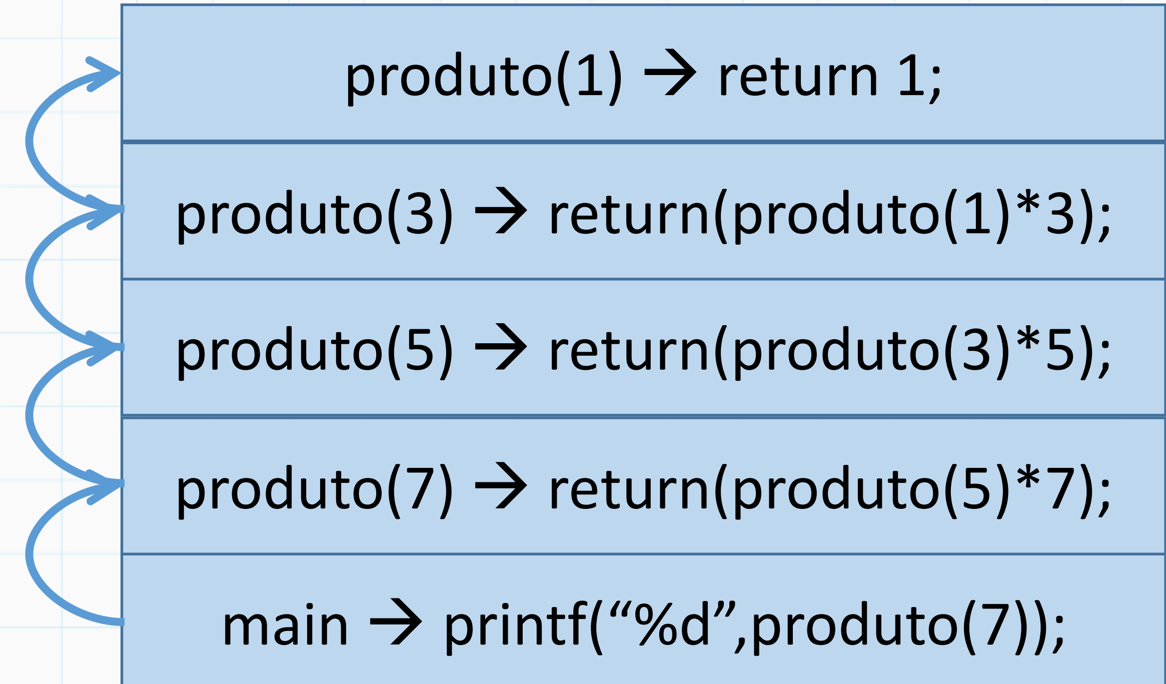
main → printf("%d",105);

Recursividade

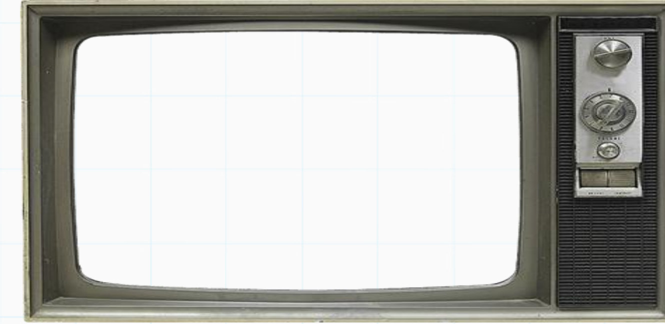
- Quando uma função é chamada recursivamente, cria-se um ambiente local para cada chamada (empilhamento) onde:



```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
```



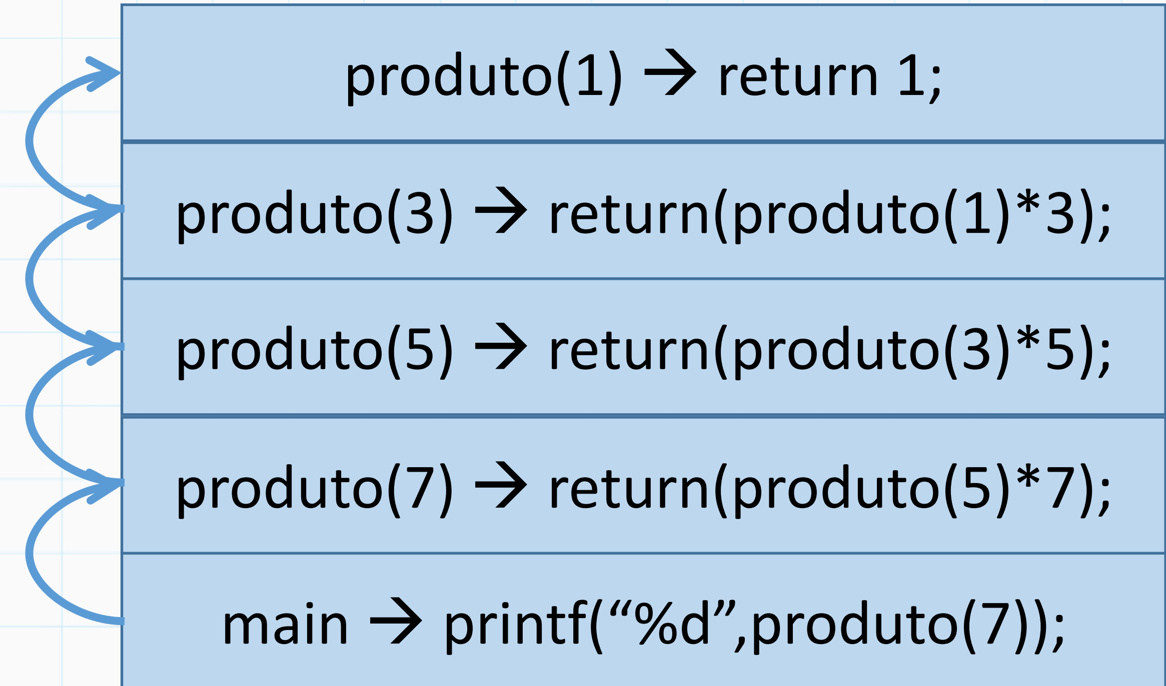
Recursividade



- Quando uma função é chamada recursivamente, cria-se um ambiente local para cada chamada (empilhamento) onde:

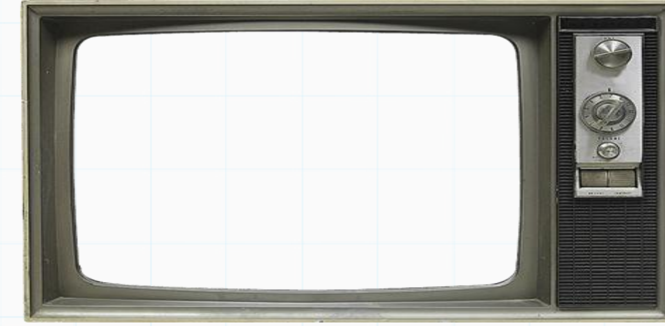
  - As variáveis locais de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes.

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
```



Recursividade

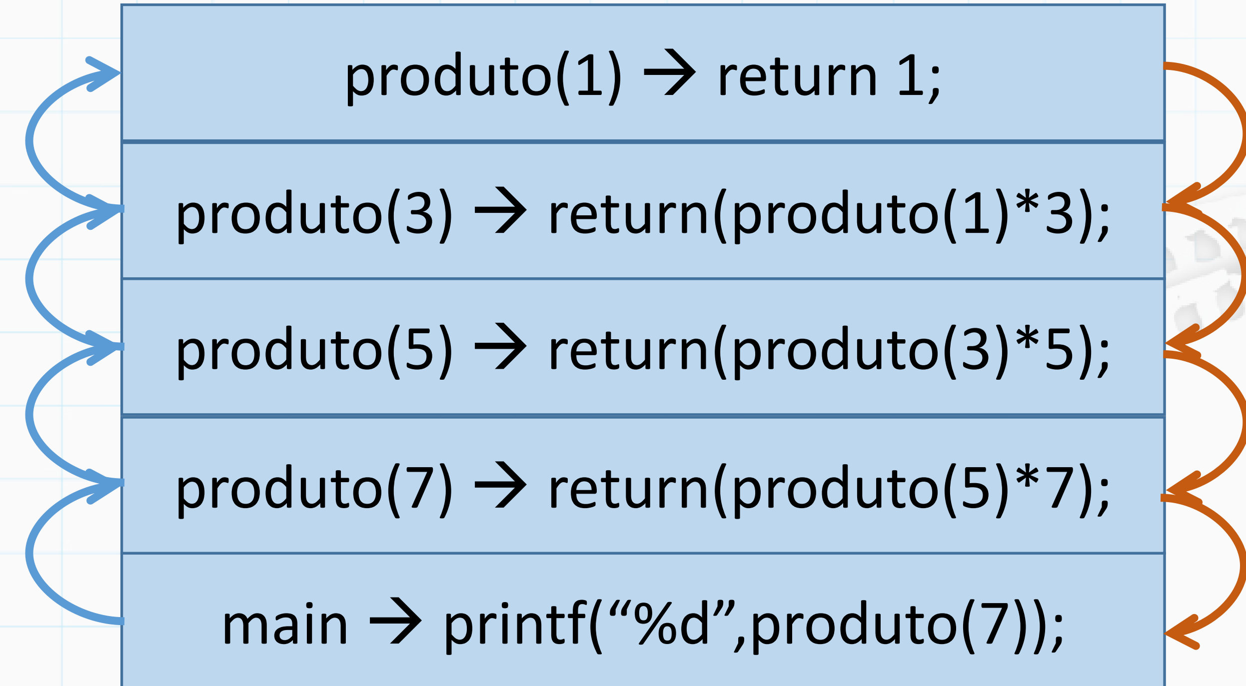
- Quando uma função é chamada recursivamente, cria-se um ambiente local para cada chamada (empilhamento) onde:



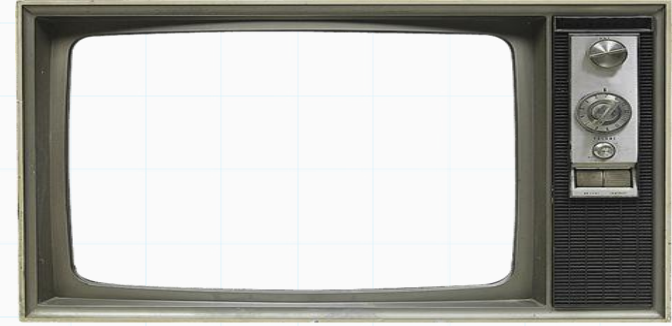
- As variáveis locais de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes.

- Ao retornarem, esses ambientes são destruídos (desempilhamento).

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
```



Recursividade

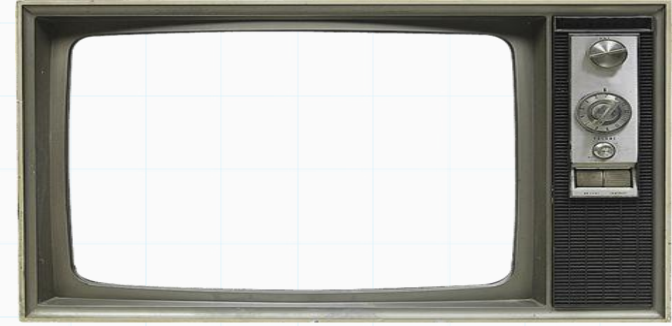


- As funções recursivas são geralmente baseadas em equações de recorrência que definem a recursão. Toda recorrência precisa ter:

  - Caso Base: é dada por definição, não precisa de recursividade para ser resolvida

  - Recorrência: será parte do problema, cuja solução depende da solução do mesmo problema, só que para um caso mais simples ou reduzido.

Recursividade



- As funções recursivas são geralmente baseadas em equações de recorrência que definem a recursão. Toda recorrência precisa ter:

  - Caso Base: é dada por definição, não precisa de recursividade para ser resolvida

  - Recorrência: será parte do problema, cuja solução depende da solução do mesmo problema, só que para um caso mais simples ou reduzido.

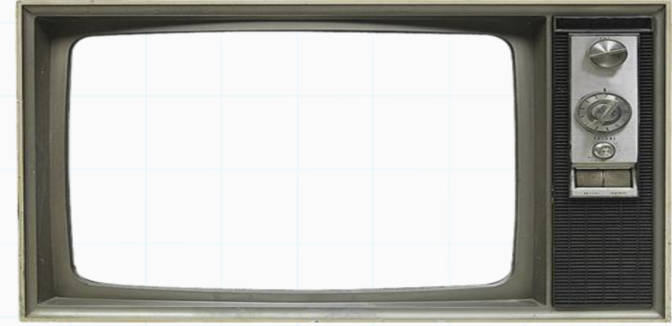
No exemplo anterior tínhamos:

- Caso Base: 1 se N=1

- Recorrência: f(N) = N * f(N-2) se N > 1

```
int produto (int N) {
        if  (N == 1)
            return 1;
        else
            return (produto(N-2) * N);
}
```

Recursividade



- As funções recursivas são geralmente baseadas em equações de recorrência que definem a recursão. Toda recorrência precisa ter:

  - Caso Base: é dada por definição, não precisa de recursividade para ser resolvida

  - Recorrência: será parte do problema, cuja solução depende da solução do mesmo problema, só que para um caso mais simples ou reduzido.

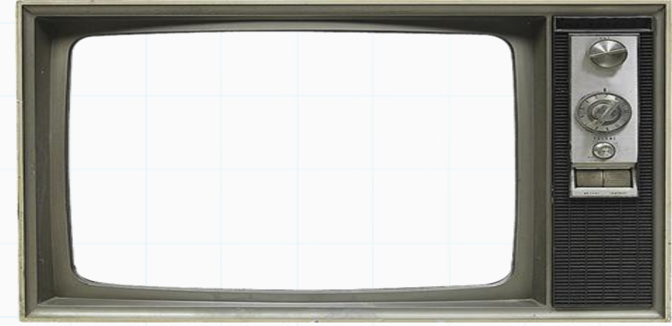
E se fosse fatorial, teríamos:

- Caso Base: ?

- Recorrência: ?

```
int fat (int N) {

            ?

}
```

Recursividade



- As funções recursivas são geralmente baseadas em equações de recorrência que definem a recursão. Toda recorrência precisa ter:

  - Caso Base: é dada por definição, não precisa de recursividade para ser resolvida

  - Recorrência: será parte do problema, cuja solução depende da solução do mesmo problema, só que para um caso mais simples ou reduzido.

E se fosse fatorial, teríamos:

- Caso Base: 1 se N=1

- Recorrência: f(N) = N * f(N-1) se N > 1

```
int fat (int N) {
    if  (N == 1)
        return 1;
    else
        return (fat(N-1) * N);
}
```

Recursividade

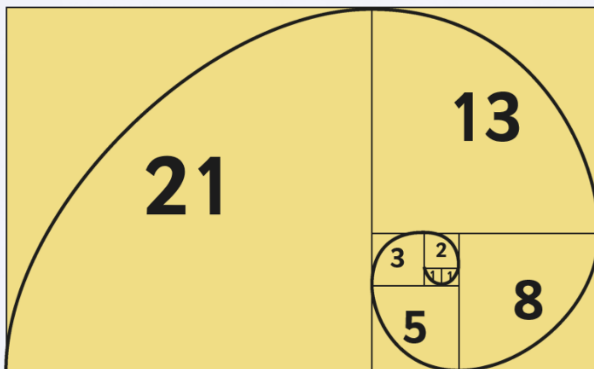
1) Fibonacci: Escreva um programa que dado um valor positivo inteiro N, imprima os primeiros N termos da série de Fibonacci. O programa deve usar uma função recursiva que dado um inteiro i, retorne o i-ésimo termo da sequencia.

int fib(int i)

Use só o que aprendemos até hoje

Sabendo que a série começa com 0 e 1, e próximo é a soma dos dois anteriores:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, ...

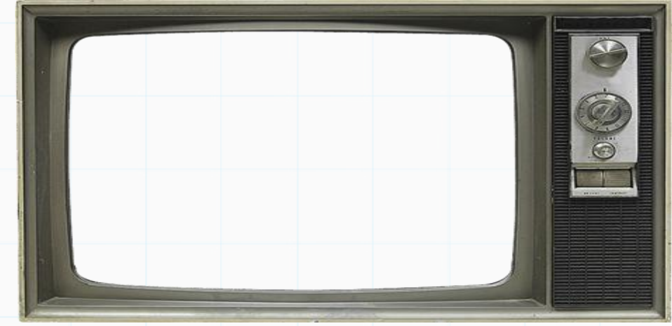


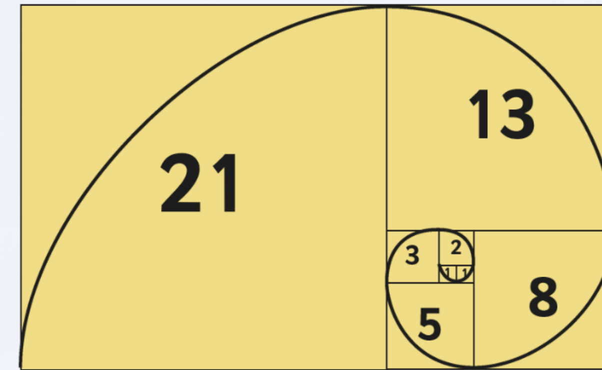
Fazer o programa principal e a função

Lembre-se: A equação de recorrência de Fibonacci é : fib(i) = fib(i-1) + fib(i-2)

Recursividade

```c
#include<stdio.h>

int fib(int n)  {
    if (n == 1)
        return 0;
    if (n == 2)
        return 1;
    if (n > 2)
        return fib(n-1) + fib(n-2);
}

int main()
{
    int n;

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

    for (int i=1; i<=n; i++)
     printf("%d, ",fib(i));

    return 0;
}
```

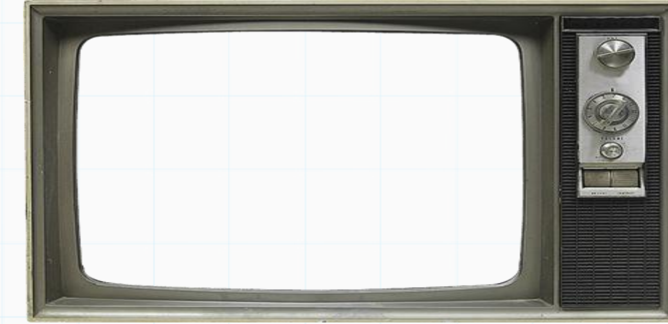




Recursividade

2) Maior: Escreva um programa que dado um vetor de inteiros de tamanho N, encontre e imprima o seu maior valor.

Use só o que aprendemos até hoje



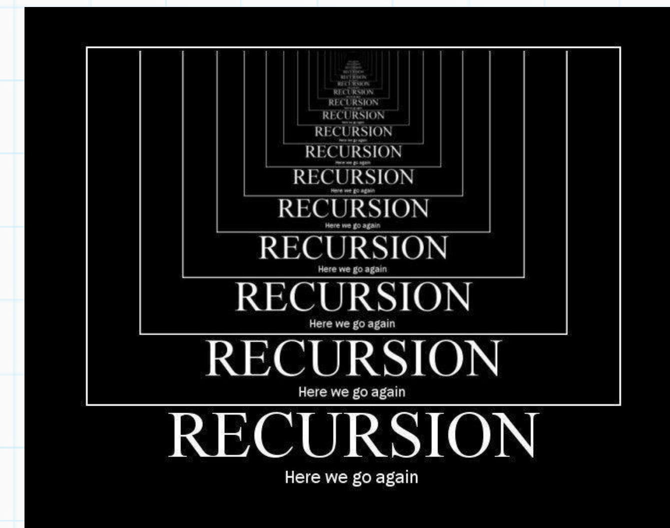
programa deve usar uma função recursiva:

int maior(int V[], int ini, int fim)

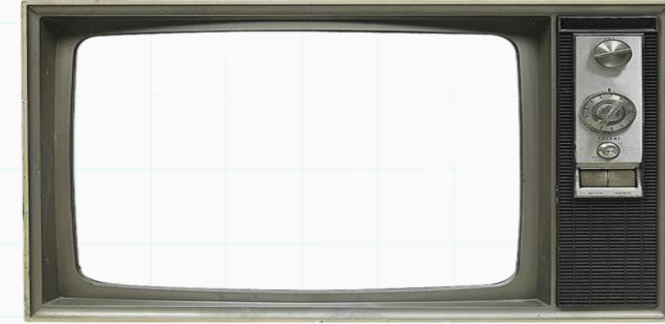
Fazer o programa principal e a função

para encontrar o maior valor de um vetor V que começa em ini e termina em fim, sabendo que a equação de recorrência é:

maior(V, ini, fim) = maior número entre V[ini] E maior(V, ini+1, fim)



Recursividade



2) Maior: Escreva um programa que dado um vetor de inteiros de tamanho N, encontre e imprima o seu maior valor.

Use só o que aprendemos até hoje

programa deve usar uma função recursiva:

Fazer o programa principal e a função

int maior(int V[], int ini, int fim)

para encontrar o maior valor de um vetor V que começa em ini e termina em fim, sabendo que a equação de recorrência é:

maior(V, ini, fim) = maior número entre V[ini] E maior(V, ini+1, fim)

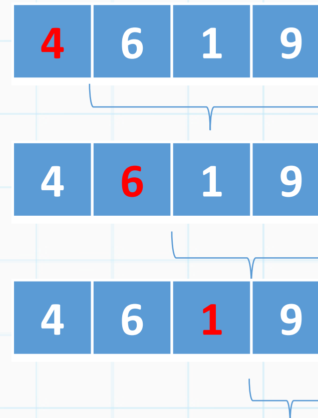
Exemplo:

- para N=4 e V = [4,6,1,9]
- a chamada recursiva inicial (na função main()) deverá ser maior(V,0,3), logo:

maior(V,0,3) = maior número entre V[0] e maior(V,1,3)

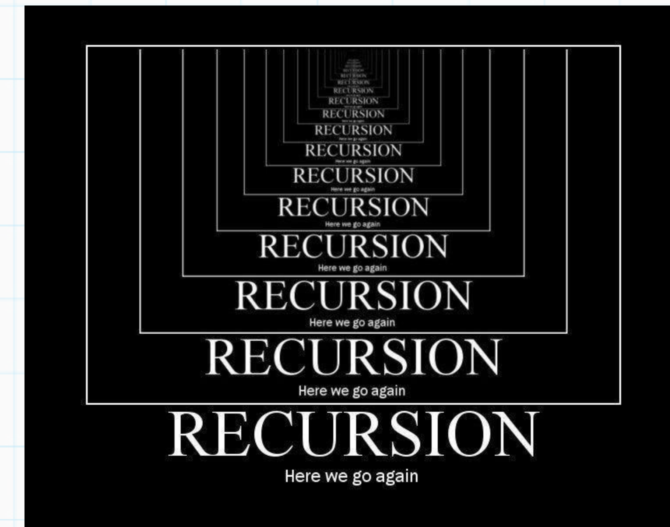
maior(V,1,3) = maior número entre V[1] e maior(V,2,3)

maior(V,2,3) = maior número entre V[2] e maior(V,3,3)



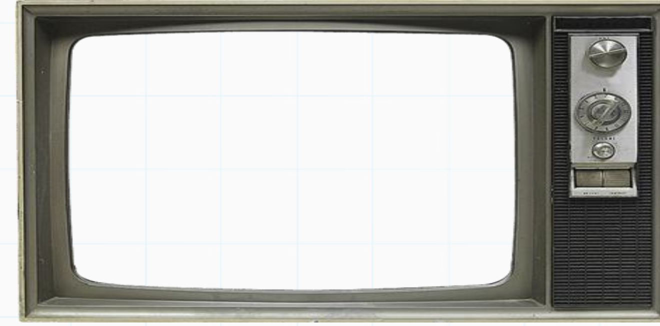
maior(V,3,3) = V[3] (caso base)

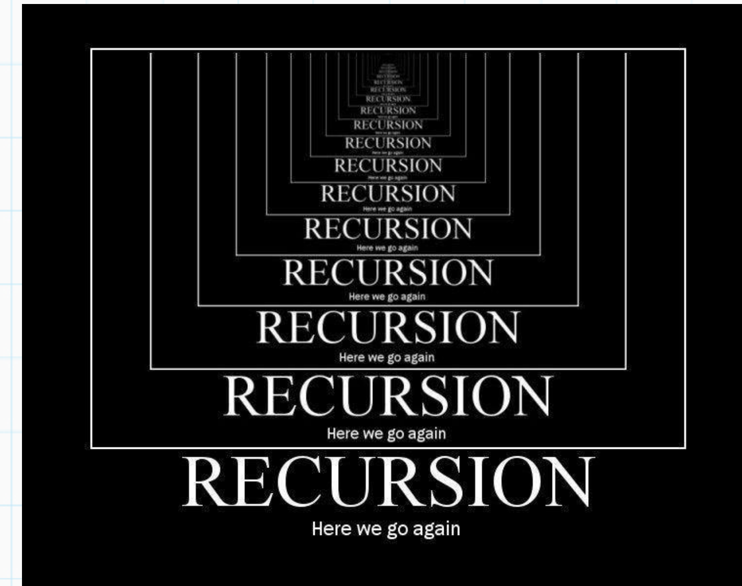
4	6	1	9



Recursividade

```c
#include <stdio.h>

int maior(int A[], int i, int n)
{
    if (i == n)
        return A[i];

    if (A[i] > maior(A, i+1, n))
        return A[i];
    else
        return maior(A, i+1, n);
}

int main()
{
    int v[] = {10, 324, 45, 90, 980};
    int n = 5;
    printf("maior %d", maior(v, 0, n-1));
    return 0;
}
```





Recursividade

Use só o que aprendemos até hoje

3) MMC: Dados dois números positivos, calcular o MMC (Mínimo múltiplo comum) deles de forma recursiva.

- O mínimo múltiplo comum (MMC) corresponde ao menor número inteiro positivo, diferente de zero, que é múltiplo ao mesmo tempo dos dois números.

MMC

Fazer o programa principal e a função

Exemplo

Múltiplos de 12 = {0, 12, 24, 36, 48, 60, 72, 84, 96...}
Múltiplos de 14 = {0, 14, 28, 42, 56, 70, 84, 98, 112...}

logo, MMC(12, 14) = 84

Recursividade

Use só o que aprendemos até hoje

3) MMC: Dados dois números positivos, calcular o MMC (Mínimo múltiplo comum) deles de forma recursiva.

- O mínimo múltiplo comum (MMC) corresponde ao menor número inteiro positivo, diferente de zero, que é múltiplo ao mesmo tempo dos dois números.

MMC

Fazer o programa principal e a função

Exemplo

Múltiplos de 12 = {0, 12, 24, 36, 48, 60, 72, 84, 96...}
Múltiplos de 14 = {0, 14, 28, 42, 56, 70, 84, 98, 112...}

logo, MMC(12, 14) = 84

Dica: a ideia é começar de 1 e vai testando os possíveis candidatos a MMC até encontrar, de forma recursiva.
Exemplo : Dado números N1 e N2

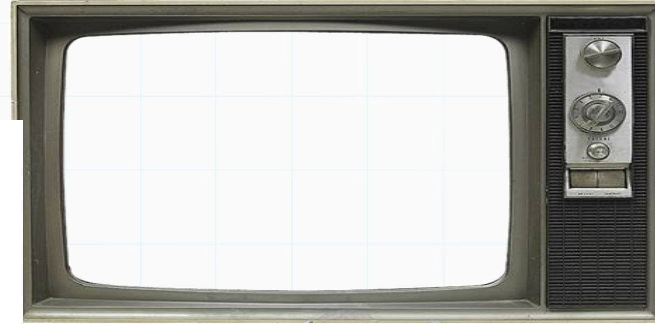
- se 1 é múltiplo de N1 e N2 então retorna 1, senão retorna chamada recursiva para testar 2
- se 2 é múltiplo de N1 e N2 então retorna 1, senão retorna chamada recursiva para testar 3
- se 3 é múltiplo de N1 e N2 então retorna 1, senão retorna chamada recursiva para testar 4

....

....

....

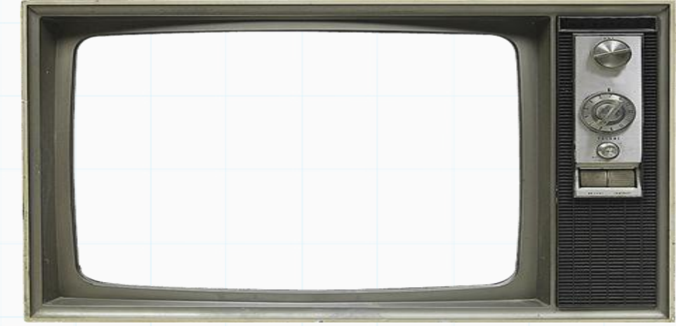
Recursividade



```c
#include <stdio.h>

int mmc(int n1, int n2, int cand)
{
    if((cand % n1 == 0) && (cand % n2 == 0))
        return cand;
    else
        return mmc(n1, n2, cand+1);
}

int main()
{
    int n1, n2;
    printf("n1:");
    scanf("%d", &n1);
    printf("n2:");
    scanf("%d", &n2);

    printf("O MMC de %d e %d eh %d\n\n", n1, n2, mmc(n1,n2,1));
    return 0;
}
```

MMC

Recursividade



4) Primo: Escreva um programa que dado um número inteiro positivo n>0, diga se ele é primo. O programa deve usar uma função recursiva para determinar a propriedade, baseada na seguinte recorrência:

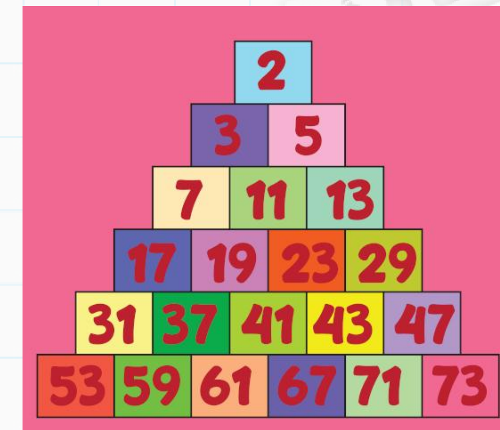
primo(n, d) =
- verdade, se d=n
- = {(n não for divisível por d é verdade ?) E primo(n,d+1)} ,se d<n

onde n é o número de queremos saber se é primo e d é um possível divisor. Com essa recorrência, partindo de

primo(n, 2)

Podemos determinar se o número é primo ou não.

Fazer o programa principal e a função



Recursividade

5) Maior: Faça um programa que receba uma matriz A nxm e identifique qual o maior elemento da matriz. O programa deve usar uma função recursiva para encontrar o maior elemento.

Use só o que aprendemos até hoje

programa deve usar uma função recursiva:

Fazer o programa principal e a função

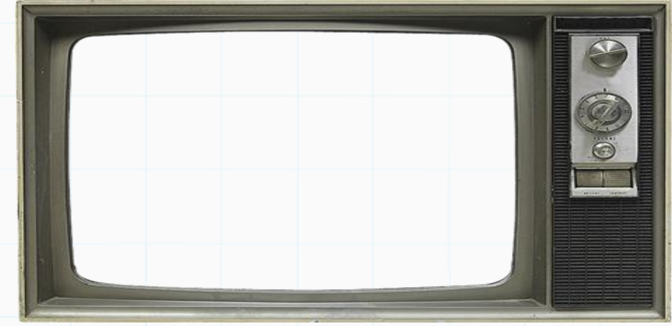
```
int maiorRec(int A[N][M], int i, int j)
```

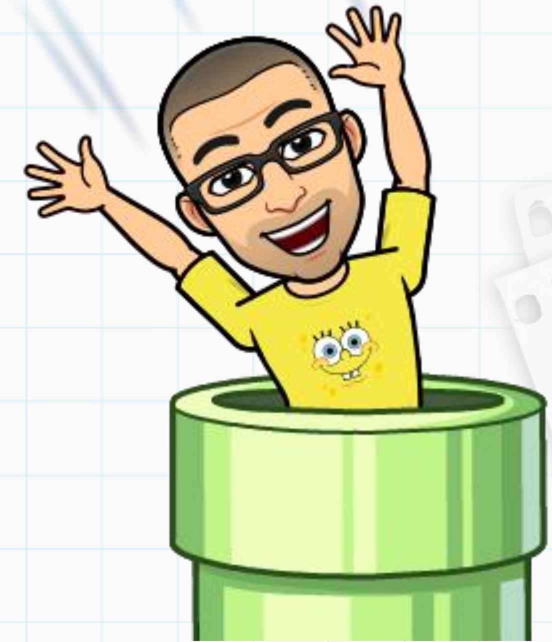
Como fazer ?

RECURSION

Here we go again

Até a próxima





Slides baseados no curso de Aline Nascimento