in4mation HUB

Fundamentos de Programação

Técnicas de Modularização - Procedimentos e Funções - Recursividade

in4mation HUB

CONCEITO

Técnica de programação, em que uma função é capaz de chamar a si mesma repetidamente para resolver um problema.

in4mation HUB

CASOS BASE E CASOS RECURSIVOS

Caso base:

- É a condição de parada da recursão. Quando essa condição é atendida, a função para de se chamar a si mesma e retorna um valor específico. É fundamental ter um caso base para evitar um loop infinito.

Caso recursivo:

- Parte da função que faz a chamada recursiva, ou seja, a função chama a si mesma com um problema menor ou mais simples. Essa chamada ocorre repetidamente até que o caso base seja alcançado.

in4mation HUB

EXEMPLO:

COM WHILE

```c
#include <stdio.h>

int main() {
    int i = 1;
    while (i <= 5) {
        printf("%d - oi\n", i);
        i++; // Incrementa o índice
    }
    return 0;
}
```

```
1 - oi
2 - oi
3 - oi
4 - oi
5 - oi
```

COM RECURSIVIDADE

```c
#include <stdio.h>

void imprimirOi(int n) {
    if (n == 0) {
        return;  // Caso base
    } else {
        imprimirOi(n - 1);
        printf("%d - oi\n", n);
    }
}

int main() {
    int n = 5;
    imprimirOi(n);
    return 0;
}
```

```
1 - oi
2 - oi
3 - oi
4 - oi
5 - oi
```

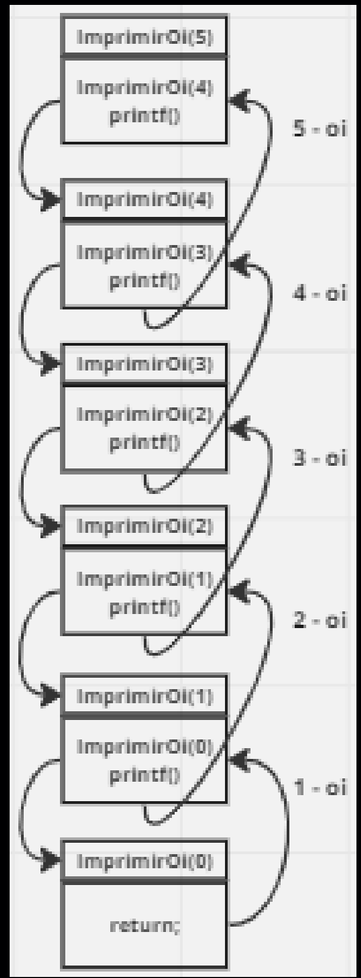
in4mation HUB

EXEMPLO:

```c
#include <stdio.h>

void imprimirOi(int n) {
    if (n == 0) {
        return;  // Caso base
    } else {
        imprimirOi(n - 1);
        printf("%d - oi\n", n);
    }
}

int main() {
    int n = 5;
    imprimirOi(n);
    return 0;
}
```




```
1 - oi
2 - oi
3 - oi
4 - oi
5 - oi
```

in4mation HUB

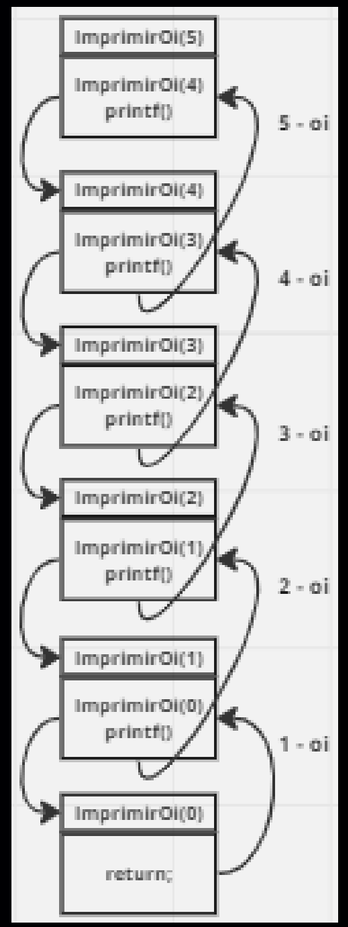
POSIÇÃO DA CHAMADA RECURSIVA:

```c
#include <stdio.h>

void imprimirOi(int n) {
    if (n == 0) {
        return;  // Caso base
    } else {
        imprimirOi(n - 1);
        printf("%d - oi\n", n);
    }
}

int main() {
    int n = 5;
    imprimirOi(n);
    return 0;
}
```

```
1 - oi
2 - oi
3 - oi
4 - oi
5 - oi
```

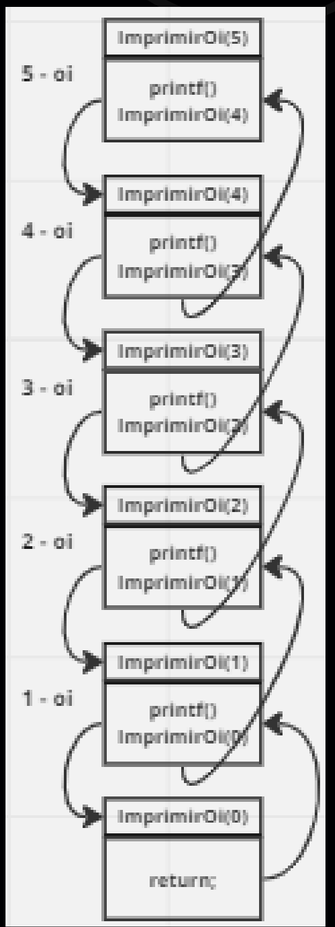



```c
#include <stdio.h>

void imprimirOi(int n) {
    if (n == 0) {
        return;  // Caso base
    } else {
        printf("%d - oi\n", n);
        imprimirOi(n - 1);
    }
}

int main() {
    int n = 5;
    imprimirOi(n);
    return 0;
}
```

```
5 - oi
4 - oi
3 - oi
2 - oi
1 - oi
```



in4mation HUB

VANTAGENS E DESVANTAGENS

Vantagens:

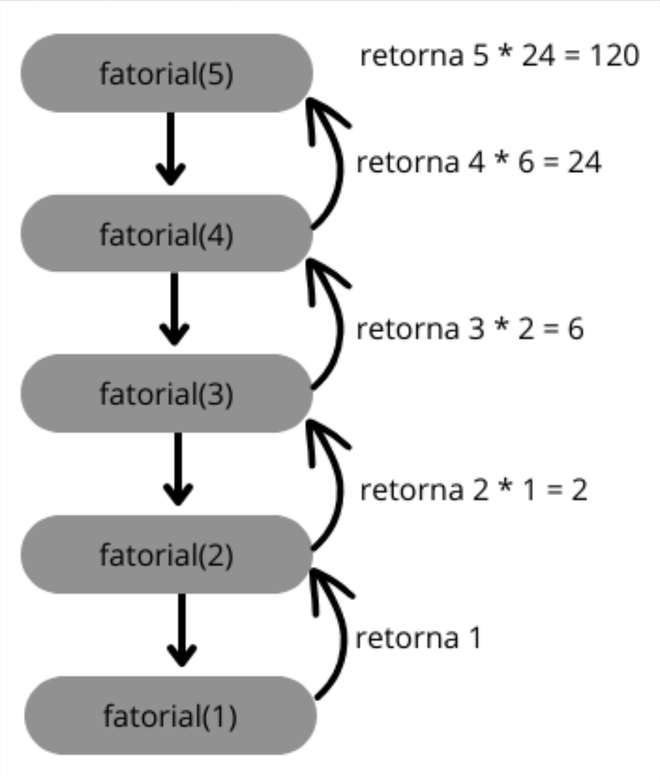
- **Simplicidade: Código mais limpo e fácil de entender para problemas que se dividem naturalmente.**
- **Natural para Estruturas Recursivas: Árvores e grafos.**

Desvantagens:

- **Consumo de Memória: Cada chamada recursiva consome memória da pilha.**
- **Desempenho: Pode ser menos eficiente em alguns casos devido ao overhead das chamadas de função.**

in4mation HUB

EXEMPLO: CÁLCULO DE FATORIAL




```c
#include <stdio.h>

// Função recursiva para calcular o fatorial de um número
int fatorial(int n) {
    // Caso base: fatorial de 0 ou 1 é 1
    if (n == 0 || n == 1) {
        return 1;
    }
    // Caso recursivo: n * fatorial(n - 1)
    else {
        return n * fatorial(n - 1);
    }
}

int main() {
    int numero = 5;
    int resultado = fatorial(numero);

    printf("O fatorial de %d é: %d\n", numero, resultado);

    return 0;
}

```

in4mation HUB

THAT'S ALL