Laboratório de Fundamentos em TIC

Controles de Execução e Recursividade

Prof. Gabriel Resende Machado

gabrielmachado@unifeso.edu.br

https://www.linkedin.com/in/machadogabriel

https://github.com/UNIFESO-Gabriel/fundamentos-em-tic

Relembrando: Escopo de Variáveis

- O escopo de uma variável define onde ela pode ser utilizada em um programa.

```c
#include <stdio.h>

void imprime_estrelas(int qtd) {
    int i = 0;
    while (i < qtd) {
        printf("*");
        i++;
    }
    printf("\n");
}

int main() {
    int i = 0, qtd = 5;
    while (i < qtd) {
        imprime_estrelas(qtd);
        i++;
    }
    return 0;
}
```

Variável local

Variáveis locais

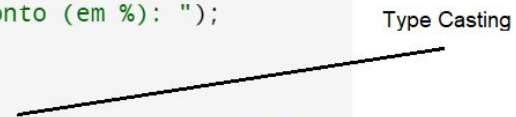
```c
#include <stdio.h>

int i;

void imprime_estrelas(int qtd) {
    i = 0;
    while (i < qtd) {
        printf("*"); i++;
    }
    printf("\n");
}

int main() {
    i = 0;
    int qtd = 5;
    while (i < qtd) {
        imprime_estrelas(qtd); i++;
    }
    return 0;
}
```

Variável global

Relembrando: *Type Casting*

- *Type Casting* é o nome dado ao procedimento de converter o tipo de uma variável em outro.

```c
#include <stdio.h>

int main() {

    float valor_inicial, valor_final;
    int desconto;

    printf("Digite o valor inicial do produto (R$): ");
    scanf("%f", &valor_inicial);

    printf("Digite o valor do desconto (em %): ");
    scanf("%d", &desconto);

    float desc_decimal = 1 - (((float)desconto) / 100.0);
    valor_final = valor_inicial * desc_decimal;

    printf("O valor do produto com desconto eh R$ %.2f\n",
        valor_final);

    return 0;
}
```



Relembrando: Estruturas de Decisão em C

```c
#include <stdio.h>

int main() {

    int numero;
    printf("Informe um numero: ");
    scanf("%d", &numero);

    if (numero < 0) {
        // faça alguma coisa caso a condição seja verdadeira.
    }
    else {
        // faça alguma coisa caso a condição seja falsa.
    }

    return 0;
}
```

- Condições em C são formadas por estruturas **IF** (SE) e **ELSE** (SENÃO);
- Nem sempre é necessário haver um bloco ELSE. Nesses casos, há um “IF simples”.

Relembrando: Casos de Omissão do ELSE

Retorno precoce de uma subrotina

```c
int max(int a, int b) {
    if (a > b) {
        return a;
    }
    return b;
}
```

Controle de *loops*

```c
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        continue; // Pula números pares
    }
    printf("%d ", i);
}
```

Tratamento de erros

```c
int divisao(int a, int b) {
    if (b == 0) {
        printf("Erro: Divisao por zero\n");
        return 0;
    }
    return a / b;
}
```

Relembrando: Estruturas de Decisão em C

- Blocos **IF** e **ELSE** podem ser encadeados;
- Caso o encadeamento seja extenso, há a estrutura **SWITCH/CASE** (para **INT** e **CHAR**).

```c
int main() {
    int opcao, valor;
    printf("Converter: \n");
    printf("1: decimal para hexadecimal\n");
    printf("2: hexadecimal para decimal\n");
    printf("\nInforme sua opção: ");
    scanf("%d", &opcao);
    if (opcao == 1) {
        printf("\nInforme o valor em decimal: ");
        scanf("%d", &valor);
        printf("%d em hexadecimal eh: %x", valor, valor);
    }
    else if(opcao == 2) {
        printf("\nInforme o valor em hexadecimal: ");
        scanf("%x", &valor);
        printf("%x em decimal eh: %d", valor, valor);
    }
    else {
        printf("\nA opção escolhida eh inválida.")
    }
}
```

```c
int main() {
    int opcao, valor;
    printf("Converter: \n");
    printf("1: decimal para hexadecimal\n");
    printf("2: hexadecimal para decimal\n");
    printf("\nInforme sua opção: ");
    scanf("%d", &opcao);
    switch (opcao) {
        case 1:
            printf("\nInforme o valor em decimal: ");
            scanf("%d", &valor);
            printf("%d em hexadecimal e: %x", valor, valor);
        break;
        case 2:
            printf("\nInforme o valor em hexadecimal: ");
            scanf("%x", &valor);
            printf("%x em decimal e: %d", valor, valor);
        break;
        default:
            printf("\nOpcao invalida. Tente outra vez.")
    }
}
```

Relembrando: Estruturas de Repetição em C

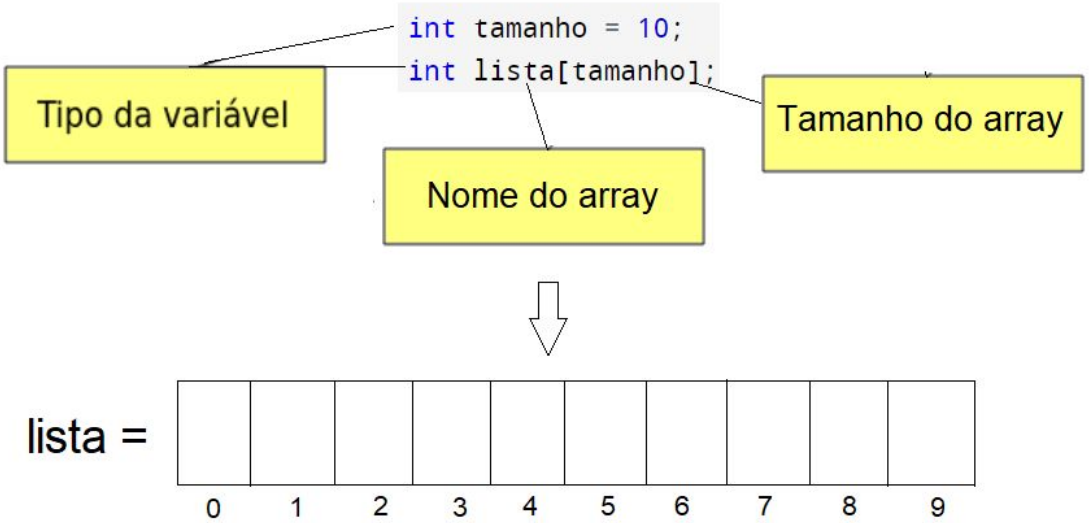
- *Loops* em C podem ser implementados a partir de estruturas **for**, **while** e **do while**;
- Cada estrutura tem um objetivo em particular e deve ser utilizado de acordo com a ocasião.

```c
for (int i = 0; i < n; i++) {
    // variável contadora;
    // número de repetições conhecido.
}
```

```c
int opcao = menu();

while (opcao != 0) {
    // variável sentinela;
    // número de repetições desconhecido.
    opcao = menu();
}
```

```c
// similar ao 'while', porém o bloco é
// executado, pelo menos, uma vez.
do {
    printf("Digite um numero positivo: ");
    scanf("%d", &num);

    if (num <= 0) {
        printf("Entrada invalida!\n");
    }
} while (num <= 0);
```

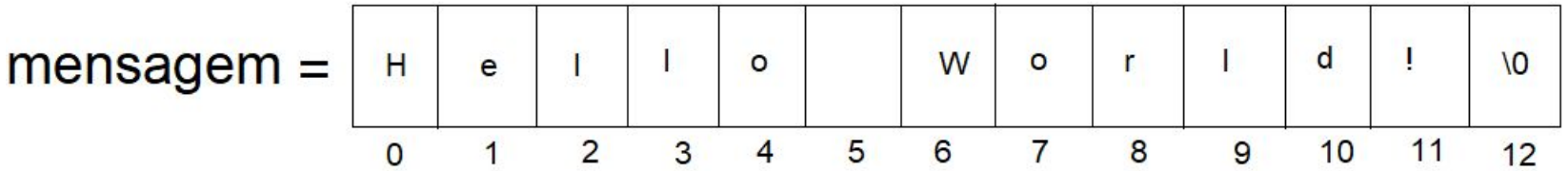
Relembrando: *Arrays* em C



- Um *array* em C é um agrupamento de variáveis do mesmo tipo (estrutura homogênea);
- Arrays utilizam colchetes em sua declaração e acesso aos elementos;
- Sempre é preciso declarar o tamanho dos vetores;
- O primeiro índice em C sempre é 0 (*zero-based*).

Relembrando: *Strings* em C

```c
char mensagem[13] = "Hello World!";
printf("%s", mensagem);
return 0;
```



- Uma *string* é representada por um ***array* de caracteres**;
- São utilizadas para armazenar e imprimir texto;
- Toda *string* em C termina com o caractere nulo \0;
- Strings podem ser lidas via *scanf*. Porém, é preferível o uso da função *fgets;*
- Uma *string* pode ser impressa no terminal via a *string* de formatação %s.

Relembrando: *Strings* em C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LENGTH 100

int main() {
    char msg[MAX_LENGTH];

    printf("Digite um texto com espacos: ");
    fgets(msg, sizeof(msg), stdin);
    printf("%s\n", msg);

}
```

- *Strings* podem ser lidas via *scanf*. Porém, é preferível o uso da função *fgets*.

Relembrando: *Strings* em C

- Há diversas funções úteis na biblioteca <*string.h*>:

```
#include <string.h>
int main()
{
      char test[100];
      char test2[]= "World!\n";
      strcpy(test,"Hello"); /*copia*/
      strcat(test,test2); /* concatenacao*/
      if (strcmp(test,"david") == 0)
            printf ("Test é o mesmo que David\n");
      printf ("comprimento de test é is %d\n",
      strlen (test));
}
```

Relembrando: *Strings* em C

- Se a *string* contém um número, podemos usar as funções **atoi** e **atof** para convertê-la em um número;
- Estas funções estão declaradas na biblioteca ***<stdlib.h>*** e **retornam 0 em caso de erro**.

```
char numberstring[]= "3.14";
int i;
double pi;
pi= atof (numberstring);
i= atoi ("12");
```

Recursividade: Exemplo de uma Fila

- Imagine que você acabou de entrar em uma fila para acessar o caixa eletrônico;
- A fila está longa e **você deseja saber quantas pessoas há na sua frente**. Abordagens:
  1. Sair da fila e contar manualmente quantas pessoas há (com risco de perder o lugar…);
  2. Perguntar à pessoa à sua frente qual o número dela na fila.

Recursividade: Exemplo de uma Fila

- Caso você opte pela Opção 2:
  1. A pessoa à sua frente também não sabe o número dela na fila…;
  2. Então, ela transmite a pergunta para a próxima pessoa à frente dela;
  3. A pergunta se propaga até a primeira pessoa da fila.

Recursividade: Exemplo de uma Fila

- Caso você opte pela Opção 2:
  1. A pessoa à sua frente também não sabe o número dela na fila…;
  2. Então, ela transmite a pergunta para a próxima pessoa à frente dela;
  3. A pergunta se propaga até a primeira pessoa da fila.

Recursividade: Exemplo de uma Fila - Abordagem 1

- A fila está longa e **você deseja saber quantas pessoas há na sua frente**:

1. Sair da fila e contar manualmente quantas pessoas há (com risco de perder o lugar…);

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

int contar_fila_iterativo(bool fila[]) {

    int qtd = 0, posicao = 0;
    do {
        qtd = qtd + 1;
    } while (fila[posicao++] != true);

    return qtd;
}

int main() {

    bool fila[11] = {false};
    fila[10] = true; // marca como 'true' o primeiro da fila.

    printf("Seu numero na fila eh: %d", 1 + contar_fila_iterativo(fila));
    return 0;
}
```

Recursividade: Exemplo de uma Fila - Abordagem 2

- A fila está longa e **você deseja saber quantas pessoas há na sua frente**:

  2. Perguntar à pessoa à sua frente qual o número dela na fila.

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

int contar_fila_recursivo(bool fila[], int posicao) {

    if (fila[posicao] == true)
        return 1;

    return 1 + contar_fila_recursivo(fila, ++posicao);
}

int main() {

    bool fila[11] = {false};
    fila[10] = true; // marca como 'true' o primeiro da fila.

    printf("Seu numero na fila eh: %d", 1 + contar_fila_recursivo(fila, 0));
    return 0;
}
```

O que é Recursividade?

- É uma forma de resolver um problema computacional onde a solução depende de soluções menores do mesmo problema;
- Uma recursão é caracterizada por **três fatores**:
  - é baseada em uma subrotina;
  - contém uma condição de parada;
  - contém uma condição recursiva.

Observações sobre Recursividade

- Recursividade é um maneira de realizar **repetições de forma funcional** (*i.e.* por meio de subrotinas);
- **Única forma de repetição disponível** em algumas linguagens de programação (Prolog, Lisp, Haskell);
- Soluções baseadas em recursividade **podem ser mais simples** do que soluções iterativas convencionais:
    - Exemplos incluem problemas envolvendo estruturas de dados inerentemente recursivas: **grafos e árvores**.
- Contudo, soluções recursivas são, geralmente, **mais complexas e difíceis de analisar**;
- Programas recursivos têm **a pilha (*stack*)** como estrutura de dados principal;
- Programas recursivos tendem a utilizar mais memória e estão sujeitos a um problema de execução chamado ***Stack Overflow***.

Recursividade - Problema 1

- **Elabore um programa recursivo que calcule o fatorial de um número inteiro n >= 0.**
- **Dicas:**
  - Considere o caso inicial como **atômico**, *i.e.* que forneça a resposta mais simples possível;
  - Para implementar o caso recursivo, pense em como reduzir o problema em partes mais simples;
  - Utilize os tipo de retorno e os parâmetros da subrotina para realizar a recursão.

Recursividade - Solução do Problema 1

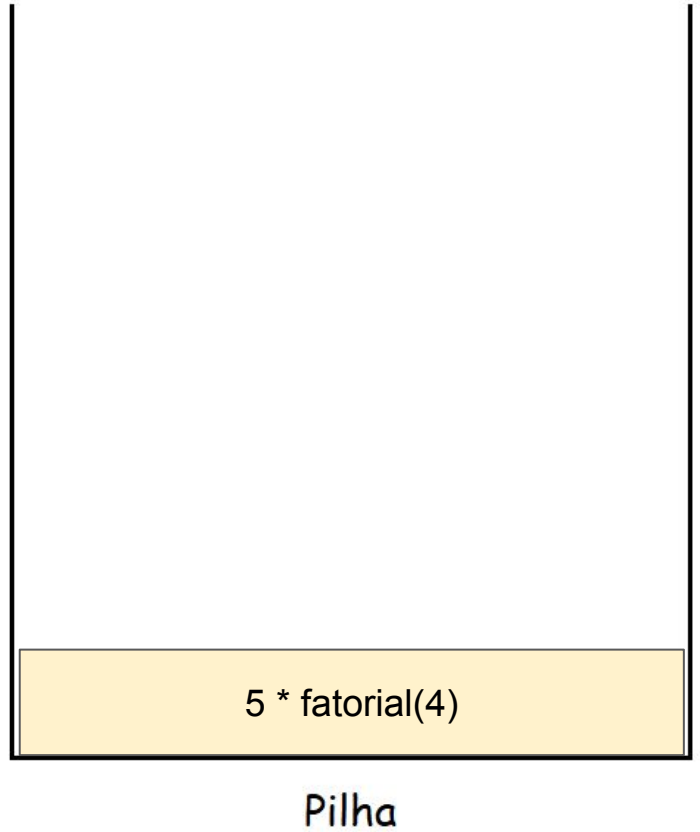
N = 5

```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```

Recursividade - Análise da Solução do Problema 1

N = 5

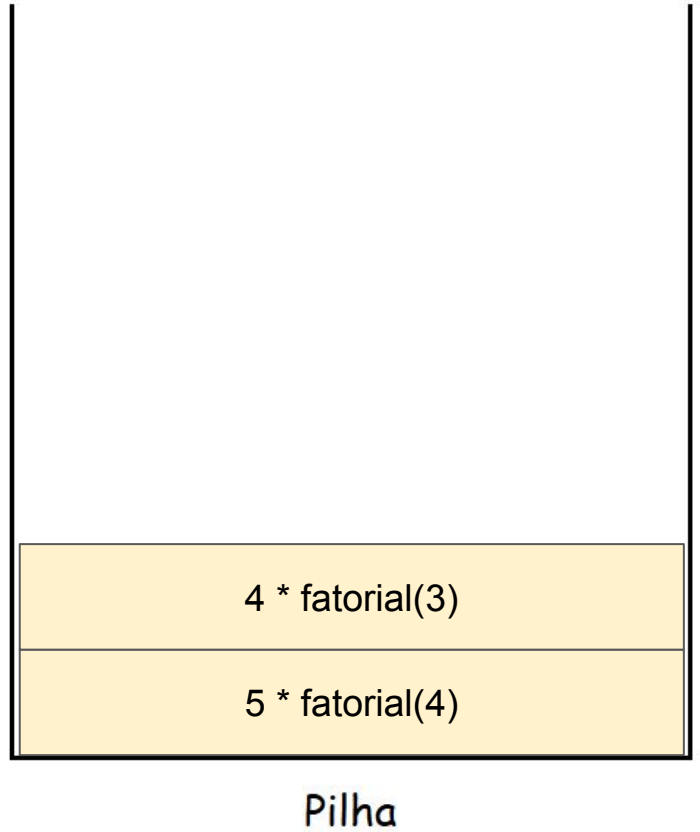
```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```



Recursividade - Análise da Solução do Problema 1

N = 5

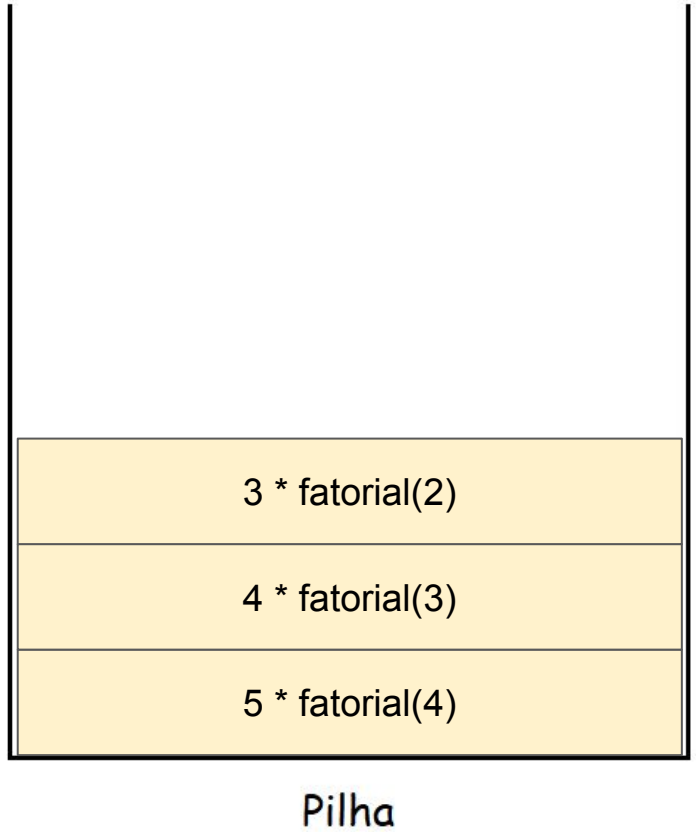
```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```



Recursividade - Análise da Solução do Problema 1

N = 5

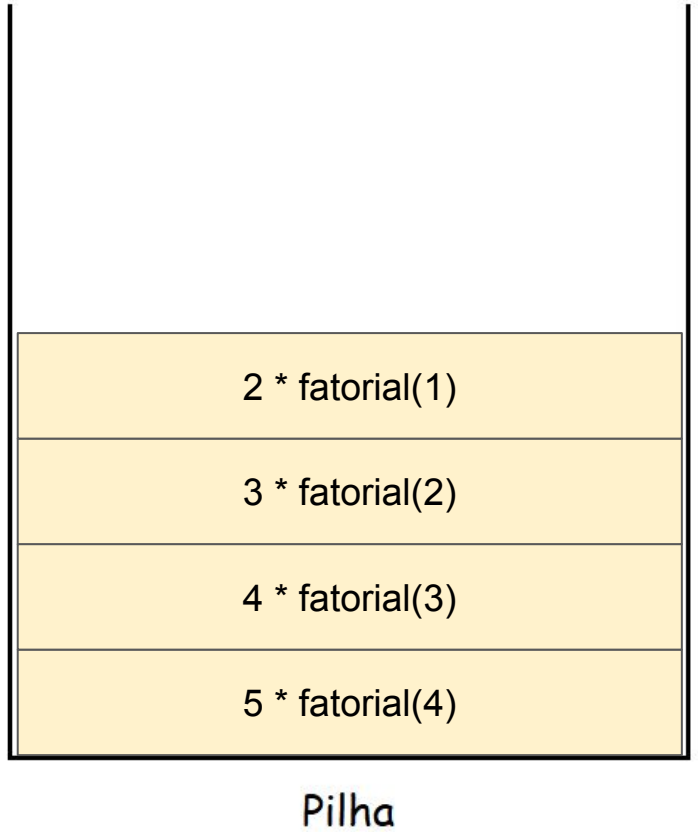
```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```



Recursividade - Análise da Solução do Problema 1

N = 5

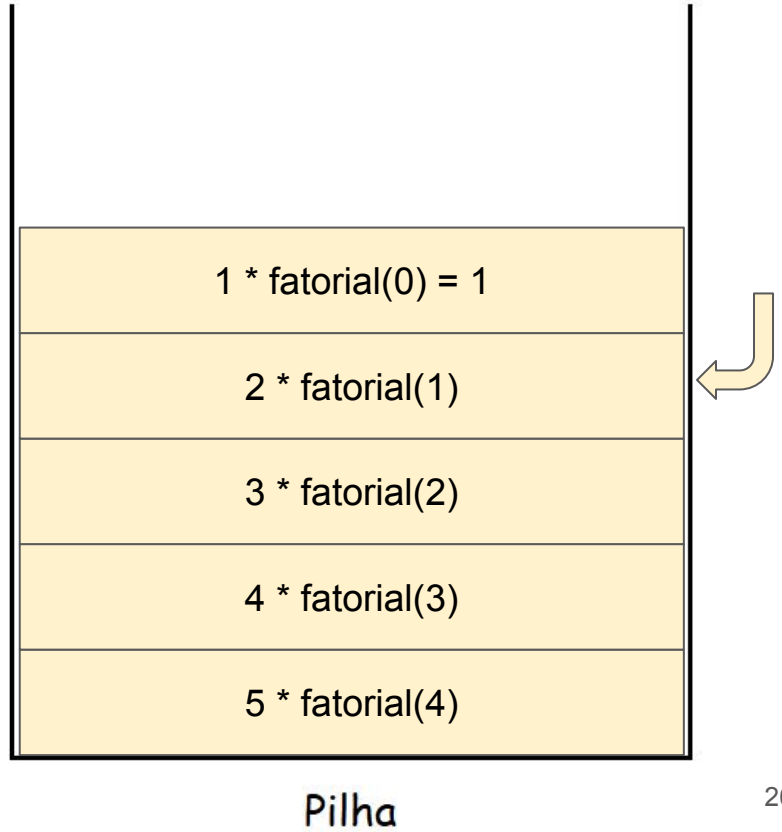
```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```



Recursividade - Análise da Solução do Problema 1

N = 5

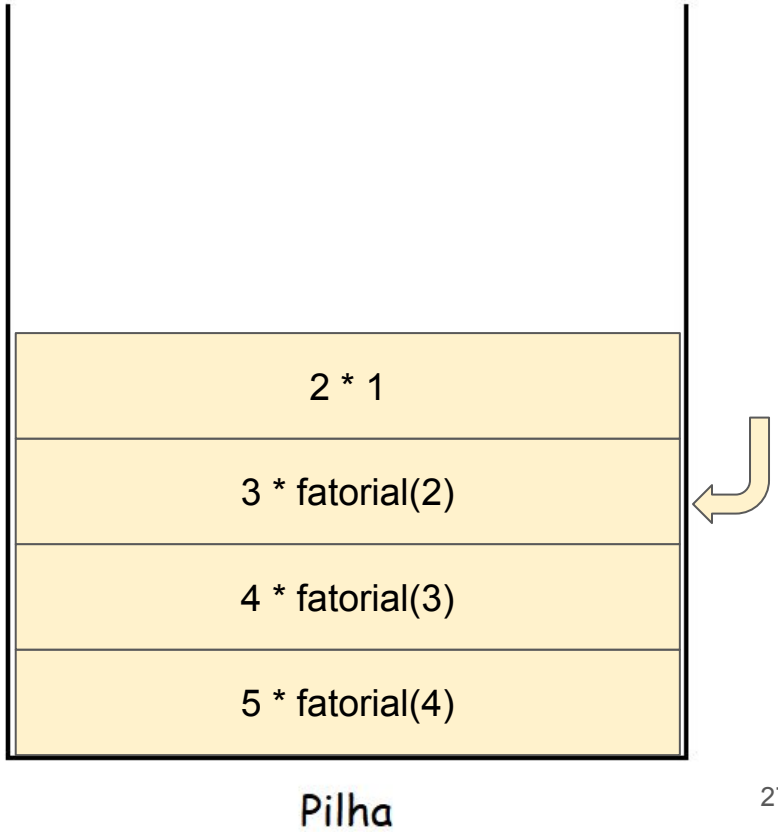
```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```



Recursividade - Análise da Solução do Problema 1

N = 5

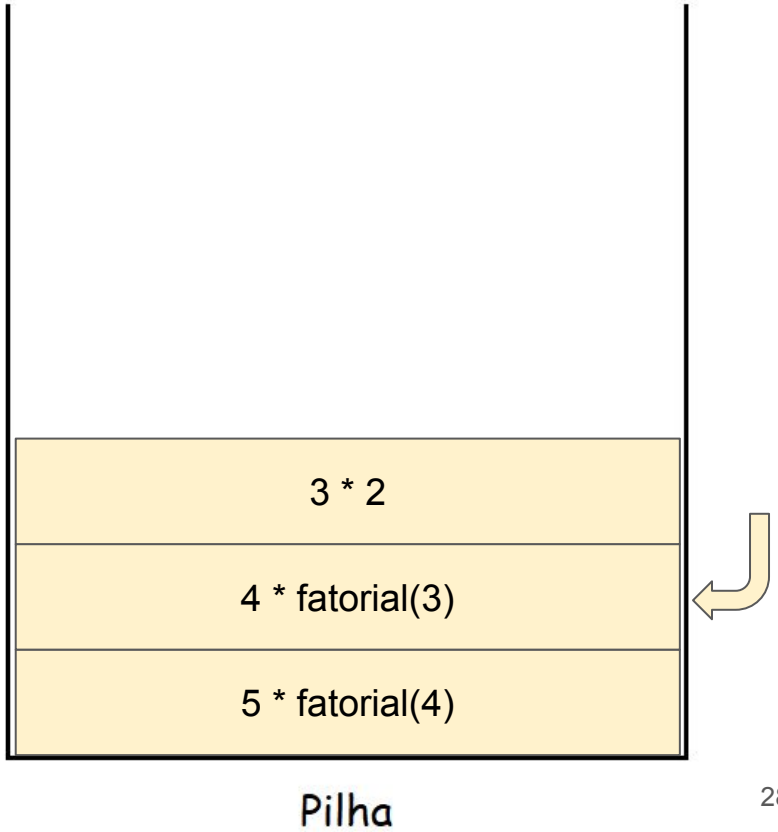
```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```



Recursividade - Análise da Solução do Problema 1

N = 5

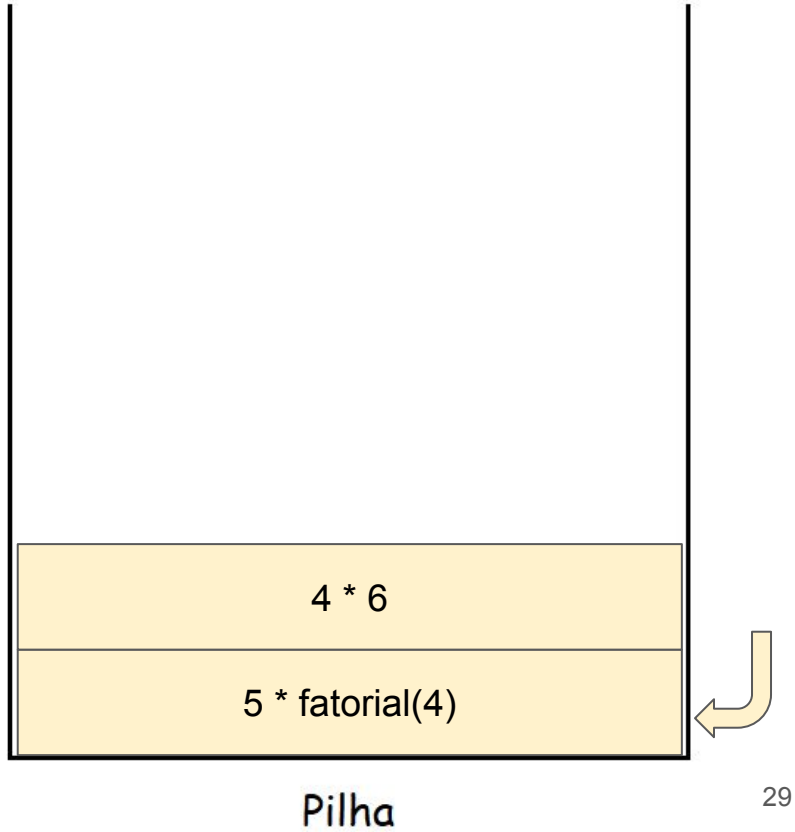
```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```



Recursividade - Análise da Solução do Problema 1

N = 5

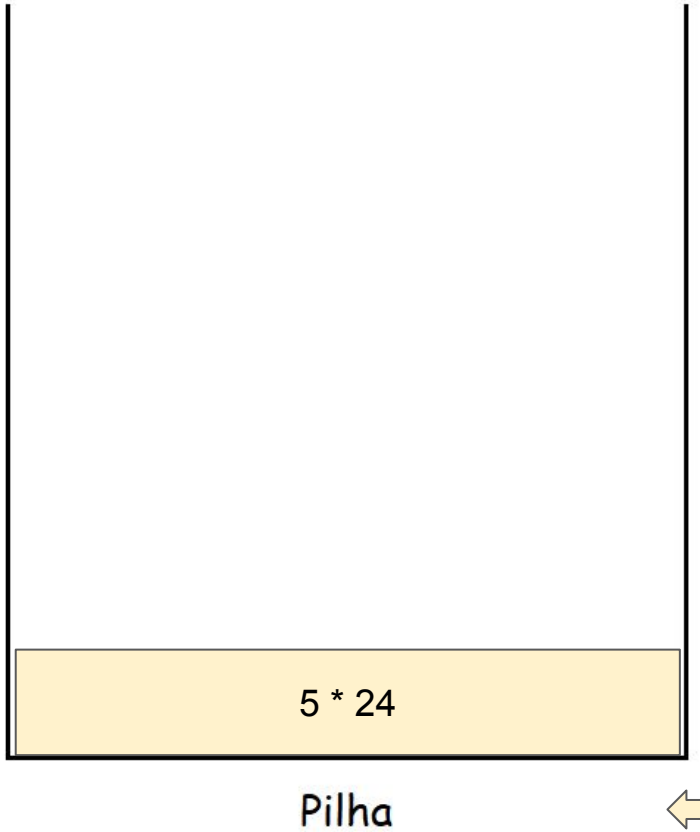
```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```



Recursividade - Análise da Solução do Problema 1

N = 5

```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```



Recursividade - Análise da Solução do Problema 1

N = 5

```c
#include <stdio.h>

int fatorial(int n) {

    if (n == 0 || n == 1)
        return 1; // caso base

    return n * fatorial(n - 1); // caso recursivo
}

int main() {

    printf("Digite um numero inteiro >= 0: ");
    int n, fatorial_n; scanf("%d", &n);

    fatorial_n = fatorial(n);
    printf("%d! = %d", n, fatorial_n);
    return 0;
}
```

Recursividade - Problema 2

- **Elabore um programa recursivo que localize o índice de um número em um *array* em ordem crescente. Caso não encontre o número, o programa deve retornar -1.**
- **Dicas:**
  - Considere o caso inicial como **atômico**, *i.e.* que forneça a resposta mais simples possível;
  - Para implementar o caso recursivo, pense em como reduzir o problema em partes mais simples;
  - Utilize os tipo de retorno e os parâmetros da subrotina para realizar a recursão.

Recursividade - Solução do Problema 2

```c
int busca_binaria(int* lista, int inicio, int fim, int numero_procurado) {

    if (inicio > fim)
        return -1;

    int mid = (inicio + fim) / 2;

    if (lista[inicio] == numero_procurado)
        return inicio;

    if (lista[fim] == numero_procurado)
        return fim;

    if (lista[mid] == numero_procurado)
        return mid;

    if (lista[mid] < numero_procurado)
        return busca_binaria(lista, mid+1, fim, numero_procurado);

    if (lista[mid] > numero_procurado)
        return busca_binaria(lista, inicio, mid-1, numero_procurado);
}
```

Recursividade - Problema 3

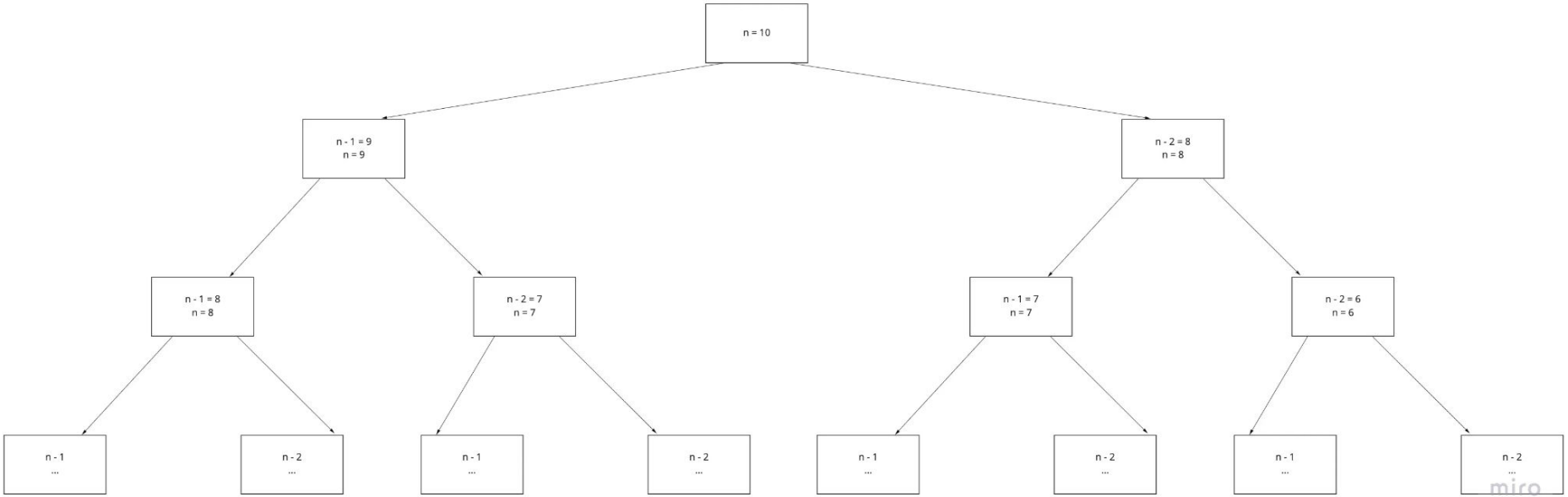
- **Elabore um programa recursivo que retorne o n-ésimo termo da sequência de Fibonacci.**
- **Dicas:**
  - Considere o caso inicial como **atômico**, *i.e.* que forneça a resposta mais simples possível;
  - Para implementar o caso recursivo, pense em como reduzir o problema em partes mais simples;
  - Utilize os tipo de retorno e os parâmetros da subrotina para realizar a recursão.

Recursividade - Solução do Problema 3

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

int fibonacci(int termo) {
    if (termo == 0)
        return 0;

    if (termo == 1)
        return 1;

    return fibonacci(termo - 1) + fibonacci(termo - 2);
}

int main() {

    printf("Digite o termo de Fibonacci: ");
    int termo; scanf("%d", &termo);
    printf("O termo %d de Fibonacci eh %d", termo, fibonacci(termo));
    return 0;
}
```

Recursividade - Discussão do Problema 3

- Algoritmo de complexidade exponencial: $O(2^n)$;
- Repetição de soluções já computadas em passos anteriores;
- Versão iterativa muito mais eficiente: $O(n)$, porém é possível chegar a $O(\log n)$ (*recursive squaring*).



Recursividade vs. Iteração - Quando Usar?

- **Recursão:**
  - Quando o problema pode ser dividido em subproblemas menores e similares;
  - Quando o problema naturalmente exibe auto-similaridade, como fractais;
  - Quando a solução se beneficia da clareza e simplicidade em detrimento de soluções iterativas.
- **Iteração:**
  - Quando o *overhead* de desempenho e memória são fatores de preocupação;
  - Quando a profundidade da recursão pode ser potencialmente muito grande;
  - Quando o problema pode ser resolvido de forma mais natural usando *loops*.

Laboratório de Fundamentos em TIC

Controles de Execução e Recursividade

Prof. Gabriel Resende Machado

gabrielmachado@unifeso.edu.br

https://www.linkedin.com/in/machadogabriel

https://github.com/UNIFESO-Gabriel/fundamentos-em-tic