Professor
Maromo

LINGUAGEM DE PROGRAMAÇÃO

Material 007

C

Github
maromo71

Agenda

Funções Recursivas

- **Conceito**
- **Exemplos**
- **Exercícios**

Material: LP_007

Recursividade

- Capacidade que uma linguagem de programação tem de permitir que uma função possa invocar a si mesma.
- A recursividade pode ser **direta** ou **indireta**.
  - **Direta**:
    - Quando uma função invoca a si mesma no seu corpo da função.
  - **Indireta:**
    - Quando uma função **f** invoca uma outra função **g** que, por sua vez, volta a invocar a função **f.**

Clássico

Fatorial de um número

Problema:

Implemente uma função fatorial que calcula o valor de
n! = n * (n – 1) * (n – 2) * ..... * 2 * 1
Sabendo-se que 0!=1.

Clássico

Versão tradicional – Sem uso de recursividade.

```c
#include <stdio.h>
int fat(int n){
    int f = 1;
    for (int i = 1; i <= n; i++) {
        f *= i;
    }
    return f;
}
int main(){
    int n = 5;
    printf("Fatorial de %d = %d \n", n, fat(n));
    return 0;
}
```

Observe

Dada a definição tradicional do fatorial:

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$$

Pode-se observar que:

$$n! = n \times (n-1)!$$

Isso porque:

$$(n-1)! = (n-1) \times (n-2) \times \ldots \times 2 \times 1$$

Portanto, a definição recursiva do fatorial é:

$$n! = n \times (n-1)!$$

Com Recursividade

```c
#include <stdio.h>
int fat(int n){
    if(n==1)return 1;
    return n * fat( n: n-1);
}
int main(){
    int n = 5;
    printf("Fatorial de %d = %d \n", n, fat(n));
    return 0;
}
```

Na execução do cálculo houve um **empilhamento de chamadas da função fatorial(int n)**, até que a mesma atingisse a condição de saída, ou seja **n= 1.**

Processo de Empilhamento e Desempilhamento

fat (n) = n * fat (n-1) → até que n = 1

fat (5) = 5 * fat (4)

fat (4) = 4 * fat (3)

fat (3) = 3 * fat (2)

fat (2) = 2 * fat(1)

fat (1) = 1

Empilhando

fat (n) = n * fat (n-1) → até que n = 1

fat (5) = 5 * 24 → 120

fat (4) = 4 * 6 = 24

fat (3) = 3 * 2 = 6

fat (2) = 2 * 1 = 2

fat (1) = 1

Desempilhando

Regras para escrita de uma função recursiva

1. **Estabeleça imediatamente um critério de término** (ou condição base) para as chamadas. Esse critério previne que a função se chame infinitamente e eventualmente estoure a pilha de chamadas.
2. Após definir claramente o critério de término, prossiga com a implementação da chamada recursiva da função. **Esta chamada é o que dá à função sua natureza recursiva.**
3. A recursividade pode **melhorar a legibilidade e a simplicidade do código**, reduzindo a quantidade de código escrito. **No entanto, é importante notar que isso pode vir à custa da performance**, especialmente em casos onde a profundidade da recursão é significativa.

Problema

Problema

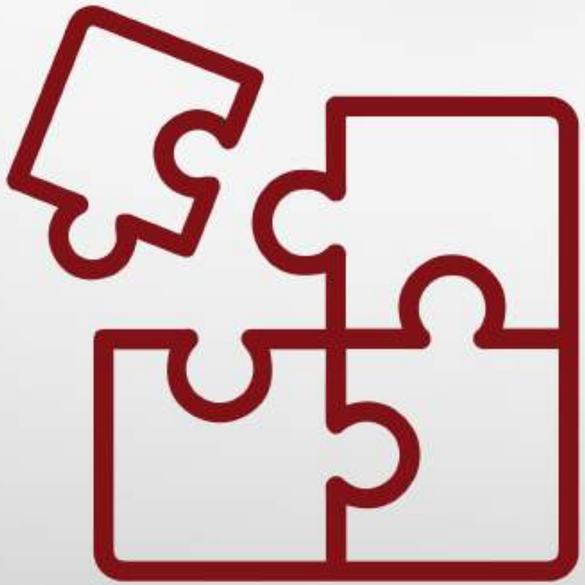
Implemente **de forma recursiva**, a **função up** que escreve na tela os n primeiros números de forma crescente.

A função terá um único parâmetro e não retorna qualquer tipo de resultado.

```
void up(int n){

}
```

Solução do problema



```c
#include <stdio.h>
void up(int n){
    if(n < 1) return;
    up(n - 1);
    printf("%d ", n);
}
int main(){
    up(10);
    return 0;
}
```

Desafio 1

Com base no exemplo anterior, implemente de forma recursiva, a função **down** que escreve na tela os **n** primeiros números de forma **decrescente**.

A função terá um único parâmetro e não retorna qualquer tipo de resultado.

```
void down(int n)
```

Desafio 2

Implemente uma função de forma recursiva que calcule a soma dos **n** primeiros números recebido como argumento o teto **N.**

S(N - 1) + N, se N > 1.
void somatoria(int n)

Solução do desafio 2

```c
#include <stdio.h>
int soma(int n){
    if(n == 1) return 1;
    return n + soma(n - 1);
}
int main(){
    printf("Soma dos %d primeiros numeros inteiros = %d\n", 10, soma(10));
    return 0;
}
```

Quebrando a cabeça

Exercícios - Aula 07
Funções Recursivas

1. Implemente uma função recursiva que calcule o valor de uma base x elevada a um expoente y.
2. Desenvolva uma função recursiva `**tamstring(char s[])**` que determine e retorne o comprimento de uma string.
3. Crie uma função recursiva `**caract(char c, char s[])**` que conte e retorne o número de ocorrências de um caracter específico c em uma string.
4. Elabore uma função recursiva `**reverse(char s[])**` que inverta e retorne a ordem dos caracteres de uma string dada.

Exercícios - Aula 06
Resolução Exercício 01

```c
#include <stdio.h>
int potencia(int x, int y) {
    if (y == 0) return 1; // condição base
    return x * potencia(x, y-1);
}
int main(){
    printf("Base 2, expoente 10 = %d \n", potencia(2,
10));
    return 0;
}
```

Exercícios - Aula 06
Resolução Exercício 02

```c
#include <stdio.h>
int tamstring(const char s[]) {
    if (s[0] == '\0') return 0; // condição base
    return 1 + tamstring(s + 1);
}
int main(){
    char frase[] = "Ola alunos";
    printf("Total de letras: %d \n", tamstring(frase));
    return 0;
}
```

Exercícios - Aula 06
Resolução Exercício 03

```c
#include <stdio.h>
int caract(char c, const char s[]) {
    if (s[0] == '\0') return 0; // condição base
    return (s[0] == c ? 1 : 0) + caract(c, s + 1);
}
int main(){
    char frase[] = "Ola alunos";
    printf("Total de letras encontradas: %d \n", caract('a',frase ));
    return 0;
}
```

Exercícios - Aula 06
Resolução Exercício 04

```c
#include <stdio.h>
#include <string.h>
void reverseAux(char s[], int start, int end) {
    if (start >= end) return; // condição base
    char temp = s[start];
    s[start] = s[end];
    s[end] = temp;
    reverseAux(s, start + 1, end - 1);
}
int main(){
    char frase[] = "Ola alunos";
    printf("Frase [%s] \n", frase);
    reverseAux(frase, 0, strlen(frase)-1);
    printf("Frase Reversa [%s] \n", frase);
    return 0;
}
```

Referências

DAMAS, L. M. D. Linguagem C. LTC, 2007.

HERBERT, S. C completo e total. 3a. ed. Pearson, 1997.