Aula prática 5

Recursão

Monitoria de Introdução à Programação

Roteiro

- **Recursão**
  - Definição
  - Em C
  - Entendendo a recursão
  - Exemplos
  - Vantagens
  - Desvantagens
  - Usos comuns
- **Dúvidas**
- **Exercícios**

Recursão – Definição

“Para entender recursão, precisamos primeiro entender recursão.”

- Recursão é o artifício de fazer um procedimento chamar outra instancia dele próprio, uma alternativa aos laços.

Exemplo:

$$\text{Fib}(n) = \begin{cases} 0 & \text{, para } n = 0 \\ 1 & \text{, para } n = 1 \\ \text{Fib}(n-1) + \text{Fib}(n-2) & \text{, para } n > 1 \end{cases}$$

Recursão – Em C

- Em C, para criar uma função recursiva, só é necessário que ela chame a si própria*.

- **Para não gerar uma recursão infinita, é necessário criar uma condição de parada**: similar à de laços, ela vai dizer quando a função deverá parar de chamar instâncias dela.

*: ou que chame outra função, que por sua vez chame a primeira, e etc.

Recursão – Em C - Exemplo

```
int fatorial (  int n ){
        int resposta;

        if(n <= 1){
                resposta = 1 ;
        } else {
                resposta = n * fatorial ( n - 1 );
    }

        return resposta;
}
```

Elementos da recursão

- Normalmente a recursão é dividida em duas partes básicas:

- Caso base
  - É a condição de parada da recursão
  - Pode existir mais de um caso base

- Caso recursivo
  - É a alma da recursão
  - Permite ao programador redefinir o problema para um que se aproxime mais de um caso base

Passos básicos de um algoritmo recursivo

1. Inicializar o algoritmo. É comum que algoritmos recursivos necessitem de um ponto de partida, o qual normalmente é passado por parâmetro;
2. Verificar se o valor avaliado bate com o caso base;
3. Redefine a resposta em um subproblema menor;
4. Roda o algoritmo no subproblema definido;
5. Combina o resultado para formular a resposta;
6. Devolve um resultado.

Algoritmo Recursivo - Fatorial

1. **fatorial(3);**
2. **3 é menor ou igual a 1? Não!**
3. **3 * fatorial(2);**

   2 é menor ou igual a 1? Não!

   2 * fatorial(1) ...

   1. 1 é menor ou igual a 1? Sim!
   2. devolve 1;

   devolve 2;
4. **devolve 6;**

Consequências

- Recursões são feitas utilizando funções
- Cada chamada a uma função possui suas próprias variáveis, independentes do restante do programa
- Logo, cada chamada recursiva guardará seus próprios valores para suas variáveis, o que é chamado de Estado.

Recursão - Divisão inteira

- **Faça a operação de divisão inteira de forma recursiva para números naturais.**

- **O quociente de uma divisão inteira é a quantidade de vezes que se pode subtrair o dividendo do divisor**

Recursão - Divisão inteira

- **Caso base: quando não posso mais subtrair**
  - dividendo < divisor
  - Neste caso, a resposta é sempre 0

- **Caso recursivo: quando posso subtrair**
  - dividendo >= divisor
  - Agora, podemos subtrair pelo menos 1 vez (não sabemos se podemos subtrair mais de 1 vez)
  - Logo, o quociente é no mínimo 1
  - Então, a resposta será 1 + a quantidade de vezes que podemos subtrair após subtrair esta 1 vez

Recursão - Divisão inteira

```
int dividir (  int dividendo , int divisor ){
        int quociente;

        if(dividendo < divisor){
                quociente = 0 ;
        } else {
                quociente = 1 + dividir ( dividendo - divisor ,
divisor);
    }

        return quociente;
}
```

Recursão - Divisão inteira

```
int quoc = dividir ( 12 , 5 );
```

dividendo = 12;
divisor = 5;
Como 12 >= 5, quociente será 1 + dividir(12 - 5 , 5)

dividendo = 7;
divisor = 5;
Como 7 >= 5, quociente será 1 + dividir(7 - 5 , 5)

dividendo = 2;
divisor = 5;
Como 2 < 5, quociente será 0

Quando a chamada recursiva acaba, voltamos a quem a chamou e temos
quociente = 1 + 0 = 1

Quando a chamada recursiva acaba, voltamos a quem a chamou e temos
quociente = 1 + 1 = 2

Recursão

- Recursão não se aplica somente a conjuntos indutivos
- Por exemplo, se temos "N" crianças e "M" barcos que podem carregar até "N" crianças, de quantas formas podemos dividi-las entre os barcos?
- Por combinação, temos esta resposta.
- Mas QUAIS seriam essas formas? É possível fazer com recursão.

Recursão - Vantagens

- Alguns problemas são mais simples de modelar de forma recursiva.
- Soluções recursivas são, geralmente, mais elegantes que suas contrapartes iterativas.
- Porque a linguagem C salva o estado da execução das funções, não é necessário se preocupar com salvar as variáveis em algum lugar antes de chamar uma nova instância da função. Numa solução iterativa, isso seria necessário.

Recursão - Desvantagens

- Recursão é lenta: Ao chamar uma função, precisamos copiar os parâmetros e inicializar as variáveis. Isso demanda tempo.
- Recursões grandes consomem uma grande quantidade de memória, pois é necessário guardar o estado das funções que estão esperando a próxima retornar
- Um deslize na definição da condição de parada pode fazer seu programa chamar a função infinitamente. **Sempre se assegure que há uma condição de parada.**

Dúvidas?

Exercício 1

- Escreva um programa que receba números do usuário até que 0 (zero) seja digitado. Depois imprima os valores digitados na ordem inversa.

- EX:
- 1745290

- Saída:
- 0925471

Obs : O usuário vai colocar um dígito por vez

Exercício 2

- Use recursão para implementar a combinação de 2 numeros N e K, 0 < K , N

- Fórmula de combinação:

$$\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$$

Exercício 3

- Dado dois números naturais, calcule o valor da função de Ackermann.

$$A(m,n) = \begin{cases} n+1 & \text{se } m = 0 \\ A(m-1,1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m-1,A(m,n-1)) & \text{se } m > 0 \text{ e } n > 0. \end{cases}$$