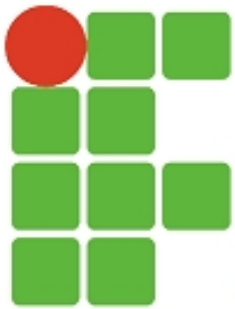

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PARAÍBA
Campus Campina Grande

7

Curso Superior de Tecnologia em Telemática
Programação e Estruturas de Dados

Recursividade – Fundamentos

Prof. César Rocha
cesarocha@ifpb.edu.br

Objetivos

- *Explorar os conceitos fundamentais acerca do uso de **funções recursivas** em estruturas de dados*
  - *Quando e como usar funções recursivas, fundamentos da recursividade, principais técnicas, exemplos práticos, vantagens e desvantagens, recursividade em estruturas de dados e boas práticas de programação.*
- *Revisar alguns conceitos já formados sobre este assunto e estabelecer uma ligação com as funções relacionadas aos TAD´s de estruturas de dados*
  - *Apresentar alternativas interessantes para as construções de código que estão sendo realizadas*

Motivação

- *Em C, vimos que os programas são organizados em blocos fundamentais de execução:* ***as funções***
  - *É bastante comum, em nossos programas, fazermos com que uma função chame uma outra função para desempenhar alguma tarefa específica*
- *Porém, em algumas situações, é útil fazer com que uma função seja chamada* ***recursivamente***
  - *Isto é, esta função irá realizar uma chamada a si própria, dentro de seu corpo*
  - *Recursividade é um tópico um pouco mais avançado da construção de algoritmos*

Importante

- *Muitos livros abordam a recursividade com base em algum problema pré-estabelecido e específico:*
  - *Séries Fibonacci, Fatorial de um número, Torres de Hanoi, Algoritmo da Bolha, entre muitos outros*
  - *Porém, quando outro problema é apresentado ao aluno, o mesmo fica condicionado a aplicar as mesmas regras do primeiro exemplo ao segundo*
- *O que é importante observar:*
  - *Problemas recursivos tendem a apresentar (vários) aspectos semelhantes uns com os outros*
  - *O aluno pode ter um bom "faro" para identificá-los*

Fundamentos da Recursividade

- *Para que se possa criar algoritmos recursivos sem grandes problemas,* ***deve-se estar atento*** *à:*
  - ❶ *uma chamada recursiva é feita de forma semelhante às chamadas de função já vistas*
    - *Não há nenhuma palavra chave específica da linguagem de programação para usar recursão, etc.*
  - ❷ *deve-se notar que toda função recursiva sabe apenas como resolver um problema (simples e atômico)*
    - *De modo que, se a função for chamada em cima deste problema pequeno, ela irá emitir um resultado imediato*
    - *A este elemento, dá-se o nome de* ***"caso base"*** *ou "condição de parada"*

Fundamentos da Recursividade

❸ entretanto, quando uma função recursiva é chamada em cima de um problema mais complexo (e maior), ela dividirá o problema em duas partes conceituais:

- A primeira parte é a que a função sabe como resolver de imediato (o caso base)
- A segunda, tornar-se-á uma parte que ela não sabe como resolver de imediato. Mas a função sabe que esta parte ficou menor que o problema original

❹ geralmente, este novo problema (menor) se parece bastante com o original

- Então, a função irá apenas chamar uma nova cópia de si mesma para trabalhar em cima deste novo problema

Fundamentos da Recursividade

❺ *a esta nova cópia da função original que lidará com o problema, dá-se o nome de* ***chamada recursiva***

- *A chamada recursiva, então, fará com que o novo problema seja novamente dividido em duas partes conceituais e chamará uma nova cópia de si mesma*
- *Assim, novos passos recursivos podem ser gerados, cada um com uma versão (parte dividida) cada vez menor do problema original*

❻ *este processo de divisão irá convergir até um ponto atômico (ou seja, o caso base)*

- *Neste momento, a última chamada recursiva que reconhecer o caso base, irá retornar o seu resultado para o método anterior (em* ***stand by****) que a chamou*

O problema do Fatorial

- *Uma vez que os aspectos a serem perseguidos em problemas recursivos foram vistos, é o momento de exercitar estes conceitos auferidos*
- *Para isso, iremos começar com um problema matemático clássico em programação conhecido com o Fatorial de um número*

Fatorial: n! = n * (n-1)!

- *Onde, por definição:*
  - *Se n = 0, fatorial será igual a 1*
  - *3! = 3*2*1 (6)*

- ***Quem é a caso base? Porque?***
- ***Como poderíamos dividir o fatorial de um número maior em fatoriais menores?***

Fatorial – solução iterativa

- *Note, abaixo, que podemos resolver este problema utilizando uma abordagem iterativa (não-recursiva)*
- *O fatorial de um número maior que zero pode ser calculado facilmente utilizando-se um* ***for{...}***

```
// Considere o código abaixo no cálculo do fatorial
// utilizando uma abordagem iterativa
int fatorial( int numero ){
    int contador, fatorial = 1;
    if ( numero==0 )  return fatorial;
    for( contador=numero; contador >=1; contador-- )
        fatorial *= contador;
    return fatorial;
}
```

O problema do Fatorial

Pense um pouco...

- Neste problema do fatorial, quem é o caso base (ou a condição de parada)?
- E ainda, como podemos dividir o problema (fatorial de um número maior) em pequenos problemas antes de fazer uma chamada recursiva (ou seja, gerar fatoriais de números menores)?
  1. podemos calcular o fatorial de qualquer número enquanto ele for diferente de 0. Se for igual, pare!
  2. O fatorial de 3 = 3 * 2 * 1 == 3 * (2!)

Fatorial – solução recursiva

- *Algumas tentativas:*

```
// Considere o código abaixo no cálculo do fatorial
// utilizando uma abordagem recursiva
int fatorial( int numero ){
    if ( numero==0 )  return 1; //caso base
    else
      return ( numero * fatorial( numero - 1) );
}
```

```
// Considere o código abaixo no cálculo do fatorial
// utilizando uma abordagem recursiva e operador ternário
int fatorial( int numero ){
  return (numero==0 ? 1 :  fatorial( numero - 1) );
}
```

Apenas UMA única linha de código foi feita!

Recursão ou Iteração?

- Ambas iteração e recursão utilizam estruturas de controle convencionais da programação
  - Iteração, normalmente, usa estruturas de repetição: **for**, **while** ou **do/while**
  - Recursão, normalmente, usa estruturas de decisão: **if**, **if/else** ou **switch**
- Ambas iteração e recursão envolvem repetições:
  - Iteração explicitamente utiliza estruturas de repetição
  - Já a recursão alcança a repetição através de chamadas repetitivas (e recursivas) de si mesma (cópias)

Recursão ou Iteração?

- Ambas iteração e recursão possuem uma condição de parada bem definida
  - A iteração termina quando a condição do loop falha
  - Recursão termina quando o caso base é encontrado
- Ambas iteração e recursão podem ocorrer infinitivamente
  - O loop infinito poderá ocorrer quando o teste condicional nunca caminhar para falso
  - Recursão infinita pode ocorrer quando chamadas recursivas não dividem o problema em pedaços menores de modo a não convergir para o caso base

Recursão ou Iteração?

- Quando a questão envolve **memória**
  - A iteração utiliza pouca memória para manipular seus contadores ou variáveis decisórias
  - Recursão causa um overhead na máquina face que cada chamada recursiva será alocada na pilha
- Quando a questão envolve **facilidade de código**
  - Geralmente, soluções iterativas possuem mais linhas de código que as recursivas (são maiores)
  - Recursão, geralmente, reduz o código significavelmente e pode refletir com maior naturalidade o problema que está sendo resolvido no programa

Exercícios

- *Usando recursividade, escreva agora uma função que possa calcular o tamanho de uma lista simplesmente encadeada*
  - *O protótipo da função final pode ser dado por:*

int tamanho(TListaEnc lista);

  - *Recursivamente, podemos dizer que uma lista encadeada é representada por:*
    - *Uma lista vazia ou*
    - *Um elemento seguido de uma (sub-)lista. Neste caso, o segundo elemento representa o 1º elem. da sub-lista.*
  - *Compare com o código já feito em aulas anteriores*

Para um bom aproveitamento:

- *Codifique os exemplos mostrados nestes slides e verifique pontos de dúvidas*
- *Resolva todas as questões da* ***lista de exercícios de recursividade***
- *Procure o professor ou monitor da disciplina e questione conceitos, listas, etc.*
- *Não deixe para codificar tudo e acumular assunto para a primeira avaliação.*
  - *Este é apenas um dos assuntos abordados na prova!*