unipampa

Universidade Federal do Pampa


# Laboratório de Programação I

# Assunto de Hoje
# Recursividade

Professor Luciano Brum
lucianobrum@unipampa.edu.br

# Recursividade

- Até o momento, foram construídos diversos exemplos de funções e procedimentos para os mais diversos fins.

- Todos exemplos trabalhados até aqui foram funções estruturadas ou construídas de forma *iterativa*.

# Recursividade

- Definição: recursividade é uma propriedade de uma função qualquer que chama/invoca a si mesma.

- A recursividade é uma forma interessante de resolver problemas por meio da divisão dos problemas em problemas menores de mesma natureza.

# Recursividade

- Se a natureza dos subproblemas é a mesma do problema, o mesmo método usado para reduzir o problema pode ser usado para reduzir os subproblemas e assim por diante.

- Quando devemos parar? Quando alcançarmos um caso trivial que conhecemos a solução.

# Recursividade

- Portanto recursão é, dentro do corpo de uma função, chamar novamente a própria função.

  - Recursão direta: a função A chama a própria função A.

  - Recursão indireta: a função A chama a função B que, por sua vez, chama A.

# Recursividade

- Problemas/algoritmos de natureza recursiva:
  - Fatorial;
  - Série de Fibbonacci;
  - Torres de Hanói;
  - Algoritmos de Ordenação (*Heap Sort*, *Quick Sort*, *Merge Sort*, etc);
  - Busca Binária;
  - *Depth-first search (DFS);*
  - *Maior ou menor elemento de uma lista/vetor;*
  - *TADs (árvores, listas, filas, árvores, grafos, etc);*

# Recursividade

- Implementações recursivas devem ser pensadas conforme a definição recursiva do problema que desejamos resolver.

- Exemplo para o problema “Fatorial”:

$$n! = \begin{cases} 1, & se\ n = 0 \\ n * (n-1)!, & se\ n > 0 \end{cases}$$

# Recursividade

- Exemplo para o problema “Fatorial”:

$$n! = \begin{cases} 1, & se\ n = 0 \\ n * (n-1)!, & se\ n > 0 \end{cases}$$

$4! = 4*3!$

$= 4*(3*2!)$

$= 4*(3*(2*1!))$

$= 4*(3*(2*(1*0!)))$

$= 4*(3*(2*(1*1))) = 24$

# Recursividade

Na linguagem C...

# Recursividade

- 1° exemplo: função fatorial iterativa.

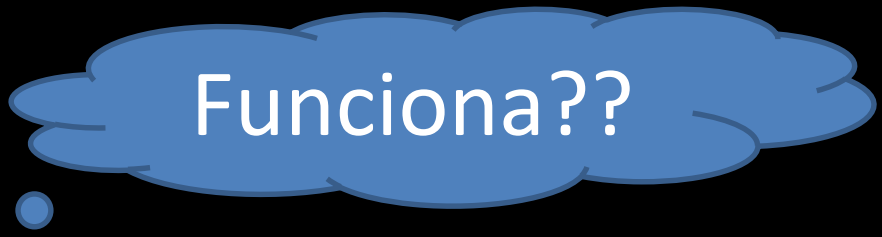
```
int fact(int n){
        int i;
        if(n==0){
                return 1;
        }
        else{
                for(i=0;i<n;i++){
                        n=n*i;
                }
        return n;
        }
}
```

Funciona??

# Recursividade

- 1° exemplo: função fatorial iterativa.

```
int fact(int n){
        int i;
        if(n==0){
                return 1;
        }
        else{
                for(i=0;i<n;i++){
                        n=n*i;
                }
        return n;
        }
}
```



# Recursividade

- 1° exemplo: função fatorial iterativa.

```
int fact(int n){
        int i;
        if(n==0){
                return 1;
        }
        else{
                for(i=1;i<n;i++){
                        n=n*i;
                }
        return n;
        }
}
```

# Recursividade

- 1° exemplo: função fatorial recursiva.

Funciona??

```
int fact(int n){

        return n*fact(n-1);

}
```

Executa infinitamente !
Por que?

# Recursividade

- Estratégia para a definição recursiva de uma função:
  1. Dividir o problema em problemas menores do mesmo tipo.
  2. Resolver os problemas menores (dividindo-os em problemas ainda menores, se necessário).
  3. Combinar as soluções dos problemas menores para formar a solução final.
- Ao dividir o problema sucessivamente em problemas menores eventualmente os casos simples são alcançados:
  - Estes não podem ser mais divididos;
  - Suas soluções são definidas explicitamente;

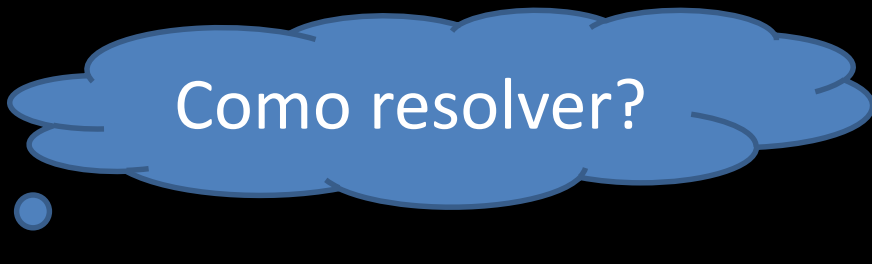
# Recursividade

- Condição de parada: TODO ALGORITMO RECURSIVO DEVE POSSUIR, PELO MENOS, UMA CONDIÇÃO DE PARADA.

- Caso contrário, o programa irá executar infinitas vezes.

# Recursividade

- 1° exemplo: função fatorial recursiva.




```
int fact(int n){

        return n*fact(n-1);

}
```

# Recursividade

- 1° exemplo: função fatorial recursiva.

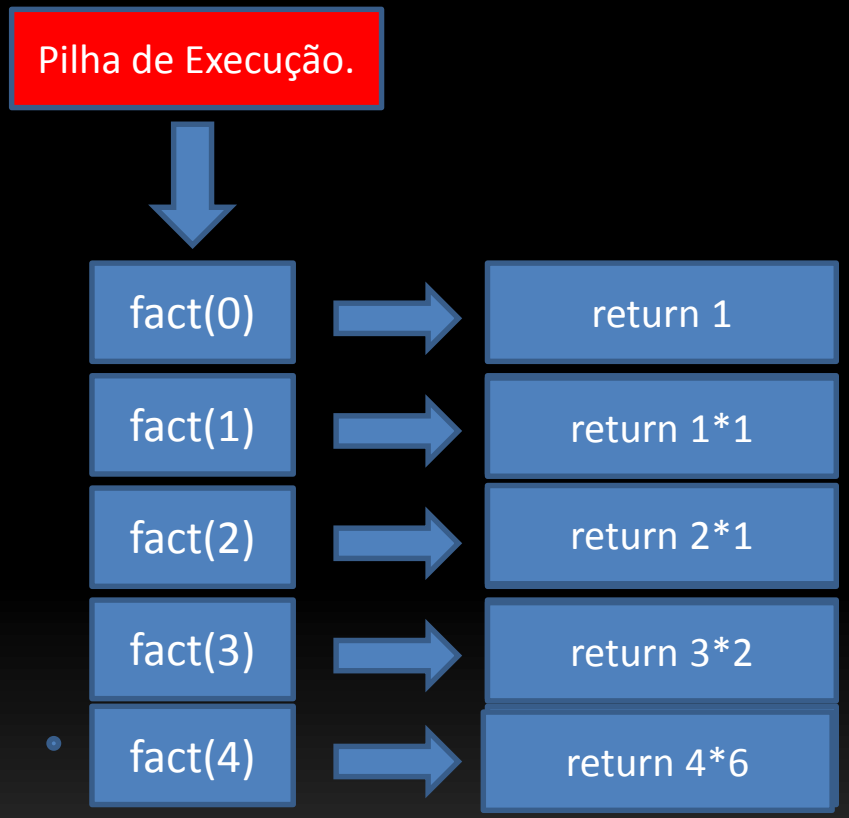
```
int fact(int n){
        if(n==0){
                return 1;
        }
        else{
                return n*fact(n-1);
        }
}
```

Vamos supor que o usuário quer calcular o fatorial de 4.

# Recursividade

- 1° exemplo: função fatorial recursiva.

```
int fact(int n){
        if(n==0){
                return 1;
        }
        else{
                return n*fact(n-1);
        }
}
```




Resultado: 24!

# Exercício em Aula

- O valor de $x^n$ pode ser definido recursivamente como:

$$x^0 = 1$$

$$x^n = x * x^{n-1}$$

- Implemente de forma recursiva o cálculo de $x^n$ para qualquer valor de ‘x’ e ‘n’.

# Recursividade

- Celes, Cerqueira e Rangel (2004) afirmam que implementações recursivas tendem a ser menos eficientes que implementações iterativas.

- Em cada chamada de função, os parâmetros e variáveis locais são empilhados na pilha de execução.

# Recursividade

- Isso também é válido para chamadas recursivas de uma função.

- Ao chamar fact(4), criamos um ambiente local para as variáveis locais e os parâmetros.

- Ao chamar fact(3), criamos um ambiente local separado para as variáveis locais e os parâmetros desta nova chamada da função.

- Isso é feito toda vez que a função for chamada até atingir a condição de parada e executar a instrução *return*.

# Recursividade

- Ponto de Reflexão:
  - Se não fosse criado um ambiente local separado para cada chamada, quais seriam os problemas?
  - E qual a diferença de tudo isso para o método iterativo?

# Recursividade

- Confirmando a afirmação dos autores à respeito de funções recursivas, vejamos um segundo exemplo:

- Série de Fibonacci.

# Recursividade

- Exemplo 2: Sequencia de Fibonacci.

- O n-ésimo termo da sequencia de Fibonacci é definido por:

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

# Recursividade

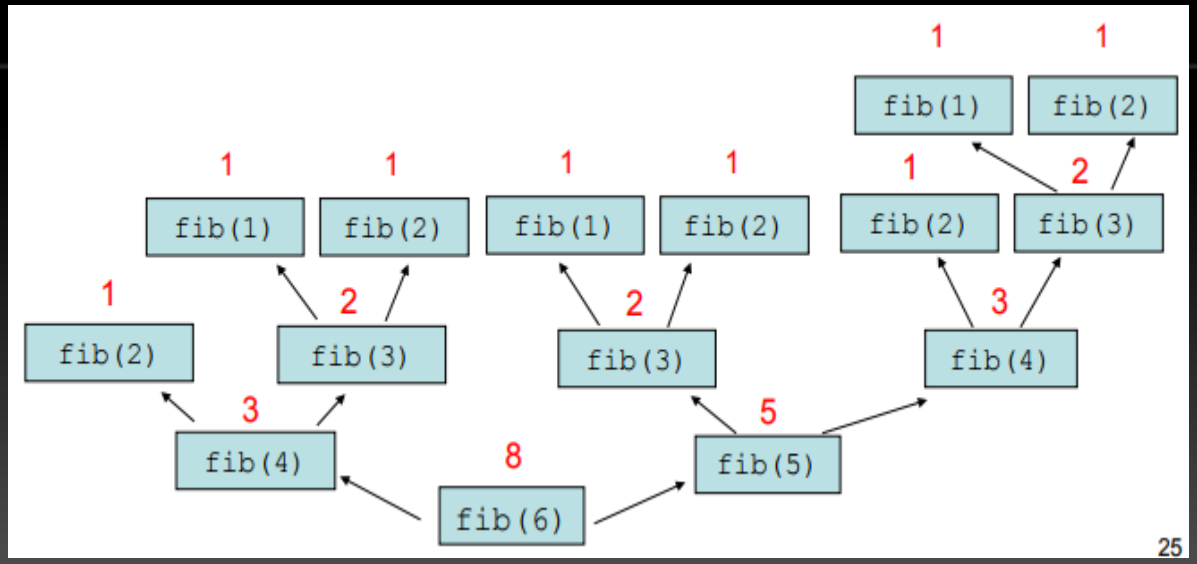
Versão iterativa:

```
int fib(int n){
        int a=1, b=1, c;
        if(n == 1 || n == 2){
                return 1;
        }
        else{
                for(i=1;i<(n-2);i++){
                        c=a+b;
                        b=c;
                        a=b;
                }
        return c;
        }
}
```

# Recursividade

Versão recursiva:

```
int fib(int n){
        if(n == 1 || n == 2){
                return 1;
        }
        else{
                return fib(n-2)+fib(n-1);
        }
}
```



# Recursividade

- Qual o problema da versão recursiva da sequência de Fibonacci?

# Recursividade

- Vantagens:
  - Recursividade vale a pena em algoritmos complexos, cuja a implementação iterativa é complexa e normalmente requer o uso explícito de uma pilha.
  - Dividir para Conquistar (Ex. Quicksort).
  - Caminhamento em Árvores (pesquisa, *backtracking*).

# Recursividade

- Observações:
  - A recursividade nem sempre é a melhor solução, mesmo quando a definição matemática do problema é feita em termos recursivos.

# Resumo

- Vimos os fundamentos teóricos e práticos da recursividade.
- Vimos como criar funções recursivas em C.
- Vimos algumas vantagens e desvantagens da recursividade.

- *Ao tentar resolver o problema, encontrei obstáculos dentro de obstáculos. Por isso, adotei uma solução recursiva.* — aluno S.Y.

- *To understand recursion, we must first understand recursion.* — anônimo.

- *Para fazer um procedimento recursivo é preciso ter fé.* — prof. Siang Wun Song.

# Lista de Exercícios

1. Faça uma função recursiva que permita calcular a média um vetor de tamanho N.
2. Crie um programa em C, que contenha uma função recursiva para encontrar o menor elemento em um vetor. A leitura dos elementos do vetor e impressão do menor elemento devem ser feitas no programa principal.
3. Faça uma função recursiva que receba um número inteiro positivo par N e imprima todos os números pares de 0 até N em ordem decrescente.
4. Faça uma função recursiva que permita somar os elementos de um vetor de inteiros.
5. Crie um programa em C, que contenha uma função recursiva que receba dois inteiros positivos k e n e calcule $k^n$. Utilize apenas multiplicações. O programa principal deve solicitar ao usuário os valores de k e n e imprimir o resultado da função.

# Atividade Semipresencial

1. Escreva duas funções, uma iterativa e outra recursiva, para resolver o problema da sequencia de Fibonacci. Escreva um relatório comparando o tempo de execução das duas funções para diversos casos de teste e outros aspectos que considerar relevante. Utilize qualquer biblioteca para medir o tempo de execução.

- O relatório deve seguir as normas de escrita da Unipampa (http://cursos.unipampa.edu.br/cursos/ppgca/files/2012/08/MANUAL_versao_final1.pdf ) e deve possuir:
  - Introdução: contextualização teórica sobre funções iterativas e recursivas.
  - Objetivo: descrever o objetivo do trabalho em duas frases.
  - Metodologias: Apresentar o código e mostrar como você resolveu o problema.
  - Resultados: Apresentar tempos de execução para diversos casos (diferentes n’s).
  - Conclusões: o que você concluiu com os resultados obtidos?
  - Referências Bibliográficas: Citar todas referências utilizadas.

# Atividade Semipresencial

- Entrega: dia 18/09 pelo moodle.
- Realizar de forma INDIVIDUAL.

unipampa

Universidade Federal do Pampa


# Dúvidas ?