SCC-601– Introdução à Ciência da Computação II

Recursão

Lucas Antiqueira

Exercício

- Implemente uma função para calcular o fatorial de um número inteiro positivo

Definição

- Uma função é dita *recursiva* quando é definida em seus próprios termos

- É uma função declarada como qualquer outra

Exercício

- Implemente uma função recursiva para calcular o fatorial de um número inteiro positivo

Exemplo

- Função que imprime os elementos de um vetor

```
void imprime(int v[], int tamanho) {
    int i;
    for (i=0;i<tamanho;i++)
        printf("%d ",v[i]);
}
```

Exercício

- Faça a versão recursiva dessa função

Exercício

- Solução:

```
void imprime_rec(int v[], int tamanho, int indice) {
   if (indice < tamanho) {
     printf("%d ", v[indice]);
     imprime_rec(v, tamanho, indice + 1);
   }
}
```

Efeitos da recursão

- A cada chamada
  - Empilham-se na memória os dados locais (variáveis e parâmetros) e o endereço de retorno
    - A função corrente só termina quando a função chamada terminar
  - Executa-se a nova chamada (que também pode ser recursiva)
  - Ao retornar, desempilham-se os dados da memória, restaurando o estado antes da chamada recursiva

Exercício

- Simule a execução da função de impressão para um vetor de tamanho 3 e mostre a situação da memória a cada chamada recursiva

```
void imprime_rec(int v[], int tamanho, int indice) {
   if (indice < tamanho) {
     printf("%d ", v[indice]);
     imprime_rec(v, tamanho, indice + 1);
   }
}
```

Recursão

- Quando usar: quando o problema pode ser definido recursivamente de forma natural

- Como usar
  - 1º ponto: definir o problema de forma recursiva, ou seja, em termos dele mesmo
  - 2º ponto: definir a condição de término (ou *condição básica*)
  - 3º ponto: a cada chamada recursiva, deve-se tentar garantir que se está mais próximo de satisfazer a condição de término
    - Caso contrário, qual o problema?

Recursão

- Problema do fatorial
  - 1º ponto: definir o problema de forma recursiva
    - n! = n * (n-1)!
  - 2º ponto: definir a condição de término
    - n=0
  - 3º ponto: a cada chamada recursiva, deve-se garantir que está mais próximo de satisfazer a condição de término
    - A cada chamada, n é decrementado, ficando mais próximo da condição de término

Recursão vs. iteração

- Qual a melhor opção?

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

Versão recursiva

Versão iterativa

```
int fatorial(int n) {
        int i, fat = 1;
        for (i = 2; i <= n; i++)
          fat = fat * i;
        return(fat);
}
```

Exercício

- Implemente uma função recursiva para calcular o enésimo número da seqüência de Fibonacci
  - 1º ponto: definir o problema de forma recursiva
  - 2º ponto: definir a condição de término
  - 3º ponto: a cada chamada recursiva, deve-se garantir que está mais próximo de satisfazer a condição de término

Exercício

- Solução
  - 1º ponto: definir o problema de forma recursiva
    - f(n)=f(n-1)+f(n-2) para n>=2
  - 2º ponto: definir a condição de término
    - n=0 ou n=1, pois f(0)=0 e f(1)=1
  - 3º ponto: a cada chamada recursiva, deve-se garantir que está mais próximo de satisfazer a condição de término
    - n é decrementado em cada chamada

Recursão vs. iteração

- Quem a melhor opção? Simule a execução

//versão recursiva

```
int fib_rec(int n) {
  int res;

  if (n < 2)
    res = n;
  else
    res = fib_rec(n-1)+fib_rec(n-2);
  return(res);
}
```

//versão iterativa

```
int fib(int n) {
  int a = 0, b = 1, k;

  for (k = 1; k <= n; k++) {
    a = a + b;
    b = a - b;
  }

  return(a);
}
```

Recursão vs. iteração

- Quem a melhor opção? Simule a execução

//versão recursiva

```
int fib_rec(int n) {
  int res;

  if (n < 2)
    res = n;
  else
    res = fib_rec(n-1)+fib_rec(n-2);
  return(res);
}
```

//versão iterativa

```
int fib(int n) {
  int a = 0, b = 1, k;

  for (k = 1; k <= n; k++) {
    a = a + b;
    b = a - b;
  }

  return(a);
}
```

Certamente mais elegante, mas duplica muitos cálculos!

Recursão vs. iteração

- Estimativa de tempo para Fibonacci (Brassard e Bradley, 1996)

n	***10***	***20***	***30***	***50***	***100***
Recursão	8 ms	1 s	2 min	21 dias	10^9 anos
Iteração	1/6 ms	1/3 ms	1/2 ms	3/4 ms	1,5 ms

Recursão vs. iteração

- Programas recursivos que possuem chamadas ao final do código são ditos terem **recursividade de cauda**
  - São mais facilmente transformáveis em programas iterativos
    - A recursão pode virar uma condição

Exercício

- Dado um vetor v de n números inteiros, implemente uma função recursiva que retorne o maior elemento do vetor

Resolução

```
int max(int v[], int n) {
   int aux;
   if (n == 1)
     return(v[0]);
   else {
     aux = max(v, n - 1);
     return (v[n-1] > aux) ? v[n-1] : aux;
   }
}
```

Máximo Divisor Comum

- Como calcular o mdc de dois números inteiros positivos?

Máximo Divisor Comum

```
int mdc_direto(int p, int q) {
   int d;

   if (q == 0)
        return p;
   if (p == 0)
        return q;

   d = (p < q) ? p : q;

   while (p % d != 0 || q % d != 0) {
        d--;
   }

   return d;
}
```

Versão iterativa

Máximo Divisor Comum

- Como calcular o mdc recursivamente?

Algoritmo de Euclides

$mdc(m,0) = m$

$mdc(m,n) = mdc(n, resto(m / n)),$ para $n > 0$

Versão recursiva

```
int mdc_rec(int p, int q) {
   if (q == 0)
       return p;
   else
       return mdc_rec(q, p % q);
}
```

Palíndromos

- Um **palíndromo** é uma palavra, frase ou qualquer outra sequência de elementos que tenha a propriedade de poder ser lida tanto da direita para a esquerda como da esquerda para a direita.

- “arara” é palíndromo
- “anotaram a data da maratona” é palíndromo (desconsiderando-se os espaços)

Palíndromos

- Como verificar recursivamente se uma seqüência de caracteres representa um palíndromo?

Solução recursiva

```
int palindromo(char *str, int length) {
   if (length <= 1)
        return 1;

   if (str[0] == str[length-1])
        return palindromo(str + 1, length - 2);
   else
        return 0;
}
```

Conversão Decimal → Binário

- Pense em um algoritmo que converta um número decimal (base 10) para base 2.

Logaritmo

- Logaritmo base 2 de um número inteiro

  $\log_2 n = m$

  Desenvolva um algoritmo recursivo que devolva m dado n.

  Considere apenas n's que sejam potências de 2.

Busca binária

- A pesquisa ou busca binária é um algoritmo de busca em vetores que parte do pressuposto de que o vetor está ordenado.

- Realiza sucessivas divisões do espaço de busca (divisão e conquista).

Busca binária

- Como seria um algoritmo recursivo para a busca binária?

Solução

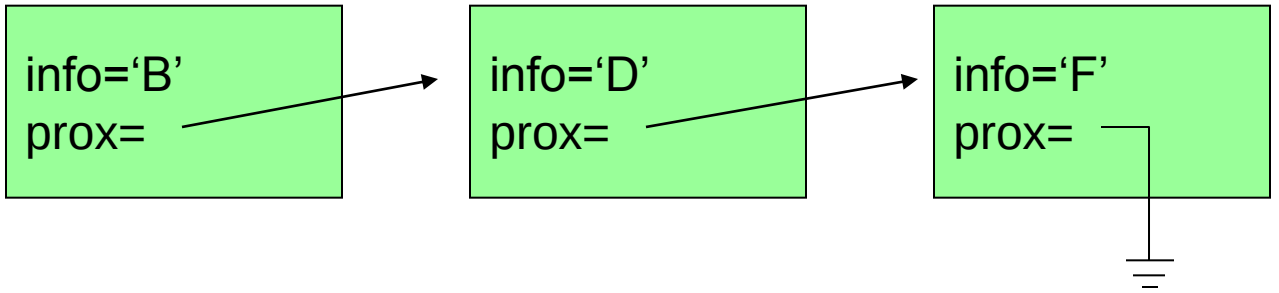
```
int busca_binaria(int a[], int inicio, int fim, int chave) {
   int meio;

   if (fim < inicio)
       return -1;
   meio = (inicio + fim) / 2;

   if (chave < a[meio])
       return busca_binaria(a, inicio, meio - 1, chave);
   else if (chave > a[meio])
       return busca_binaria(a, meio + 1, fim, chave);
   else if (chave == a[meio])
       return meio;
}
```

Lista Encadeada

- Imagine que você tem declarado vários blocos de memória para a seguinte estrutura:

```
struct bloco {
     char info;
     struct bloco *prox;
}
```

Cada bloco aponta para o endereço do próximo bloco alocado. Por exemplo:



- Faça uma função recursiva que imprima os dados armazenados (considere que os dados já estão lidos e alocados na memória).

Solução

```
typedef struct bloco {
   char info;
   struct bloco *prox;
} no;

typedef struct {
   no *inicio, *fim;
} Lista;
```

```
void imprimir_rec(no *p) {
   if (p != NULL) {
        printf("%c ", p->info);
        imprimir_rec(p->prox);
   }
}
```

Permutações

- Pense em um algoritmo que obtenha todas as permutações de uma dada string.

- Ex.: Para a string “ABC”, as permutações são:

ABC

ACB

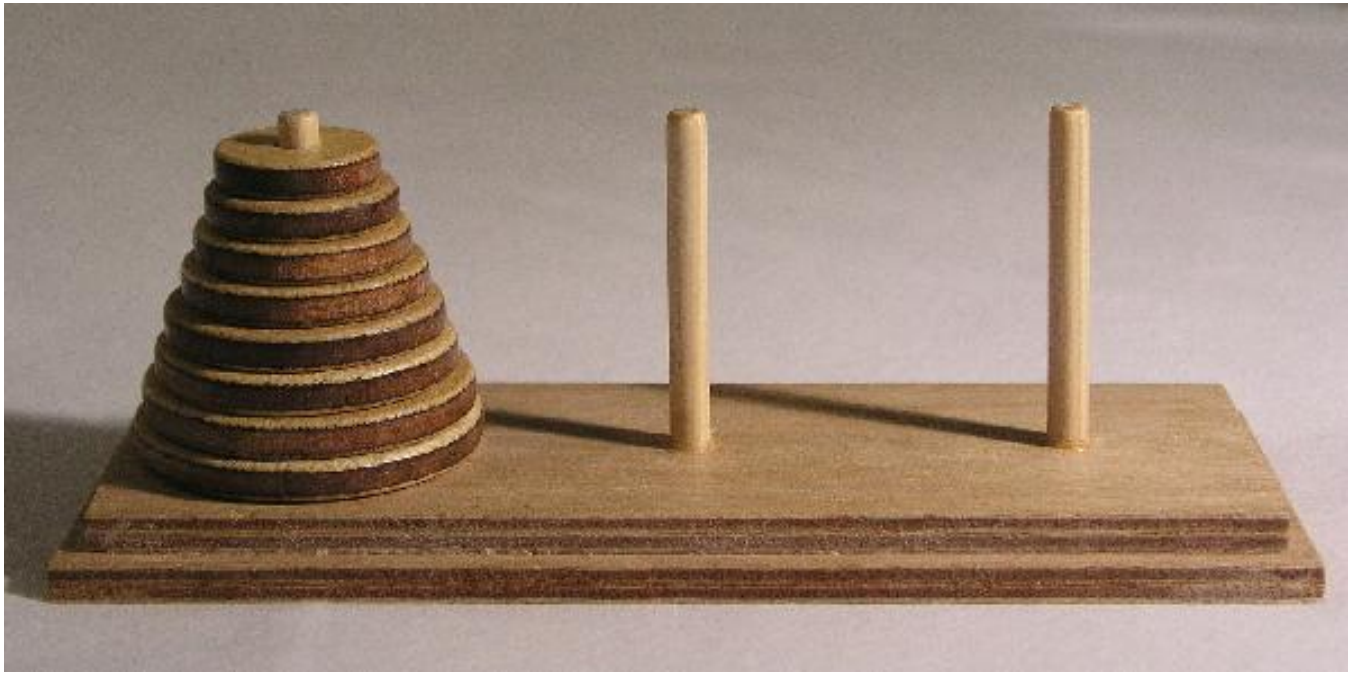
BAC

BCA

CBA

CAB

Torres de Hanoi



Torres de Hanoi

- Tradicionalmente com 3 hastes: Origem, Destino, Temporária
- Número qualquer de discos de tamanhos diferentes na haste Origem, dispostos em ordem de tamanho: os maiores embaixo
- Objetivo: usando a haste Temporária, movimentar um a um os discos da haste Origem para a Destino, sempre respeitando a ordem de tamanho
  - Um disco maior não pode ficar sobre um menor!

Torres de Hanoi

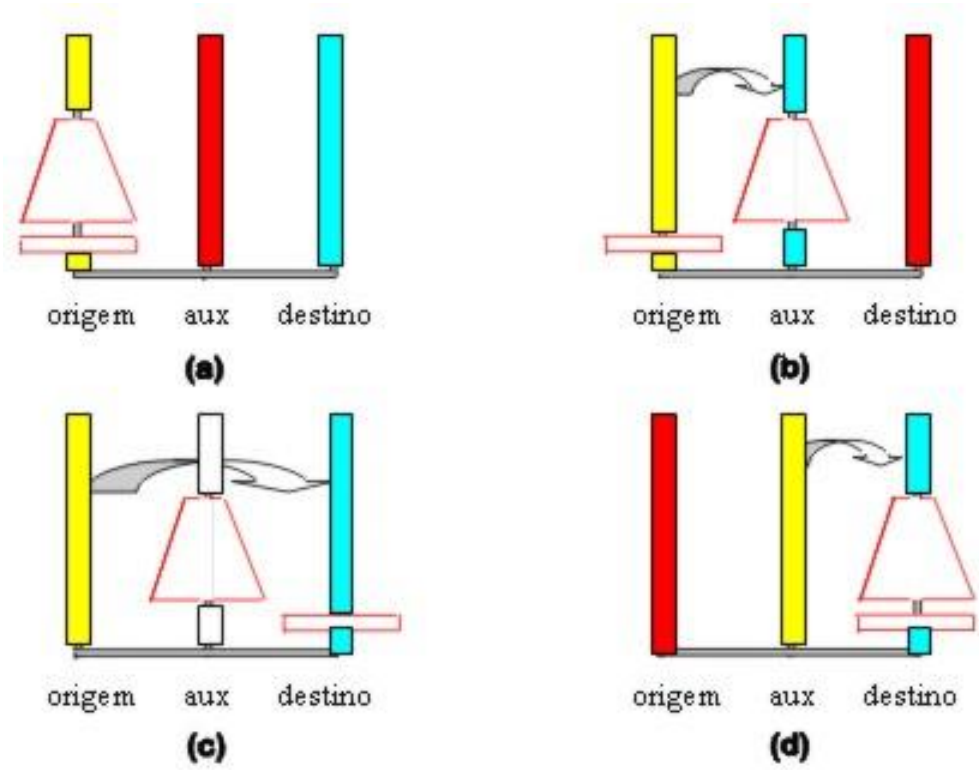
- Exemplo

http://www.mazeworks.com/hanoi/

Torres de Hanoi

- Implemente uma função recursiva que resolva o problema das Torres de Hanoi

Torres de Hanoi



(a) Estado inicial

(b) Mover n-1 discos da haste Origem para a haste Temporária (aux)

(c) Mover o disco n da haste Origem para a haste Destino

(d) Recomeçar, movendo os n-1 discos da haste Temporária (aux) para a haste Destino

Torres de Hanoi

```
#include <stdio.h>

void mover(int, char, char, char);

int main(void) {
    mover(3, 'O', 'T', 'D');
    return 0;
}

void mover(int n, char Orig, char Temp, char Dest) {
    if (n==1)
        printf("Mova o disco 1 da haste %c para a haste %c\n", Orig, Dest);
    else {
        mover(n - 1, Orig, Dest, Temp);
        printf("Mova o disco %d da haste %c para a haste %c\n", n, Orig, Dest);
        mover(n - 1, Temp, Orig, Dest);
    }
}
```

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

n=2; Orig=´O´;Temp=´D´;Dest=´T´
mover(1,´O´,´T´,´D´);

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

n=2; Orig=´O´;Temp=´D´;Dest=´T´
mover(1,´O´,´T´,´D´);

n=1; Orig=´O´;Temp=´T´;Dest=´D´
“Mova o disco 1 da haste ´O´ para a haste ´D´”

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

n=2; Orig=´O´;Temp=´D´;Dest=´T´
mover(1,´O´,´T´,´D´);

n=1; Orig=´O´;Temp=´T´;Dest=´D´
“Mova o disco 1 da haste ´O´ para a haste ´D´”

“Mova o disco 2 da haste ´O´ para a haste ´T´”
mover(1,´D´,´O´,´T´);

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

n=2; Orig=´O´;Temp=´D´;Dest=´T´
mover(1,´O´,´T´,´D´);

n=1; Orig=´O´;Temp=´T´;Dest=´D´
“Mova o disco 1 da haste ´O´ para a haste ´D´”

“Mova o disco 2 da haste ´O´ para a haste ´T´”
mover(1,´D´,´O´,´T´);

n=1; Orig=´D´;Temp=´O´;Dest=´T´
“Mova o disco 1 da haste ´D´ para a haste ´T´”

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

n=2; Orig=´O´;Temp=´D´;Dest=´T´
mover(1,´O´,´T´,´D´);

n=1; Orig=´O´;Temp=´T´;Dest=´D´
“Mova o disco 1 da haste ´O´ para a haste ´D´”

“Mova o disco 2 da haste ´O´ para a haste ´T´”
mover(1,´D´,´O´,´T´);

n=1; Orig=´D´;Temp=´O´;Dest=´T´
“Mova o disco 1 da haste ´D´ para a haste ´T´”

“Mova o disco 3 da haste ´O´ para a haste ´D´
mover(2,´T´,´O´,´D´);

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

n=2; Orig=´O´;Temp=´D´;Dest=´T´
mover(1,´O´,´T´,´D´);

n=1; Orig=´O´;Temp=´T´;Dest=´D´
“Mova o disco 1 da haste ´O´ para a haste ´D´”

“Mova o disco 2 da haste ´O´ para a haste ´T´”
mover(1,´D´,´O´,´T´);

n=1; Orig=´D´;Temp=´O´;Dest=´T´
“Mova o disco 1 da haste ´D´ para a haste ´T´”

“Mova o disco 3 da haste ´O´ para a haste ´D´
mover(2,´T´,´O´,´D´);

n=2; Orig=´T´;Temp=´O´;Dest=´D´
mover(1,´T´,´O´,´D´);

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

n=2; Orig=´O´;Temp=´D´;Dest=´T´
mover(1,´O´,´T´,´D´);

n=1; Orig=´O´;Temp=´T´;Dest=´D´
“Mova o disco 1 da haste ´O´ para a haste ´D´”

“Mova o disco 2 da haste ´O´ para a haste ´T´”
mover(1,´D´,´O´,´T´);

n=1; Orig=´D´;Temp=´O´;Dest=´T´
“Mova o disco 1 da haste ´D´ para a haste ´T´”

“Mova o disco 3 da haste ´O´ para a haste ´D´
mover(2,´T´,´O´,´D´);

n=2; Orig=´T´;Temp=´O´;Dest=´D´
mover(1,´T´,´O´,´D´);

n=1; Orig=´T´;Temp=´O´;Dest=´D´
“Mova o disco 1 da haste ´T´ para a haste ´O´”

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

n=2; Orig=´O´;Temp=´D´;Dest=´T´
mover(1,´O´,´T´,´D´);

n=1; Orig=´O´;Temp=´T´;Dest=´D´
“Mova o disco 1 da haste ´O´ para a haste ´D´”

“Mova o disco 2 da haste ´O´ para a haste ´T´”
mover(1,´D´,´O´,´T´);

n=1; Orig=´D´;Temp=´O´;Dest=´T´
“Mova o disco 1 da haste ´D´ para a haste ´T´”

“Mova o disco 3 da haste ´O´ para a haste ´D´
mover(2,´T´,´O´,´D´);

n=2; Orig=´T´;Temp=´O´;Dest=´D´
mover(1,´T´,´O´,´D´);

n=1; Orig=´T´;Temp=´O´;Dest=´D´
“Mova o disco 1 da haste ´T´ para a haste ´O´”

“Mova o disco 2 da haste ´T´ para a haste ´D´”
mover(1,´O´,´T´,´D´);

mover(3,´O´,´T´,´D´);

n=3; Orig=´O´;Temp=´T´;Dest=´D´
mover(2,´O´,´D´,´T´);

n=2; Orig=´O´;Temp=´D´;Dest=´T´
mover(1,´O´,´T´,´D´);

n=1; Orig=´O´;Temp=´T´;Dest=´D´
“Mova o disco 1 da haste ´O´ para a haste ´D´”

“Mova o disco 2 da haste ´O´ para a haste ´T´”
mover(1,´D´,´O´,´T´);

n=1; Orig=´D´;Temp=´O´;Dest=´T´
“Mova o disco 1 da haste ´D´ para a haste ´T´”

“Mova o disco 3 da haste ´O´ para a haste ´D´
mover(2,´T´,´O´,´D´);

n=2; Orig=´T´;Temp=´O´;Dest=´D´
mover(1,´T´,´O´,´D´);

n=1; Orig=´T´;Temp=´O´;Dest=´D´
“Mova o disco 1 da haste ´T´ para a haste ´O´”

“Mova o disco 2 da haste ´T´ para a haste ´D´”
mover(1,´O´,´T´,´D´);

n=1; Orig=´O´;Temp=´T´;Dest=´D´
“Mova o disco 1 da haste ´O´ para a haste ´D´”

Torres de Hanoi

- Desafio para casa
  - Tente fazer a versão não recursiva do programa

CRÉDITOS

Parte do material gentilmente cedido pelo Prof. Thiago A. S. Pardo.