Licenciatura em Engenharia Informática – 19/20

Programação

6: Recursividade

Francisco Pereira (xico@isec.pt)

Exemplo

- Calcular o fatorial de um inteiro positivo

```
0! = 1

1! = 1

n! = n × (n-1)!, se n > 1
```

Definição recursiva

Funções Recursivas

- Há problemas que são definidos naturalmente de forma recursiva

- Nestes casos, pode ser mais simples escrever funções recursivas:
  - Funções que se chamam a si próprias

Função Fatorial

Condição de paragem

```
int fatorial(int val)
{
   if(val <= 1)
       return 1;
   else
       return val * fatorial(val-1);
}
```

Chamada recursiva

Recursividade: Programação Preguiçosa

Instituto Superior de Engenharia de Coimbra

- A tarefa a resolver é demasiado complicada
  - Qual é a situação mais simples de todas?
  - Como é que a tarefa complicada pode ser ligeiramente simplificada?

Estrutura de uma função recursiva

- Condição de paragem:
  - Termina as chamadas recursivas
  - Surge antes da chamada recursiva
- Chamada recursiva:
  - A tarefa a resolver na nova chamada deve ser mais simples
  - O caminho percorrido pelas chamadas recursivas termina na condição de paragem

Função recursivas incorretas

```
int fatorial(int val)
{
   return val * fatorial(val-1);
   if(val <= 1) return 1;
}
```

```
int fatorial(int val)
{
   if(val <= 1) return 1;
   else return val * fatorial(val);
}
```

```
int fatorial(int val)
{
   if(val <= 1) return 1;
   else return val * fatorial(val+1);
}
```

Vantagens e desvantagens

- Vantagens:
  - Código mais compacto
  - Fácil de escrever e de compreender
- Desvantagens:
  - Ocupação intensiva de recursos (pilha do programa)

Alguns Exemplos

Instituto Superior de Engenharia de Coimbra

- Manipulação de strings
- Operações sobre listas ligadas
- Manipulação de vetores
- Puzzle das Torres de Hanói

Exemplo 1: Contar

- Calcular o tamanho de uma string

```
int conta(char *st)
{
   if(*st == '\0')
       return 0;
   else
       return 1+conta(st+1);
}
```

Exemplo 2: Inverter

- Escrever uma string por ordem inversa

```
void puts_inv(char *st)
{
   if(*st == '\0')
       return;
   else
   {
       puts_inv(st+1);
       putchar(*st);
   }
}
```

Exemplo 3: Capicua

- Verificar se uma string é capicua

```
int capicua(char *st, int tam)
{
   if(tam <= 1)
       return 1;
   else if(*st != *(st+tam-1))
       return 0;
   else
       return capicua(st+1, tam-2);
}
```

2 condições de paragem

Manipulação de listas ligadas

- Operações a implementar:
  1. Contar o número de nós
  2. Escrever a informação por ordem inversa

```
typedef struct numero no, *pno;

struct numero {
   int val;
   pno prox;
};
```

Operação 1: Contar

Instituto Superior de Engenharia de Coimbra

```
int conta_nos(pno p)
{
   if(p == NULL)
       return 0;
   else
       return 1 + conta_nos(p->prox);
}
```

Operação 2: Inverter

```
void escreve_inv(pno p)
{
   if(p == NULL)
       return;
   else
   {
       escreve_inv(p->prox);
       printf("%d\t", p->val);
   }
}
```

Exercícios: Outras Funções

Instituto Superior de Engenharia de Coimbra

1. Calcular a potência de um número inteiro
   - Devolve o valor calculado
2. Verificar se um vetor de inteiros está ordenado de forma crescente
   - Devolve 1 se estiver ordenado (0, caso contrário)
3. Verificar se duas palavras são iguais
   - Devolve 1 se forem iguais (0, caso contrário)

Problema dos Ursos de Peluche

Uma criança recebe N ursos

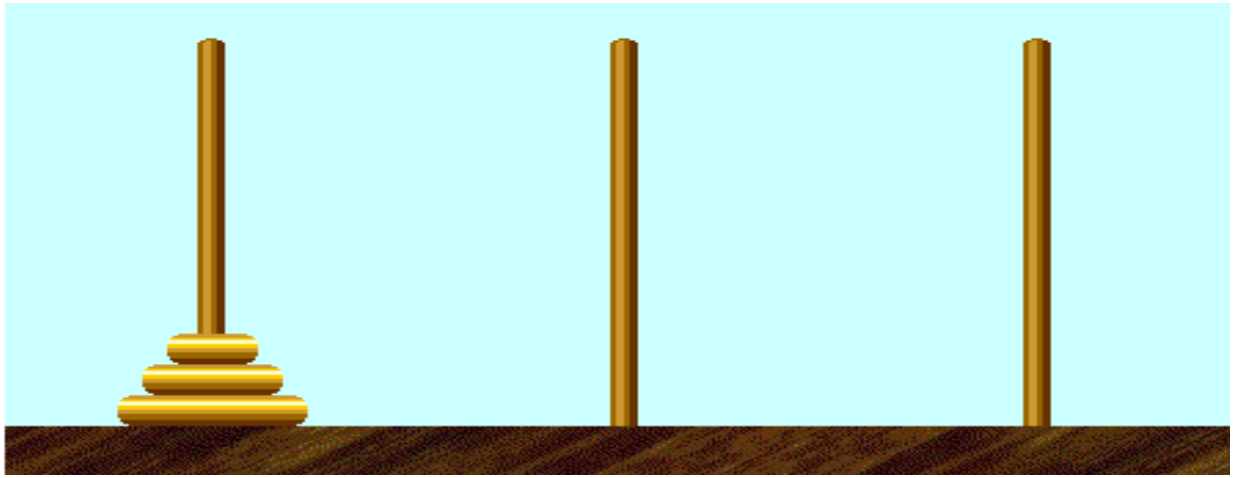
Pode repetir os seguintes movimentos:

- Se N for divisível por 2, devolve N/2 ursos
- Se N for divisível por 3 ou por 4, multiplica os últimos 2 dígitos de N e devolve esse valor
- Se N for divisível por 5, devolve 42 ursos

Que valores iniciais de N garantem que a criança consiga ficar com exatamente 42 ursos?

Torres de Hanói

- Puzzle surgido no final do século XIX, inspirado numa lenda Hindu:
  - 3 torres A, B, C e vários discos de diâmetro diferente



  - Objectivo: Mover todos os discos da torre A para a torre C

Torres de Hanói: Regras e Restrições

- Regra do jogo:
  - Em cada passo só é possível mover o disco que está no topo de qualquer uma das torres
  - Restrição:
    - Um disco só pode ser colocado numa torre vazia ou sobre outro de diâmetro superior
  - De acordo com a lenda, os monges tinham que resolver um puzzle com 64 discos:
    - Número mínimo de jogadas é igual a $(2^{64}–1)$!!!

Torres de Hanói: Resolução

- Passo 1:
  - Resolver no papel (ou na animação flash) um problema com 4 discos
- Passo 2:
  - Implementar uma função não-recursiva em C que resolva o problema para qualquer número de discos.
  - A função deve indicar a sucessão de passos até atingir a solução final

Torres de Hanói: Resolução

- Passo 3:
  - Implementar uma função recursiva em C que resolva o problema para qualquer número de discos.
  - A função deve indicar a sucessão de passos até atingir a solução final

Torres de Hanói: Função recursiva

- Mover n discos da torre A para a torre C
  - Caso particular: Se n=1 então a solução é trivial
  - Caso geral: A solução para n discos (n≠1) pode ser obtida a partir da solução para n-1 discos.
    - Deslocar os n-1 discos que se encontram em cima de A para B utilizando C como auxiliar
    - Deslocar o disco restante de A para C
    - Deslocar os n-1 discos de B para C utilizando A como auxiliar

Torres de Hanói: Função recursiva

- Considera-se que:
  - Os discos estão numerados de 1 a n (sendo 1 o menor)
  - As torres são identificadas pelos caracteres A, B, C
  - A função escreve no monitor quais os movimentos efetuados
  - Recebe como argumentos:
    - Número de discos a movimentar
    - Torre inicial
    - Torre final
    - Torre auxiliar

Torres de Hanói: Função recursiva

```
#include <stdio.h>

void hanoi(int n, char or, char dest, char aux)
{
   if(n == 1)
       printf("Disco %d: %c -> %c\n", n, or, dest);
   else {
       hanoi(n-1, or, aux, dest);
       printf("Disco %d: %c -> %c\n", n, or, dest);
       hanoi(n-1, aux, dest, or);
   }
}

int main()
{
   int n;

   printf("Numero de discos: ");
   scanf("%d", &n);
   hanoi(n, 'A', 'C', 'B');
   return 0;
}
```