MC-102 — Aula 16
Recursão

Instituto de Computação – Unicamp

Segundo Semestre de 2009

Roteiro

Recursividade

Motivação

Programas recursivos são, em geral, mais simples de escrever, analisar e entender

Recursividade

Um objeto é dito recursivo se pode ser definido em termos de si próprio.

Recursão

é o processo de resolução de um problema, reduzindo-o em um ou mais subproblemas com as seguintes caracteríticas:

1 - São idênticos aos problemas originais;

2 - São mais simples de resolver.

Recursividade

- Uma vez realizada a primeira subdivisão, a mesma técnica de decomposição é usada para dividir cada subproblema
- Eventualmente, os subproblemas tornam-se tão simples que é possível resolvê-los sem efetuar novas subdivisões
- A solução completa do problema original é obtida através da "montagem" das soluções componentes
- Este processo de resolução de problemas esté diretamente ligado ao conceito de indução matemática.
- A indução fraca toma como hipótese a idéia de que a solução de um problema de tamanho t pode ser obtida a partir da solução de subproblemas de tamanho $t - 1$

Componentes da Recursão

Toda recursão é composta por

- Um caso base, uma instância do problema que pode ser solucionada facilmente. Por exemplo, é trivial fazer a soma de uma lista com um único elemento.
- Uma ou mais chamadas recursivas, onde o objeto define-se em termos de sí próprio, tentando convergir para o caso base. A soma de uma lista de n elementos pode ser definida a partir da lista da soma de $n - 1$ elementos.

Recursão na matemática

Como definir recursivamente a soma abaixo?

$$\sum_{k=m}^{n} k = m + (m+1) + \cdots + (n-1) + n$$

Recursão na matemática

Definição recursiva

$$\sum_{k=m}^{n} k = \begin{cases} m & \text{se } n = m \\ \sum_{k=m}^{n-1} k + n & \text{se } n > m \end{cases}$$

Recursão na matemática

Como definir recursivamente o fatorial abaixo?

$$n! = \prod_{k=1}^{n} k = n \times (n-1) \times (n-2) \times ... \times 1$$

Recursão na matemática

Definição recursiva

$$n! = \prod_{k=1}^{n} k = \begin{cases} 1 & \text{se } n = 1; n = 0 \\ \prod_{k=m}^{n-1} k * n & \text{se } n > 1 \end{cases}$$

ou

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

Recursão na computação

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

Veja o código: `fat-recursivo.c` e a versão iterativa fat.c

Pilha de execução

- A cada chamada de função o sistema reserva espaço para parâmetros, variáveis locais e valor de retorno.

main	s
	ret: ??
soma	m: 5
	n: 10
	ret: ??
soma	m: 6
	n: 10
	...

Estouro de pilha de execução

"To understand recursion you must first understand recursion."

- O que acontece se a função não tiver um caso base?
- O sistema de execução não consegue implementar infinitas chamadas. (Lembre-se, somente Chuck Norris conta até o infinito).

Veja o código rec-infinita.c

Potência

$$x^n = \begin{cases} \frac{1}{x^{(-n)}} & \text{se } n < 0 \\ 1 & \text{se } n = 0 \\ x.x^{(n-1)} & \text{se } n > 0 \end{cases}$$

```
double pot(double x, int n) {
  if (n == 0) return 1;
  else if (n < 0)
         return 1/pot(x, -n);
       else
         return x*pot(x, n-1);
}
```

Ver: potencia-rec.c e sua versão iterativa pot-iterativa.c

Número de dígitos de um inteiro

```
int num_digitos_rec(int n) {
    if (abs(n)<=9)
        return(1);
    else
        return (1+ num_digitos_rec(n/10));
}
```

Ver: digitos-rec.c e sua versão iterativa digitos.c