UFOP

DEPARTAMENTO DE COMPUTAÇÃO

Recursividade

Algoritmos e Estruturas de Dados I

DECOM – UFOP

Conceito de Recursividade

- Fundamental em Matemática e Ciência da Computação
  - Um programa recursivo é um programa que chama a si mesmo
  - Uma função recursiva é definida em termos dela mesma
- Exemplos
  - Números naturais, Função fatorial, Árvore
- Conceito poderoso
  - Define conjuntos infinitos com *comandos* finitos

Recursividade

- A recursividade é uma estratégia que pode ser utilizada sempre que o cálculo de uma função para o valor n, pode ser descrita a partir do cálculo desta mesma função para o termo anterior (n-1).

Exemplo – Função fatorial:

$n! = n * (n-1) * (n-2) * (n-3) * \ldots * 1$

$(n-1)! = (n-1) * (n-2) * (n-3) * \ldots * 1$

logo:

$n! = n * (n-1)!$

Recursividade

- Definição: 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 uma função B que, por sua vez, chama A

Condição de parada

- Nenhum programa nem função pode ser exclusivamente definido por si
  - Um programa seria um loop infinito
  - Uma função teria definição circular
- Condição de parada
  - Permite que o procedimento pare de se executar
  - *F(x)* > 0 onde *x* é decrescente
- Objetivo
  - Estudar recursividade como ferramenta *prática*!

Recursividade

- Para cada chamada de uma função, recursiva ou não, os parâmetros e as variáveis locais são empilhados na pilha de execução.

Execução

- Internamente, quando qualquer chamada de função é feita dentro de um programa, é criado um **Registro de Ativação** na **Pilha de Execução** do programa
- O registro de ativação armazena os parâmetros e variáveis locais da função bem como o "ponto de retorno" no programa ou subprograma que chamou essa função.
- Ao final da execução dessa função, o registro é desempilhado e a execução volta ao subprograma que chamou a função

Exemplo

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


Main() {
   int f;
   f = fat(5);
    printf("%d",f);
}
```

Quando vale a pena usar recursividade

- Recursividade vale a pena para 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)

Dividir para Conquistar

- Duas chamadas recursivas
  - Cada uma resolvendo a metade do problema
- Muito usado na prática
  - Solução eficiente de problemas
  - Decomposição
- Não se reduz trivialmente como fatorial
  - Duas chamadas recursivas
- Não produz recomputação excessiva como fibonacci
  - Porções diferentes do problema

Exercício

- Crie uma função recursiva que calcula a potência de um número:
  - Como escrever a função para o termo n em função do termo anterior?
  - Qual a condição de parada?
- Qual a complexidade desta função?

Função de Potência Recursiva

```
int pot(int base, int exp)
{
  if (!exp)
    return 1;

  /* else */
  return (base*pot(base, exp-1));
}
```

Análise de complexidade:

$T(0) = 1;$

$T(b,n) = 1 + T(b,n-1);$

O(n)

Exercícios

- Implemente uma função recursiva para computar o valor de 2^n

- O que faz a função abaixo?

```
void f(int a, int b) { // considere a > b
     if (b = 0)
        return a;
     else
        return f(b, a % b);
}
```

Respostas

```
Pot(int n) {
    if (n==0)
        return 1;
    else
        return 2 * Pot(n-1);
}
```

- Algoritmo de Euclides. Calcula o MDC (máximo divisor comum) de dois números a e b