LP – Linguagem de Programação I

Conceito básico da recursividade

Uma função diz-se recursiva quando ela se chama a si própria, direta ou indiretamente (por intermédio de outras funções). A recursividade permite implementar algoritmos que decompõem o problema a resolver numa sucessão de problemas cada vez mais simples com a mesma estrutura do problema original.
Como uma função recursiva pode chamar-se a si mesma infinitamente, é essencial que exista uma condição que determine o fim do processo recursivo. Esta condição determina o momento em que a função deverá parar de se chamar a si mesma. Para se conseguir implementar uma solução recursiva estrutura-se normalmente a função correspondente em duas partes fundamentais:

1. **O caso elementar ou condição de paragem**. O caso elementar é resolvido sem utilização de recursividade, sendo este ponto geralmente um limite superior ou inferior da regra geral. Dado que não existe a necessidade de se chamar uma nova instancia da função, diz-se que este caso corresponde ao ponto de paragem da recursividade.
2. **Regra Geral.** A regra geral da recursividade simplifica a resolução do problema através da utilização recursiva de casos sucessivamente mais simples do problema. O processo repete-se até se atingir o caso elementar que determina o ponto de paragem de todo o processo.

É importante entender que quando uma função se chama a ela própria, passa a ser executada uma nova cópia da função, o que implica que as suas variáveis locais são independentes das variáveis locais da primeira cópia, e não podem relacionar-se diretamente.

Existem algumas vantagens e desvantagens na utilização de métodos baseados na recursividade.
Algumas delas são sumariadas a seguir:

Vantagens:

- A utilização de uma função recursiva pode simplificar a solução de alguns problemas;
- Pode-se obter um código mais conciso e eficaz nessas situações;
- Uma solução recursiva pode, por outro lado, eliminar a necessidade de manter o controle manual sobre uma série de variáveis normalmente associadas aos métodos alternativos à recursividade.

Desvantagens:

- As funções recursivas são geralmente mais lentas e ocupam mais memória do que as funções iterativas equivalentes, uma vez que são feitas muitas chamadas consecutivas a funções;
- Um erro de implementação pode levar ao esgotamento dos recursos associados à pilha que gere a chamada a funções. Isto é, caso não seja indicada nenhuma condição de paragem, ou essa condição foi definida de forma errada e nunca será satisfeita, então o processo recursivo não terá fim.

Exemplo clássico da recursividade

Para ajudar a compreender o funcionamento de uma função recursiva utiliza-se geralmente o exemplo do cálculo do *fatorial* de um número inteiro positivo. O fatorial pode ser definido em função do fatorial do número anterior o que torna a sua implementação um caso típico da utilização de métodos recursivos.

Para um dado número inteiro positivo *n* podemos definir o fatorial como:
n! = n*(n-1)*(n-2)*…*2*1 ou seja n! = n*(n-1)!

Esta função pode ser definida recursivamente em função de um caso elementar e de uma forma geral:
0! = 1 (por definição, caso mais simples)

n! = n * (n - 1)! para n>0 (formula recursiva)

Uma possível implementação em C de uma função que calcula o fatorial de um dado número *N* é apresentada a seguir.

```
long Fatorial(int N) {
      if (N == 0) // Caso base que define a condição de paragem
            return 1;
      else
            return ( N * Fatorial ( N-1 ) );
}
```

Note que, enquanto *N* não for igual a 0, a função *Fatorial* chama-se a si mesma com um valor cada vez menor até chegar a 0. A condição (*N==0*) representa o critério de paragem.
Há certos algoritmos que são mais eficientes quando implementados de maneira recursiva. A utilização de métodos recursivos tende a consumir mais memória e a ser mais lenta. Lembre-se que memória é consumida cada vez que o computador faz uma chamada a uma função. A utilização de funções recursivas sem controlo pode esgotar rapidamente a memória do computador.

Cuidados a ter com funções recursivas.

Se num determinado método recursivo não tiver sido definida uma condição de paragem adequada poderão surgir alguns problemas. Na função definida anteriormente para o cálculo do fatorial, o que aconteceria se por engano fosse passado como parâmetro o valor -1? A função entraria num ciclo recursivo infinito pois nunca mais a condição de paragem seria válida. O mais certo seria o programa originar um erro de falta de memória da pilha que gere as chamadas a funções (*stack overflow*). Para evitar este tipo de problemas, teremos que garantir que a condição de paragem utilizada, lida com todas as situações não desejadas. Uma possível solução para o exemplo do cálculo do factorial seria a reformulação da condição de paragem:

```
long Fatorial(int N) {
      if (N > 1)
            return ( N * Fatorial ( N-1 ) );
      else
            return 1;
}
```

Desenho de algoritmos recursivos:

1. Obter uma solução de como o problema pode ser dividido em passos menores
2. Definir um ponto de paragem
3. Definir uma regra geral

4. Verificar se o algoritmo termina
5. Desenhar uma árvore de execução do programa, mostrando o desenvolvimento do processo.

Referências Bibliográficas

Faculdade de Ciências e Tecnologia da Universidade de Coimbra, ESTRUTURAS DE DADOS E ALGORITMOS, Ficha 3, 2004