PROGRAMAÇÃO E ALGORITMOS (LEI)

Universidade da Beira Interior, Departamento de Informática
Hugo Pedro Proença, 2016/2017

Resumo

- Recursividade
  - Definição
  - Tipos de Recursividade
  - Exemplos
  - Exercícios

Recursividade

- Em linguagens imperativas e estruturadas, o conceito base é a **função:**
  - Unidade lógica de código (conjunto finito de blocos) que desempenha determinada funcionalidade.
  - Função para calcular o máximo de um vector, mostrar resultados no écran, pedir dados ao utilizador,…
- A linguagem C é um exemplo clássico de uma linguagem de programação imperativa, estruturada em funções.
  - Motivo para desaconselhar a utilização de "**goto**"

Recursividade

- Ao mandar executar uma função, são reservados pelo sistema operativo recursos para as variáveis e instruções a executar.
- É conveniente pensar nas funções por nós programadas (no código-fonte), como algo que vai ser copiado e posteriormente executado.
  - O que é executado são cópias das "nossas" funções.
- De cada vez que é encontrada uma chamada a uma função, são reservados os recursos necessários e transferida a execução para as respectivas instruções.
- Quando a execução da função termina, regressa ao local a partir de onde a função foi chamada.

Recursividade

- Exemplo:

```
void fun1(){
    linha 1;
    linha 2;
}
void fun2(){
    linha3;
    fun1();
    linha 4;
}
main(){
    fun1();
    fun2();
}
```

- A sequência de instruções executadas será:
  - Linha 1, 2, 3, 1, 2, 4
- Cópias das linhas 1 e 2 são executadas 2 vezes!

Recursividade

- Em termos de definição, uma função diz-se recursiva se contiver dentro dela uma chamada à própria função:

```
void funcao(){
    linha 1;
    linha 2;
    funcao();
    linha 3;
    linha4;
}
```

  - Para este exemplo, qual a sequência de instruções (linhas) executadas?

Recursividade

- O exemplo anterior executaría indefinidamente as linhas 1 e 2, pelo que não podería sequer ser considerado um algoritmo (conjunto **finito** de passos ordenados).
- **Para uma função recursiva estar bem formada:**
  - **Tem que possuir um critério de paragem antes da chamada recursiva.**
  - **O valor de retorno (caso o tenha) da chamada recursiva tem que ser utilizado dentro da própria função.**
- O critério de paragem é uma condição que determina se a chamada recursiva é ou não feita.

Recursividade

- Critério de Paragem. Exemplo:

```
void funcao(int u){
        printf("%d",u);
        if (u<100)
                funcao(u+1);
}
```

- É importante garantir que o critério de paragem esteja colocado antes das chamadas recursivas e que seja sempre atingido.
- No exemplo acima, bastaría que em vez de u+1 estivesse u-1, para a função já não estar bem definida em termos recursivos.

Recursividade

- Exemplo de uma função recursiva para calcular o factorial de um número.
  - O factorial de 7=7*6*5*4*3*2*1
  - Como (6*5*4*3*2*1) é o factorial de 6, podemos concluir que:
    - Factorial 7= 7 * Factorial 6.

```
int factorial(int n){
    int x;
    if (n<=1)                    //critério de paragem
        return(1);
    x=factorial(n-1);            //chamada recursiva
    return(n*x);                 //utilização do valor recursivo
}
```

Recursividade

- As funções recursivas seguem normalmente uma estratégia de "Divide e Conquista".
  - Um problema pode ser decomposto em sub-problemas de menor dimensão.
  - Cada chamada à função recursiva deve fazer apenas uma pequena parte do trabalho, que será terminado pelas chamadas recursivas subsequentes.
  - Após todas as chamadas à função, o problema deverá ter sido tratado na globalidade.

Tipos de Recursividade

- **Recursividade Linear:** Diz-se que uma função é linearmente recursiva se em cada execução da função apenas existe uma chamada recursiva.

- funcao1()
  - funcao1()
    - funcao1()
      - funcao1()
        - funcao1()
          - funcao1()
            - funcao1()
              - ...

Recursividade

- **Recursividade Mútua** existe quando 2 funções se chamam entre si, apesar de eventualmente nenhuma delas ter uma chamada à própria função.
- Exemplo: Determinar se um número é par ou impar.

```
int par(){
    if (n==0)
                return(1);
    return(!impar(n-1))
}
int impar(){
    if (n==1)
                return(1);
    return(!par(n-1));
}
```

Recursividade

- Diz-se que uma função é de **Recursividade Exponencial** se for feita mais que uma chamada à própria função, durante a sua execução.
  - Leva normalmente a custos computacionais elevados.
  - Funcao1()
    - Funcao1()
      - Funcao1()
        - Funcao1()
        - Funcao1()
      - Funcao1()
        - Funcao1()
        - Funcao1()
    - Funcao1()
      - ...

Recursividade

- O termo **Recursividade Terminal** aplica-se quando o último passo feito por uma função é a chamada recursiva.
- Funcao 1()
  - Linhas 1,2,3
  - Funcao 1()
    - Linhas 1,2,3
    - Funcao1()
      - ...
- São normalmente funções que fácilmente se transformam em iterativas.