9ª Aula - Ordenação

Programação
Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@tecnico.ulisboa.pt

Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

Ordenação - Introdução

- **Ordenar valores**, quer **numéricos** quer **literais**, é uma tarefa frequente em muitos tipos de programas.
- Por vezes, o número de elementos a ordenar é **bastante grande** o que obriga a procurar **estratégias eficientes de ordenação**.
- O algoritmo mais simples de ordenação (embora pouco eficiente) designa-se por **bubble sort**[1] e consiste no seguinte (crescente):
  1. Começa-se nos **dois primeiros elementos**, se estiverem na **ordem certa**, **ficam** como estão; caso contrário **trocam-se**;
  2. Depois passa-se ao **segundo** e **terceiro repete-se a operação**;
  3. E assim sucessivamente até ao **último par**;
  4. Se **nenhuma troca foi feita acabou**, **senão** volta-se ao **início**.
- Se tivermos **N** elementos a ordenar, na pior das hipóteses, teremos de efectuar $\mathbf{N}^2$ comparações. O que é um número de operações bastante elevado para ordenar 1 milhão de valores.
- Um dos algoritmos de ordenação mais eficientes é o **quick sort**.

[1] Porque o método se assemelha às bolhas (bubble) a subirem nos líquidos

Ordenação - Exemplo (Prog08_03.c)

Seja um vetor **r1** que se pretende ordenar (método de **bubble sort**):

```
while (1)
  {
    t = 0;
    for (i1=1 ; i1<qt ;++i1)
      {
        if (r1[i1 - 1] > r1[i1])
          {
            r0 = r1[i1 - 1];
            r1[i1 - 1] = r1[i1];
            r1[i1] = r0;
            t = 1;
          }
      }
    if (t == 0) break;
  }
```

- Seja o **loop de ordenação**;
- Inicializa-se o **teste de troca**;
- Vai percorrer-se o vetor para testar eventuais trocas;
- Se r1[i1 - 1] > r1[i1] vão **trocar-se** os seus valores;
- Para trocar dois valores, usamos uma **variável intermédia**, depois trocam-se os valores;
- Havendo troca marca-se '**t=1**';
- Se houve trocas (**t=1**) **repete-se**; caso contrário **sai-se** do loop.

Ver: www.mycstutorials.com/articles/sorting/bubblesort

Ordenação - Exemplo 2 (Prog08_04.c)

- Sendo a **ordenação** uma **tarefa bem definida**, podemos criar uma **função** onde ela será realizada ('**Prog08_04.c**').
- O código é idêntico ao anterior, no entanto, vejamos como se escreve o **cabeçalho da função** e como ela **é chamada**.
- A função de ordenação **bubble_sort** irá receber dois argumentos **r1** (vetor dos números a ordenar) e **qt** (quantos são):

  void bubble_sort (**long int** ***r1**, **long int qt**) { ... }

  em que tipo **void** significa que **não é retornado qualquer valor**.
- Como vimos, quando temos uma **variável dimensionada**, o **nome** é o **ponteiro para o início da posição de memória** em que ela se encontra, assim, '**long int** ***r1**', quer dizer que '**r1**' é um **ponteiro para inteiros** e não um inteiro.
- Ou seja, se '**r1**' é um ponteiro para um '**long int**', então, '***r1**' é o valor do '**long int**' que se encontra na **posição** '**r1**'.
- Para chamarmos a função: **bubble_sort** (**r1**, **qt**);

Ordenação - Exemplo 2 (Prog08_04.c)

- Aproveitemos o que vimos sobre **valores** e **ponteiros** para construir a função **troca_valores** que terá por tarefa **trocar dois números**.
- Como vimos, se queremos que as **alterações nos valores** das variáveis **permaneçam após a execução** de uma função, devemos efectuá-las nas respectivas **posições de memória**.
- Temos **dois modos** equivalente para o dizer à **função**:
  1. Através do **ponteiro para** ('**&**'), ou seja se **r1[i1]** é o **valor do i1**^{esimo} **elemento**, o ponteiro para ele será '**&r1[i1]**':

     troca_valores (**&r1[i1 - 1]**, **&r1[i1]**);

  2. Se **r1** é o ponteiro para o primeiro elemento (o '**0**' não esquecer), então o ponteiro para o segundo será **r1+1** e, em geral, o ponteiro para o elemento **k** será **r1+k**[2]

     troca_valores (**r1+(i1-1)**, **r1+i1**);

[2] **Nota:** Nesta operação, o compilador de **C** verifica o espaço ocupado por um elemento e move-se por saltos desse tamanho.

Ordenação - Exemplo 2 (Prog08_04.c)

Vejamos como construir a função **troca_valores**:

- Ela não vai **retornar valores**, logo é do tipo **void**;
- Por outro lado recebe **dois ponteiros para inteiros**, logo, as suas declarações são do tipo '**long int *n1**';
- Se **n1** é o **ponteiro** para a memória, então ***n1**[3] será o seu **valor**, isto é, o '**número desejado**'.
- O código da função **troca_valores** será então:

```
void troca_valores (long int *n1, long int *n2)
{
   long int i1 ;
   i1 = (*n1);
   (*n1) = (*n2);
   (*n2) = i1;
}
```

[3] Não esquecer que '***ponteiro**' é o valor para o qual o ponteiro aponta

Ordenação - Exemplo 3 (Prog05_12.c)

- Podemos utilizar os conhecimentos que adquirimos de ordenação de números para os aplicar à **função logística**;
- No programa '**Prog05_11.c**' tinhamos calculado as órbitas periódicas num certo intervalo do parâmetro **r** $\in$ **[r1,r2]** e guardado os seus valores num vetor;
- Agora, no programa '**Prog05_12.c**', depois de calcularmos os **valores** para cada órbita periódica vamos **ordená-los**.
- Para tal, copiamos as **funções** que criámos no programa '**Prog08_04.c**' e alteramo-las para **ordenarem** reais em **dupla precisão** (**double**).

Ordenação - Exemplo 4 (Prog08_05.c)

- Como último exemplo, podemos ver o programa '**Prog08_05.c**' idêntico a '**Prog08_04.c**' em que se substituiu o método '**bubble sort**' por '**quick sort**'.
- A comparação dos tempos de cálculo dos dois métodos é bem evidente se usarmos mais de 100 000 números aleatórios.
- A biblioteca de **C** dispõe de uma função de ordenação que usa o método '**quick sort**'. Essa função designa-se por '**qsort**'.
- Podem encontrar-se exemplificações animadas em:
  - **http://en.wikipedia.org/wiki/Bubble_sort**
  - **http://en.wikipedia.org/wiki/Quick_sort**