Introdução à Programação

1.Ano LCC-MIERSI
DCC - FCUP

Nelma Moreira

Aula 10

Ordenação

2 15 -1 7 9 4 6

Existem muitos métodos de ordenar

* Bolha
* Inserção
* Seleção
* Quicksort
* etc...

Bolha (bubble sort)

* Percorrer os elementos da sequência e comparar pares de elementos consecutivos trocando-os se não estiverem ordenados.
* Ao fim de uma iteração o elemento maior é encontrado (e está na última posição).
* Repetir o processo até todos os elementos estarem ordenados (no máximo n vezes).

```
para i=0,... n-1 e
 enquanto houver trocas
  para j=0,...,n-i-2
   se a[j]>a[j+1] trocar
      a[j] com a[j+1]
```

```
void bolha (int a[],int n) {
 int i, k, aux,troca = TRUE;

 for(i=0;troca == TRUE && i < n-1;
                        i++) {
   troca = FALSE;
   for(k=0; k < n-1-i; k++)
    if (a[k] > a[k+1]) {
      troca = TRUE;
      aux = a[k];
      a[k] = a[k+1];
      a[k+1] = aux;
    }
 }
```

Execução:

```
2  15  -1  7  9  4  6
2  -1  7  9  4  6  15
-1  2  7  4  6  9  15
-1  2  4  6  7  9  15
-1  2  4  6  7  9  15
```

Inserção

* É o método usado para ordenar em jogos de cartas.
* Introduzir cada elemento `a[i]`, na subsequência ordenada de

`a[0], ..., a[i-1]`

de modo a mantê-la ordenada.

```
para i=1,... n-1
  faça x=a[i]
  inserir x na posição
  correcta entre as posições
  0 e i-1
```

```
void insercao (int a[],int n) {
  int i, j, aux;
  for (i=1; i < n; i++) {
    aux = a[i];
    for(j=i-1;
        j >= 0 && a[j] > aux;
        j--) a[j+1] = a[j];
    a[j+1] = aux;
  }
}
```

Execução:

```
2  15  -1  7  9  4  6
2  15  -1  7  9  4  6
-1  2  15  7  9  4  6
-1  2  7  15  9  4  6
-1  2  7  9  15  4  6
-1  2  4  7  9  15  6
-1  2  4  6  7  9  15
```

Seleção

* selecionar sucessivamente o menor elemento da sequência e colocá-lo na primeira posição não ordenada.

* Supondo que de `v[0]` a `v[i-1]` está ordenado, procurar de `v[i+1]` a `v[n-1]` o elemento menor e trocá-lo com `v[i]`;

```
Para i=0,..., n-2:
  para j=i+1, ..., n-1
   procurar o elemento
   menor e trocá-lo com
   a[i];
```

```
void selord(int a[], int n) {
  int i;
  for(i = 0;i < n-1; i++){
    int j, min = i, aux;
    for(j = i + 1; j < n; j++)
      if(a[j] < a[min]) min = j;
    aux = a[i];
    a[i] = a[min];
    a[min]= aux;
  }
}
```

Execução:

```
 2 15 -1  7  9  4  6
-1 15  2  7  9  4  6
-1  2 15  7  9  4  6
-1  2  4  7  9 15  6
-1  2  4  6  9 15  7
-1  2  4  6  7 15  9
-1  2  4  6  7  9 15
```

Explicação mais detalhada:

```
i      0  1   2  3  4  5  6
a[i]   2  15 -1  7  9  4  6
-----------------------------------------------------
a[i]   2  15 -1  7  9  4  6        0..6: min=-1
a[i] -1| 15  2  7  9  4  6         1..6: min=2
a[i] -1  2| 15  7  9  4  6         2..6: min=4
a[i] -1  2  4|  7  9 15  6         3..6: min=6
a[i] -1  2  4  6|  9 15  7         4..6: min=7
a[i] -1  2  4  6  7|15  9          5..6: min=9
a[i] -1  2  4  6  7  9|15
-----------------------------------------------------
```

Eficiência

* Para estimar a eficiência, podemos contar o número `c(n)` de vezes que a comparação da instrução

  `if(v[j]<v[min])...`

  é efectuada (porque é a mais frequente):

$$c(n) = (n - 1) + (n - 2) + \ldots + 1 =$$
$$= n(n - 1)/2 = (n^2 - n)/2$$

* Dizemos que se trata de um algoritmo de ordem $O(n^2)$
* Esta é a **complexidade** da generalidade dos algoritmos de ordenação, embora haja alguns mais eficientes

Variáveis e Funções : âmbito e classes

Âmbito de variáveis

* O **âmbito** duma variável é a área de programa em que ela é acessível.

* As variáveis podem ser:

  * **globais** (ou externas): acessíveis em todo o programa. São definidas fora de qualquer função.

  * **locais** (ou automáticas): só são acessíveis no bloco em que estão definidas. O armazenamento da variável é alocado quando se inicia o bloco (p.e chamada duma função) e é libertado quando o bloco termina (p.e saída duma função).

```c
int g;  // g é uma variável global

int main() {
 int l;  // l é uma variável local à função main()

 g = 1; l = 2;

 {  int vl;  //
    vl = g + l; // g e l são também conhecidas...
  }
   /* aqui nao se tem acesso à variável vl */
}
```

Variáveis com o mesmo nome

* Duas variáveis **locais** em blocos distintos são distintas.

* Pode-se declarar uma variável local com o mesmo nome duma variável **externa**. Neste caso a variável local tem precedência sobre a externa; a externa fica escondida.

```
int count,total;  // count, total  são externas
int main(){
  count = 0;
  total = 1;
  {
     int count;          // count variável local
     count = 0;
     while (count < 10) {
       total += count++;
     }
  }
  printf("%d %d\n", count, total); /* qual o valor escrito? */
}
```

Classes das Variáveis

* A classe indica a **duração** duma variável
* A classe duma variável pode ser **permanente** ou **temporária**.
* As variáveis externas são sempre permanentes.
* São criadas e inicializadas antes do programa começar e não desaparecem até ele terminar.
* Em particular, os seus valores permanecem depois das funções que os modificam terminarem.

- As variáveis automáticas são temporárias:
  - são inicializadas sempre que se entra num bloco (p.e numa função) e desaparecem quando se sai do bloco (ou da função).
  - podem-se tornar permanentes utilizando na sua declaração o qualificador `static`.
  - Embora, raramente utilizado, o qualificador `auto` pode ser usado para assegurar que uma variável é automática.
  - Podemos ainda obrigar que sejam guardadas num registo programável do CPU. Para isso usa-se o qualificador `register` (cabe ao compilador ignorá-lo ou não!)

Funções e Variáveis

* Vimos que um programa em C é um conjunto de definições de funções e cada uma delas pode ter uma lista de parâmetros a que correspondem os argumentos de cada chamada.

```c
int complica(int);

int main() {
  int i;
  for(i = 0; i < 100; i++)
    if (complica(i)) printf("%d e' impar\n", i);
  return 0;
}

int complica(int i) {
  int j, n = 1;
  printf("%d \n",n++);
  j = i % 2;
  if (i == 10) i = 100;
  return j;
  }
```

O que escreve este programa?

Passagem de argumentos

* Os argumentos das funções são **passados por valor:**
  * a função que é chamada recebe o valor de cada argumento e guarda-o numa variável temporária.
* Os parâmetros da definição duma função são **variáveis locais,** inicializadas com os valores com que a função é chamada.
* Assim, se o valor dum parâmetro for alterado na função chamada, a variável usada como argumento não é alterada, na função que a chamou.

Funções e Variáveis Externas

* Quando se usa uma variável externa numa função deve-se declarar essa variável colocando o prefixo `extern` e a sua declaração normal, indicando que ela está definida fora dessa função.

* Contudo só é obrigatório, quando a essa função estiver definida **antes** da definição da variável ou quando as funções que constituem o programa estiverem definidas em vários ficheiros (e a função que a use estiver definida num ficheiro diferente do da sua definição).

```
int main() {
  extern int i;
  for(i=0; i <100; i++)
    if (complica2())
       printf("%d impar\n", i);
  return 0
}

int complica2(){
  extern int i;
  int j;
  static int n=1;
  printf("%d \n",n++);
  j = i % 2;
  if (i == 10) i = 100;
  return j;
}

int i;
```

O que escreve este programa? Classifica as variáveis

Variáveis Indexadas como Parâmetros de Funções

```
int max(int [],int); // prototipo da funcao

int main() {
 int a[10]={1,2,33,5,6,7,8}, n=7;
 int b[]={2,39,5}, m=3;
 printf("o máximo de a[] é %d e o de b[] é %d\n",
                    max(a,n),max(b,m));
 return 0;
}

int max(int v[],int n) {
    int imax = 0;
    while(--n > 0) if(v[n] > v[imax]) imax = n;
    return v[imax];
}
```

* Se uma variável indexada for passada como argumento, os seus elementos não são copiados, mas sim é passado o **endereço** do seu primeiro elemento. E, a função que foi chamada **pode alterar** os valores da variável.

* Diz-se que as variáveis indexadas são passadas por **referência**.

* O que escreve o programa anterior? Indica uma vantagem de se passar uma variável indexada por argumento em relação a definí-la externamente. indexada.

Declaração de Funções e Protótipos (II)

* Em C as funções são objectos **externos** uma vez que são definidas fora de qualquer outra função.
* Assim, tal como as variáveis externas, ou estão definidas antes das funções que as chamam ou têm de ser declaradas antes!
* Senão é pressumido que o tipo de resultado é `int` e o dos argumentos também!

- Na declaração tem de se indicar:
  - o **tipo** de resultado da função: se a função não retornar nenhum valor o seu tipo é `void` (da palavra inglesa, vazio) e também tem de ser indicado na definição da função.
  - o **nome** da função
  - a lista de parâmetros, indicando apenas o seu tipo ou também o seu nome (que não necessita de ser o mesmo dos dos parâmetros da definição da função).
  - Se a função não tiver parâmetros deve-se indicar `void`.

* Para a função `max` do exemplo anterior podiamos ter

```
int max(int [],int);
```

* ou

```
int max(int a[],int m);
```

* Para a função `complica2` deviamos ter:

```
int complica2(void);
```

* Estas declarações chamam-se os **protótipos** das respectivas funções.

Estrutura de um Programa em C

- Comandos para inclusão de ficheiros com declarações (`#include`)
- Definições de constantes (`#define`)
- Declarações de objectos externos (protótipos de funções e variáveis)
- Definições de funções (incluindo a função `main()`)

Um exemplo com quase tudo