

# **ESTRUTURA DE DADOS**

## **ORDENAÇÃO E PESQUISA**

# Olá!

Ao final desta aula, você será capaz de:

1. Compreender e usar o método de ordenação insertion sort, (inserção) em estruturas homogêneas e em estruturas heterogêneas;
2. Compreender e usar o método de ordenação selection sort (seleção) em estruturas homogêneas e em estruturas heterogêneas;
3. Compreender e usar o método de ordenação bubble sort (bolha) em estruturas homogêneas e em estruturas heterogêneas;
4. Compreender e usar os métodos de pesquisa sequencial em estruturas homogêneas e em estruturas heterogêneas;
5. Compreender e usar os métodos de pesquisa binária em estruturas homogêneas e em estruturas heterogêneas.

## 1 Ordenar e Pesquisar - uma parceria que não se desfaz mais

A ordenação e a pesquisa são dois tópicos fundamentais para a programação e ficou muito difícil escolher entre tantos métodos quais deveriam ser abordados sem ter que agregar novos conceitos a esta aula. No que diz respeito aos métodos de Ordenação, a decisão ficou mais complicada porque um dos métodos precisa do conceito de recursividade que não é objeto de estudo desta disciplina e que necessita de uma maturidade maior de programação. Sabemos que este estudo não se encerra em uma aula e sugiro que você continue sua pesquisa para conhecer os outros métodos que falaremos, de forma superficial sobre eles. Esta aula estará dividida em dois momentos: Ordenação e Pesquisa.

## 2 Ordenação

### Introdução

Normalmente, quando o usuário insere dados, ele não se preocupa se os mesmos estão, ou não, em ordem, pois ele acredita que o sistema fará isso para ele. Este é um quesito básico e o cliente não precisa mais deixar isto claro. Sendo assim, qualquer programador, precisa conhecer os métodos de ordenação. **Decorar?** Não! Não estou dizendo isso, mesmo porque ele poderá estar na sua biblioteca, mas alguns são tão simples que você acabará

sabendo. O que disse foi saber escolher o método certo para cada aplicação, mas, para que isso seja possível, você deverá compreender a lógica de cada método para que não tenha que decorar as vantagens de cada um e sim, saber citá-las baseado no conhecimento da lógica do algoritmo de cada método.

### **Recordando...**

Antes de conhecermos os métodos de ordenação, seria conveniente reforçar um pouco mais, já que vimos na aula passada, como os vetores numéricos (*int*, *float* ou *double*), vetor de char, matriz de char ou matriz de struct podem ter seus conteúdos comparados e trocados de lugar se não estiverem ordenados de forma crescente, decrescente ou alfabética, como veremos nos exemplos a seguir.

**Nos exemplos, foram usados vetores com dois elementos porque ainda não apresentamos os algoritmos de ordenação.**

### **Int, float ou double**

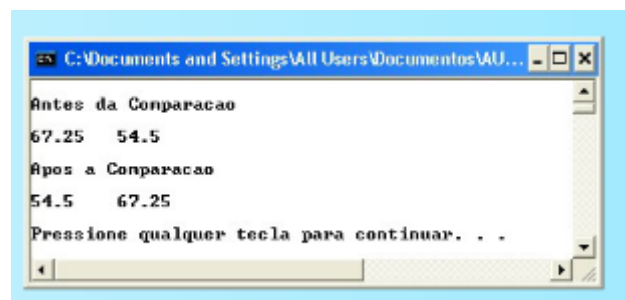
Você pode comparar os conteúdos de vetores numéricos usando os operadores relacionais >, >=, <, <=, == e !=.

Para trocar os conteúdos das variáveis, o comando de atribuição poderá ser usado.

```
if(vet[0]>vet[1])
{
    aux=vet[0];
    vet[0]=vet[1];
    vet[1]=aux;
}
```

### **Exemplo**

```
comparando-double.cpp
1 #include<iostream>
2 using namespace std;
3 int main()
4 {
5     double vet[]={ 67.250, 54.500},aux;
6     cout<<"\nAntes da Comparacao\n";
7     cout<<"\n"<<vet[0]<<"\t"<<vet[1];
8     if (vet[0]>vet[1])
9     {
10         aux=vet[0];
11         vet[0]=vet[1];
12         vet[1]=aux;
13     }
14     cout<<"\n\nApós a Comparacao\n";
15     cout<<"\n"<<vet[0]<<"\t"<<vet[1];
16     cout<<"\n\n";
17     system("pause");
18 }
```



## Char de um caracter

Essa é uma confusão muito comum no início, pois como fazer a diferença entre as declarações `char nome[30];` e `char sexo[30];` ?

Aparentemente, nenhuma. A diferença só acontecerá quando você construir o trecho de entrada de dados. Se usar um único comando, significa que todos os caracteres formam um conjunto unitário, mas se você possibilitar a entrada de 30 caracteres, como por exemplo o sexo de 30 pessoas, então esse é um conjunto de 30 elementos independentes.

Nesse exemplo, estamos nos referindo ao segundo caso e, sendo assim, você pode comparar os conteúdos usando os operadores relacionais `>`, `>=`, `<`, `<=`, `==` e `!=`.

Para trocar os conteúdos das variáveis, o comando de atribuição poderá ser usado.

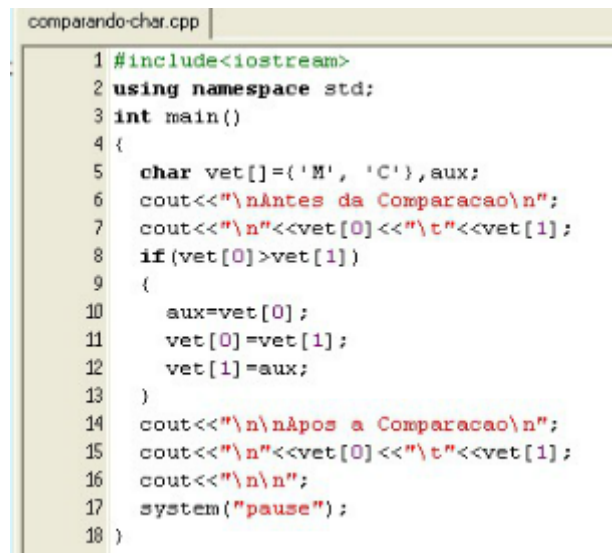
```
if(vet[0]>vet[1])
```

```

{
aux=vet[0];
vet[0]=vet[1];
vet[1]=aux;
}

```

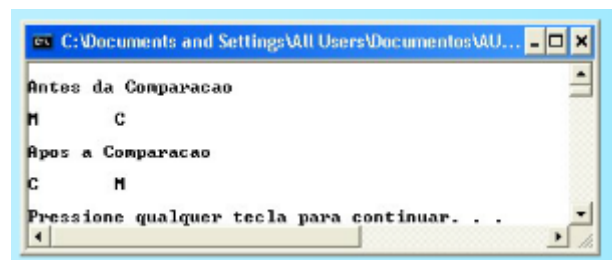
### Exemplo



```

comparando-char.cpp
1 #include<iostream>
2 using namespace std;
3 int main()
4 {
5     char vet[]={ 'M', 'C'},aux;
6     cout<<"\nAntes da Comparacao\n";
7     cout<<"\n"<<vet[0]<<"\t"<<vet[1];
8     if (vet[0]>vet[1])
9     {
10         aux=vet[0];
11         vet[0]=vet[1];
12         vet[1]=aux;
13     }
14     cout<<"\n\nApos a Comparacao\n";
15     cout<<"\n"<<vet[0]<<"\t"<<vet[1];
16     cout<<"\n\n";
17     system("pause");
18 }

```



```

C:\Documents and Settings\All Users\Documentos\WU...
Antes da Comparacao
M      C
Apos a Comparacao
C      M
Pressione qualquer tecla para continuar. . .

```

### Vetor de char

Nós agora vamos comparar vetores de char que juntos podem formar uma matriz bidimensional.

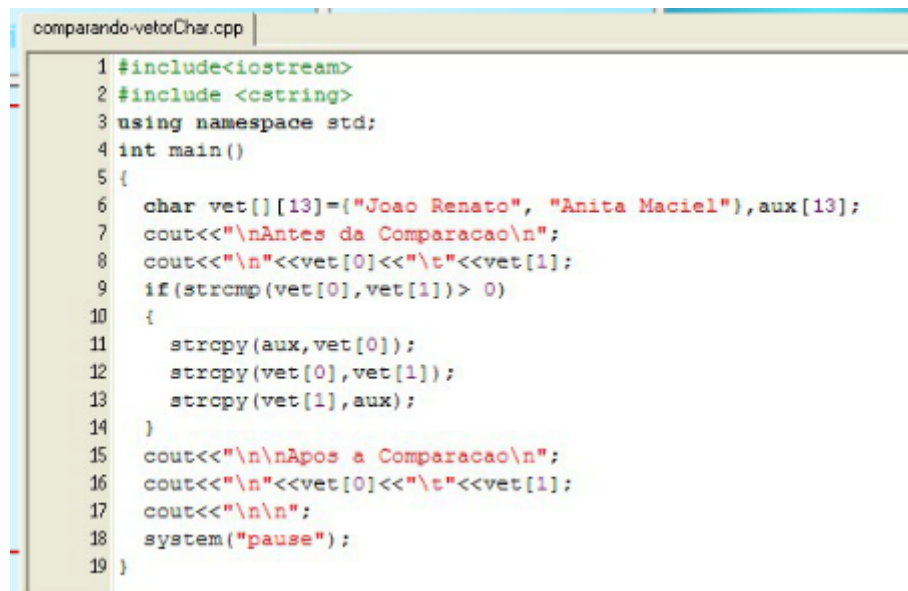
Você pode observar que cada caractere ocupa um endereço na MP e se pedirmos para exibir o nome de um vetor de char, sairá o primeiro endereço de todos os endereços que são ocupados pelo conjunto. Por essa razão, foi criado um conjunto de funções para manipular vetores de char e nesse nosso estudo, usaremos duas delas:

```
strcmp(
```

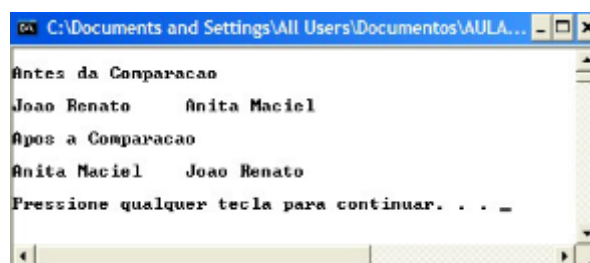
abreviatura de string compare) cuja função é comparar dois vetores de char e strcpy(abreviatura de string copy) cuja função é copiar o conteúdo de um vetor de char nas posições ocupadas por outro vetor de char. Essas funções foram estudadas na AULA10 da disciplina de Algoritmos.

```
if(strcmp(vet[0],vet[1]) > 0)
{
strcpy(aux,vet[0]);
strcpy(vet[0],vet[1]);
strcpy(vet[1],aux);
}
```

### Exemplo



```
comparando-vetorChar.cpp
1 #include<iostream>
2 #include <cstring>
3 using namespace std;
4 int main()
5 {
6     char vet[][13]={"Joao Renato", "Anita Maciel"},aux[13];
7     cout<<"\nAntes da Comparacao\n";
8     cout<<"\n"<<vet[0]<<"\t"<<vet[1];
9     if(strcmp(vet[0],vet[1])> 0)
10    {
11        strcpy(aux,vet[0]);
12        strcpy(vet[0],vet[1]);
13        strcpy(vet[1],aux);
14    }
15    cout<<"\n\nApos a Comparacao\n";
16    cout<<"\n"<<vet[0]<<"\t"<<vet[1];
17    cout<<"\n\n";
18    system("pause");
19 }
```



```
C:\Documents and Settings\All Users\Documentos\AULA...
Antes da Comparacao
Joao Renato    Anita Maciel
Apos a Comparacao
Anita Maciel    Joao Renato
Pressione qualquer tecla para continuar. . . _
```

### Membros de struct

Neste último exemplo, estaremos comparando membros de structs que fazem parte de um vetor de structs. As regras da comparação seguirão as mesmas dos exemplos anteriores, isto é, depende do tipo do membro, mas, como o tipo do escolhido é inteiro, poderemos usar os operadores relacionais >, >=, <, <=, == e != e, para trocar os conteúdos das variáveis, o comando de atribuição.

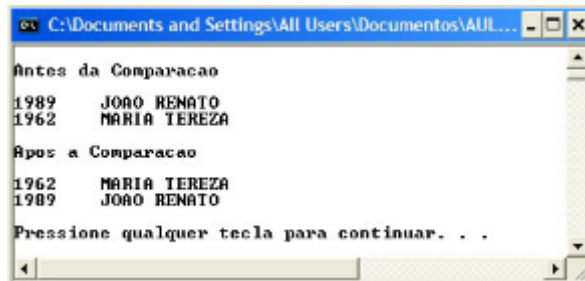
Entretanto, se tivéssemos escolhido o membro vetor de char, teríamos que proceder da mesma forma que no exemplo anterior.

```
if(candidatos[0].ninsc>candidatos[1].ninsc)
{
    aux=candidatos[0];
    candidatos[0]=candidatos[1];
    candidatos[1]=aux;
}
```

### Exemplo

```
1 #include<iostream>
2 using namespace std;
3 int main()
4 {
5     struct exemplo
6     {
7         int ninsc;;
8         char nome[31];
9     }aux,candidatos[]={1989, "JOAO RENATO",1962, "MARIA TEREZA"};
10    cout<<"\nAntes da Comparacao\n";
11    cout<<"\n"<<candidatos[0].ninsc<<"\t"<<candidatos[0].nome;
12    cout<<"\n"<<candidatos[1].ninsc<<"\t"<<candidatos[1].nome;
13    if(candidatos[0].ninsc>candidatos[1].ninsc)
14    {
15        aux=candidatos[0];
16        candidatos[0]=candidatos[1];
17        candidatos[1]=aux;
18    }
19    cout<<"\n\nApos a Comparacao\n";
20    cout<<"\n"<<candidatos[0].ninsc<<"\t"<<candidatos[0].nome;
21    cout<<"\n"<<candidatos[1].ninsc<<"\t"<<candidatos[1].nome;
22    cout<<"\n\n";
23    system("pause");
24 }
```

Cliqu



## 3 Os métodos de ordenação

Os métodos de ordenação constituem um bom exemplo de como resolver problemas utilizando computadores. As técnicas de ordenação permitem p. de algoritmos distintos para resolver uma mesma tarefa. Dependendo da aplicação, cada algoritmo considerado possui uma vantagem particular sobre os outros algoritmos. (ZIVIANI, N., 2002, p.69)

Ordenar significa dispor elementos de um conjunto seguindo um determinado critério. Esse critério estará baseado em um atributo do elemento do conjunto (nome, matrícula, salário, coeficiente de rendimento, etc). Sabemos que um conjunto quando se encontra ordenado, facilita a recuperação dos seus elementos. Você já imaginou se a lista telefônica não estivesse ordenada?

### Ordenação Interna versus Ordenação externa

Quando o método de ordenação só usa a Memória Principal para realizar o processo de ordenação, esse método é classificado como Ordenação Interna, mas se usa uma memória auxiliar porque o arquivo é muito grande, é classificado como Ordenação Externa. Os métodos de ordenação interna são classificados em dois tipos:

### 3.1 Métodos Simples

- indicados para conjuntos pequenos;
- usam muitas comparações;
- os códigos são pequenos;
- códigos de fácil entendimento.

**Exemplos:**

**1. Selection Sort**

**2. Insertion Sort**

**3. Bubble Sort**



### 3.2 Métodos Eficientes ou Sofisticados

- indicados para conjuntos grandes; ,
- usam menos comparações;
- os códigos são grandes;
- alguns incluem o conceito de recursividade, deixando-os muito complexos.

**Exemplos:**

**1. Shell Sort**

**2. Quick Sort**

**3. Heap Sort**

**4. Merge Sort**

Embora saibamos das vantagens dos métodos eficientes, fizemos a opção de apresentar os métodos simples, deixando para um outro momento, o estudo dos métodos eficientes por serem complexos e, como sempre digo, um passo de cada vez. Na disciplina de Algoritmos, falamos que não existe uma solução única para um problema. Evidentemente, as soluções apresentadas para os métodos de ordenação também não são únicas, mas tentei manter um padrão para ficar mais fácil o entendimento. Sendo assim, preferi usar em todos os métodos, a estrutura do *for*.



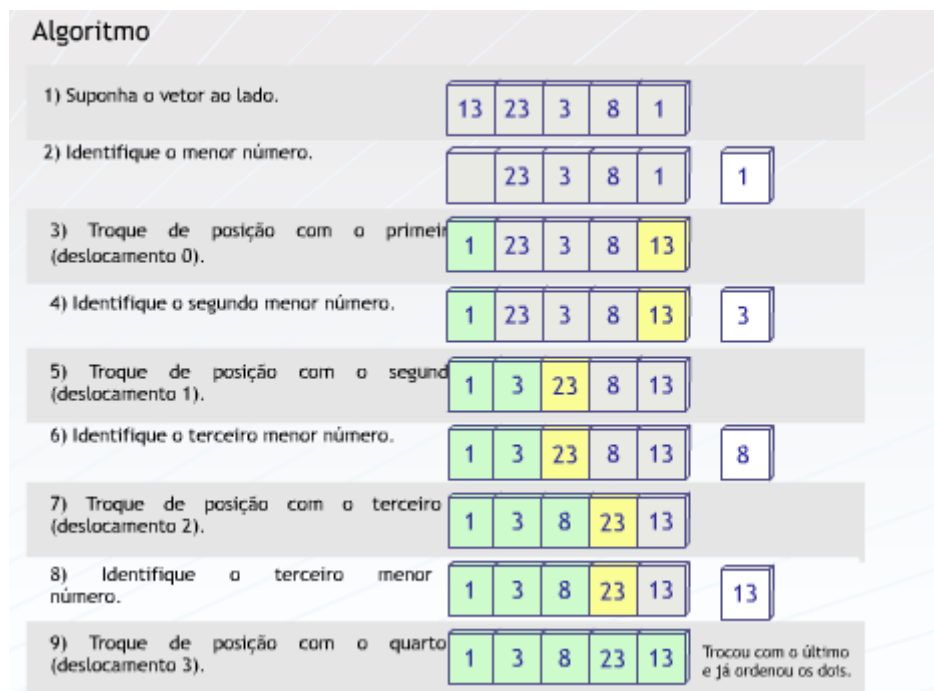
## 4 Selection Sort (Ordenação por seleção)

Começaremos pelo método cujo algoritmo é o mais simples de entendimento. Afirmamos isso porque seu princípio é sempre buscar o menor e ir posicionando-o no início como veremos a seguir.

Construiremos o passo a passo com figuras. Depois, apresentaremos todos os passos juntos. Codificaremos na linguagem C++, incluindo alguns trechos de impressão para ajudá-lo no entendimento do método e disponibilizaremos uma animação para tornar ainda mais clara a explicação. Esperamos que com todos esses passos, você consiga compreender a lógica deste método, mas se ficar alguma dúvida, fale com seu professor.

### Atenção

Relembrando: embora estejamos falando de primeiro, segundo, terceiro, quarto ou quinto, sabemos que cada elemento do vetor é indexado pelo deslocamento em relação ao endereço base, isto é, índice 0, 1, 2, 3 ou 4.



Você terá que construir um algoritmo que tenha uma repetição dentro de outra repetição, visto que você percorreu o vetor quatro vezes (2,4,6,8) para identificar o menor e, a cada vez que percorria o vetor, você comparava os elementos para encontrar o menor. Supondo que você usará duas estruturas de repetição (for), poderemos afirmar que cada vez que o for mais interno terminar, o algoritmo identificará o menor número e fará a troca.

A cada nova busca, o número de comparações diminuirá e esses procedimentos se repetirão até que o vetor esteja ordenado.

A figura abaixo reúne os passos importantes para lhe auxiliar na construção do algoritmo.

Seleciona o menor, após o termino do for mais interno.



Tente construir uma função de nome seleção que receba um vetor, o tamanho do vetor e ordene pelo método Selection Sort(Ordenação por Seleção). Depois, clique no botão e veja uma outra solução para essa função. Clique também no botão exemplo para acompanhar o passo a passo do programa.

```
void selecao(int vet[], int tam)
{
    int j,i, aux, temp;
    for (i=0; i<tam -1; i++)
    {
        aux = i;
        for(j=i+1; j<tam; j++)
            if(vet[aux] > vet[j] )
                aux=j;
        temp=vet[aux];
        vet[aux]= vet[i];
        vet[i]=temp;
    }
}
```

### Atenção

Observe que o for mais externo não se preocupa com a última posição do vetor porque quando compara a penúltima com a última, a última, automaticamente, será ordenada.

O for mais interno irá “varrendo” o vetor a partir da posição seguinte que estiver ordenada até a última posição do vetor fazendo comparações e armazenando a posição do menor valor.

A variável aux guarda a posição do menor elemento da comparação.

Quando o for mais interno finalizar, o valor armazenado na posição indicada em aux, será trocado de lugar como o elemento que se encontrar em posição trocada até que todo o vetor esteja ordenado.

Você poderá acompanhar a execução do algoritmo de duas maneiras: através da salda, uma vez que incluímos trechos para exibir o vetor em momentos diferentes do processo de ordenação ou através da figura.

## código fonte

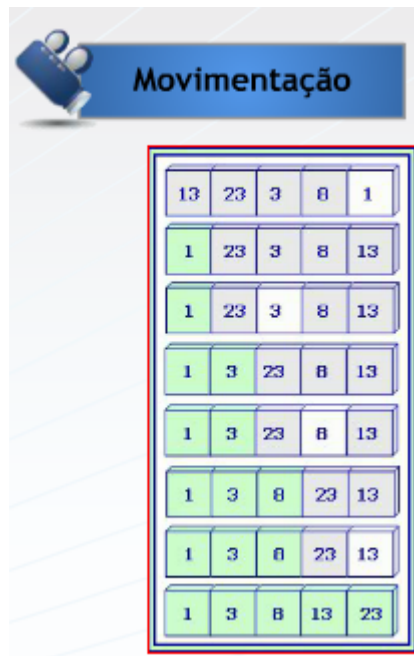
```
selecao.cpp
1 #include<iostream>
2 using namespace std;
3 void selecao(int vet[], int
4 int main()
5 {
6     int x,vet[]={13, 23, 3, 8
7     system("cls");
8     cout<<"\nAntes da chamada
9     for(x=0; x<5;x++)
10         cout <<vet[x]<<"\t";
11     cout<<"\n";
12     selecao(vet, 5);
13     cout<<"\n\nDepois da chama
14     for( x=0; x<5;x++)
```

## Saída

```
C:\Documents and Settings\Convidado\Meus documentos\ESTRUC...
Antes da chamada da Funcao - SELECAO
      13      23      3      8      1
i= 0   antes  13      23      3      8      1
i= 0   depois 1      23      3      0      13
i= 1   antes  1      23      3      8      13
i= 1   depois 1      3      23      8      13
i= 2   antes  1      3      23      0      13
i= 2   depois 1      3      8      23      13
i= 3   antes  1      3      0      23      13
i= 3   depois 1      3      8      13      23

Depois da chamada da Funcao - SELECAO
      1      3      8      13      23

Pressione qualquer tecla para continuar. . . .
```



## 5 Insertion Sort (Ordenação por inserção)

Suponha que todas as suas contas a pagar cheguem antes do dia 30 e que, nesse dia, você coloque em ordem de acordo com o dia de pagamento. Então, você pega a primeira conta com a mão direita e passa para a esquerda e como é a primeira, poderia afirmar que ela está ordenada. Quando você pega a segunda, com a mão direita, verifica se a data de vencimento é maior do que a primeira e se for, coloca atrás da conta que está na mão esquerda, mas se não for, coloca na frente. Este processo vai se repetindo até a última conta. Este é o princípio básico do método de inserção que considera o vetor como dois subvetores, um ordenado como as contas da sua mão esquerda e o outro, desordenado, como as contas que a sua mão direita vai apanhando e tentando localizar a posição correta no conjunto de contas da mão esquerda.

### Algoritmo

1) Considere o primeiro elemento (deslocamento 0) ordenado. Assim como nas contas, “na mão esquerda”.



2) Compare o segundo elemento (deslocamento 1) com o primeiro (deslocamento 0). Como ele é maior, ficará na segunda posição. Neste caso, não se altera.



3) É a vez do terceiro elemento (deslocamento 2). Como ele é menor do que o elemento da posição 2 (deslocamento 1) e menor do que o elemento da posição 1 (deslocamento 0), os dois elementos deverão ser deslocados para direita para que ele entre na primeira posição (deslocamento 0).



4) Agora chegou a vez do quarto elemento do vetor (deslocamento 3). Como ele é menor do que os elementos que se encontram nas posições 2 (deslocamento 1) e 3 (deslocamento 2), os dois elementos deverão ser deslocados para a direita para que o quarto elemento entre na posição 2 (deslocamento 1).



5) Finalmente, chegou a vez do elemento da posição 5 (deslocamento 4). Como ele é o menor de todos, se faz necessário que os outros desloquem para direita, uma vez que o último elemento assumirá a primeira posição. (deslocamento 0).



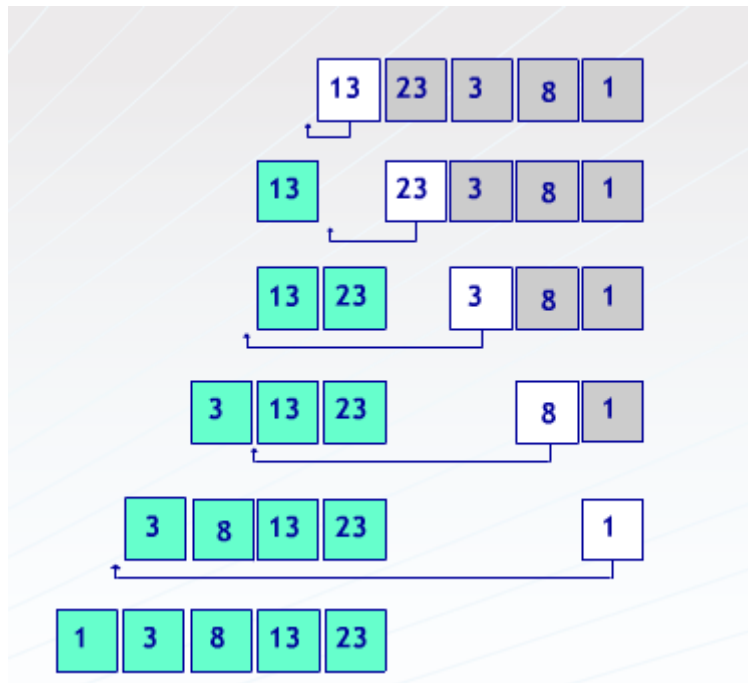
6) O vetor está ordenado.



## 6 A lógica do Método Insertion Sort

Você terá que construir um algoritmo que tenha uma repetição dentro de outra repetição. Entretanto, a repetição interna, se usada com a estrutura do for, deverá ter um teste que além de controlar a variável do for, interrompa a repetição se a variável tiver um conteúdo maior do que as que lhe antecedem em posição no vetor.

A figura abaixo reúne os passos importantes para lhe auxiliar na construção do algoritmo.



Tente construir uma função de nome inserção que receba um vetor, o tamanho do vetor e ordene pelo **método Insertion Sort(Ordenação por Inserção)**.

Veja a seguir uma outra solução para essa função. Veja também um exemplo para acompanhar o passo a passo do programa.

```
void insercao(int vet[], int tam)
```

```
{
```

```
int j,i, aux;
```

```
for (i=1;i<tam;i++)
```

```

{
aux = vet[i];
for(j=i; j>0 && aux <vet[j-1];
j--)
vet[j]=vet[j-1];
vet[j]=aux;
}
}

```

### **Atenção**

Observe que o for mais externo começa com a segunda posição do vetor porque o for de dentro sempre compara com os elementos anteriores. O teste de  $j > 0$  se justifica porque dentro dos colchetes, ele subtrai de 1 o valor de  $j$  e se deixasse  $j$  chegar a zero, o índice ficaria negativo o que não é permitido na linguagem C++.

O elemento fundamental do for interno é o teste:  $j > 0 \ \&\& \ \text{aux} < \text{vet}[j-1]$  porque incorpora um teste interrompendo a repetição quando a condição  $\text{aux} < \text{vet}[j-1]$  não for mais atendida. O for usado desta forma é uma herança da linguagem C e talvez mais fácil de ser construído do que a estrutura do while. Nada contra a estrutura do while, mas como já havia dito, tentamos manter uma uniformidade na apresentação desses métodos.

**Você poderá acompanhar a execução do algoritmo de duas maneiras: através da saída, uma vez que incluímos trechos para exibir o vetor em momentos diferentes do processo de ordenação ou através da figura.**



## código fonte

```
insercao.cpp
1 #include<iostream>
2 using namespace std;
3 void insercao(int vet[], int tam);
4 int main()
5 {
6     int x,vet[]={13, 23, 3, 8, 1};
7     system("cls");
8     cout<<"\nAntes da chamada da Funcao - INSERCAO\n\t\t";
9     for(x=0; x<5;x++)
10         cout <<vet[x]<<"\t";
11     cout<<"\n";
12     insercao(vet, 5);
13     cout<<"\n\nDepois da chamada da Funcao - INSERCAO\n\t\t";
14     for(x=0; x<5;x++)
15         cout <<vet[x]<<"\t";
16     cout<<"\n";
17     system("pause");
18 }
19 void insercao(int vet[], int tam)
20 {
21     int j, i, aux,a;
22     for (i=1;i<tam;i++)
23     {
24         aux = vet[i];
25         cout<<"\ni= "<<i<<"\ntantes\t";
26         for(a=0; a<5;a++)
27             cout <<vet[a]<<"\t";
28         for(j=i; j>0 && aux <vet[j-1]; j--)
29         {
30             vet[j]=vet[j-1];
31             cout<<"\n\tdurante\t";
32             for(a=0; a<5;a++)
33                 cout <<vet[a]<<"\t";
34         }
35         vet[j]=aux;
36         cout<<"\n\tdepois\t";
37         for(int a=0; a<5;a++)
38             cout <<vet[a]<<"\t";
39     }
40 }
```

Saída

```

C:\Documents and Settings\Convidado\Meus documentos\ESTRU...
Antes da chamada da Funcao - INSERCAO
      13      23      3      8      1
i= 1  antes  13      23      3      8      1
      depois 13      23      3      8      1
i= 2  antes  13      23      3      8      1
      durante 13      23      23      8      1
      durante 13      13      23      8      1
      depois 3      13      23      8      1
i= 3  antes  3      13      23      8      1
      durante 3      13      23      23      1
      durante 3      13      13      23      1
      depois 3      8      13      23      1
i= 4  antes  3      8      13      23      1
      durante 3      8      13      23      23
      durante 3      8      13      13      23
      durante 3      8      8      13      23
      durante 3      3      8      13      23
      depois 1      3      8      13      23

Depois da chamada da Funcao - INSERCAO
      1      3      8      13      23

Pressione qualquer tecla para continuar. . . .

```

Movimentação

13	23	3	8	1
13	23	3	8	1
13	23	3	8	1
13	23	23	8	1
13	13	23	8	1
3	13	23	8	1
3	13	23	8	1
3	13	23	23	1
3	13	13	23	1
3	8	13	23	1
3	8	13	23	1
3	8	13	23	23
3	8	13	13	23
3	8	8	13	23
3	3	8	13	23
1	3	8	13	23

## 7 Bubble Sorte (Ordenação por Bolha)

Por que bolha?

Faça a seguinte experiência. Encha uma bacia com água e, quando a água estiver totalmente parada, jogue um objeto pesado dentro da bacia. Observe que bolhas de ar irão subir. Justificativas como essa são dadas para explicar o nome deste método, visto que os valores adjacentes são comparados de forma contínua dando a impressão de um borbulhamento.

Para que você pudesse entender o funcionamento deste método, não foi possível fazer uma simplificação maior como fizemos nos outros métodos.

### **Algoritmo**

- 1) O quinto elemento (deslocamento 4) é comparado com o quarto elemento (deslocamento 3) e como é menor, eles trocam de posição.
- 2) O quarto elemento (deslocamento 3) é comparado com terceiro elemento (deslocamento 2) e como ele é menor, eles trocam de posição.
- 3) O terceiro elemento (deslocamento 2) é comparado com segundo elemento (deslocamento 1) e como ele é menor, eles trocam de posição.
- 4) O segundo elemento (deslocamento 1) é comparado com primeiro elemento (deslocamento 0) e como ele é menor, eles trocam de posição.
- 5) Após o menor número ter chegado à posição correta, o processo recomeça com o quinto elemento (deslocamento 4) que é comparado com o quarto elemento (deslocamento 3), mas como não é menor, nada acontece.
- 6) Sendo assim, o processo recomeça com o quarto elemento (deslocamento 3) que é comparado com terceiro elemento (deslocamento 2) e como ele é menor, eles trocam de posição.
- 7) O terceiro elemento (deslocamento 2) é comparado com segundo elemento (deslocamento 1) e como ele é menor, eles trocam de posição.
- 8) Após o segundo menor número ter chegado à posição correta, o processo recomeça com quinto elemento (deslocamento 4) que é comparado com o quarto elemento (deslocamento 3), e como ele é menor, eles trocam de posição.
- 9) O quarto elemento (deslocamento 3) é comparado com terceiro elemento (deslocamento 2) e como ele é menor, eles trocam de posição.
- 10) Após o terceiro menor número ter chegado à posição correta, o processo recomeça com o quinto elemento (deslocamento 4) sendo comparado com o quarto elemento (deslocamento 3), e como ele não é menor, nada acontece.
- 11) Como já estão foram comparados os dois últimos elementos, o processo de ordenação já está finalizado.

13	23	3	8	1
13	23	3	1	8
13	23	1	3	8
13	1	23	3	8
1	13	23	3	8
1	13	23	3	8
1	13	3	23	8
1	3	13	23	8
1	3	13	8	23
1	3	8	13	23
1	3	8	13	23

## 8 A lógica do Método Bubble Sort

Você terá que construir um algoritmo que tenha uma repetição dentro de outra repetição. Na repetição interna, toda vez que dois valores de posições adjacentes forem encontrados fora de ordem, deverão ser trocados de posições. No exemplo, o borbulhamento começou de baixo para cima, mas você poderá fazer de outra forma. A figura abaixo reúne os passos importantes para lhe auxiliar na construção dos algoritmos.

13	23	3	8	1
13	23	3	1	8
13	23	1	3	8
13	1	23	3	8
1	13	23	3	8
1	13	23	3	8
1	13	3	23	8
1	3	13	23	8
1	3	13	8	23
1	3	8	13	23
1	3	8	13	23

Tente construir uma função de nome bolha que receba um vetor, o tamanho do vetor e ordene pelo método Bubble Sort. Depois, veja a seguir uma outra solução para essa função. Veja também um exemplo para acompanhar o passo a passo do programa.

```
void bolha(int vet[], int tam)
{
    int j,i, aux;
    for (i=0; i<tam -1; i++)
        for(j=tam-1; j>i; j--)
            if(vet[j] < vet[j-1] )
            {
                aux=vet[j];
                vet[j]= vet[j-1];
                vet[j-1]=aux;
            }
}
```

### Atenção

Observe que o for mais externo começa na primeira posição do vetor e o for interno sempre começa na última posição, significando que o borbulhamento começará de baixo para cima.

O teste fará troca toda vez que encontrar valores adjacentes onde o valor maior se encontrar em posição menor do que o valor menor. Isto ocorre porque o método está ordenando de forma crescente.

Esse processo vai se repetindo até que todos elementos sejam colocados em suas posições corretas.

**Você poderá acompanhar a execução do programa de duas maneiras: através da saída, uma vez que incluímos trechos para exibir o vetor em momentos diferentes do processo de ordenação ou através da figura.**

**Código fonte**

```
1 #include<iostream>
2 using namespace std;
3 void bolha(int vet[], int tam):
4 int main()
5 {
6     int x ,vet[]={13, 23, 3, 8 , 1};
7     system("cls");
8     cout<<"\nAntes da chamada da Funcao - BOLHA\n\t\t";
9     for(x=0; x<5;x++)
10         cout <<vet[x]<<"\t";
11     cout<<"\n";
12     bolha(vet, 5);
13     cout<<"\n\nDepois da chamada da Funcao - BOLHA\n\t\t";
14     for(x=0; x<5;x++)
15         cout <<vet[x]<<"\t";
16     cout<<"\n\n";
17     system("pause");
18 }
19 void bolha(int vet[], int tam)
20 {
21     int j,i, aux,a;
22     for (i=0; i<tam -1; i++)
23     {
24         cout<<"\ni="<<i<<"\tcantes\t";
25         for(a=0; a<5;a++)
26             cout <<vet[a]<<"\t";
27         for(j=tam-1; j>i; j--)
28         {
29             if(vet[j] < vet[j-1] )
30             {
31                 aux=vet[j];
32                 vet[j]= vet[j-1];
33                 vet[j-1]=aux;
34             }
35             cout<<"\ni="<<i<<"\tdepois\t";
36             for(a=0; a<5;a++)
37                 cout <<vet[a]<<"\t";
38         }
39     }
40 }
```

**Saída**

```

C:\Documents and Settings\Convidado\Meus documentos\ESTR...
Antes da chamada da Funcao - BOLHA
      13      23      3      8      1
i=0  antes  13      23      3      8      1
      depois 13      23      3      1      8
i=0  depois 13      23      1      3      8
i=0  depois 13      1      23      3      8
i=0  depois 1      13      23      3      8
i=1  antes  1      13      23      3      8
i=1  depois 1      13      23      3      8
i=1  depois 1      13      3      23      8
i=1  depois 1      3      13      23      8
i=2  antes  1      3      13      23      8
i=2  depois 1      3      13      8      23
i=2  depois 1      3      8      13      23
i=3  antes  1      3      8      13      23
i=3  depois 1      3      8      13      23
Depois da chamada da Funcao - BOLHA
      1      3      8      13      23
Pressione qualquer tecla para continuar. . .

```

Movimentação

13	23	3	8	1
13	23	3	1	8
13	23	1	3	8
13	1	23	3	8
1	13	23	3	8
1	13	23	3	8
1	13	3	23	8
1	3	13	23	8
1	3	13	8	23
1	3	8	13	23
1	3	8	13	23

## 9 Afinal, qual seria o melhor método de ordenação?

Como disse no início, não teríamos como fazer um estudo incluindo muitos métodos de ordenação para que ao final, pudéssemos fazer uma análise consistente. Sendo assim, levantaremos somente alguns pontos sobre os três métodos estudados que são os mais simples, mas não são os indicados para grandes volumes de dados.

### **SELECTION SORT**

- Ideal para arquivos pequenos.
- Muito simples.
- Não melhora sua performance se o arquivo já estiver ordenado.
- Tem menos movimentos do que o Insertion Sort.

### **INSERTION SORT**

- Só ordena quando necessário.
- Quando um elemento é inserido, todos os elementos maiores do que ele, são deslocados para a direita.
- É considerado o melhor dos três métodos estudados.

### **BUBBLE SORT**

- É o mais conhecido método.
- É muito simples.
- Muito lento.
- É considerado o pior dos três estudados.

Vamos agora empregar os três métodos estudados numa mesma aplicação para que possamos constatar diferenças visíveis na construção do código, implicando num tempo maior para ordenar o conjunto de dados uma vez que o número de trocas é muito maior. Observe com muita atenção esse aspecto. O programa consiste em armazenar códigos e alturas de 5 atletas. Ordenar pelo código e exibir códigos e alturas.

### **Seleção**



```

selecaoAplicacao.cpp
1 #include<iostream>
2 using namespace std;
3 void selecao(int vet[], float h[],int tam);
4 int main()
5 {
6     int x,codigo[5];
7     float alt[5];
8     for(x=0; x<5;x++)
9     {
10         cout << "\nCodigo do atleta: ";
11         cin>>codigo[x];
12         cout<<"\nAltura: ";
13         cin>>alt[x];
14     }
15     selecao(codigo, alt, 5);
16     cout<<"\nCodigo\tAltura\n";
17     for(x=0; x<5;x++)
18         cout<<"\n" <<codigo[x]<<"\t"<<alt[x];
19     cout<<"\n";
20     system("pause");
21 }
22 void selecao(int vet[], float h[],int tam)
23 {
24     int j,i, aux, temp;float temp1;
25     for (i=0; i<tam-1; i++)
26     {
27         aux = i;
28         for(j=i+1; j<tam; j++)
29             if(vet[aux] > vet[j])
30                 aux=j;
31         temp=vet[aux];
32         vet[aux]= vet[i];
33         vet[i]=temp;
34
35         temp1=h[aux];
36         h[aux]= h[i];
37         h[i]=temp1;
38     }
39 }

```

```

C:\Documents and Settings\Anita Lopes\
Codigo do atleta: 23
Altura: 1.81
Codigo do atleta: 13
Altura: 1.67
Codigo do atleta: 6
Altura: 1.49
Codigo do atleta: 17
Altura: 1.56
Codigo do atleta: 4
Altura: 1.66
Codigo  Altura
4      1.66
6      1.49
13     1.67
17     1.56
23     1.81
Press any key to continue . . .

```

## Inserção

```

insercaoAplicacao.cpp
1 #include<iostream>
2 using namespace std;
3 void insercao(int vet[], float h[],int tam);
4 int main()
5 {
6     int x,codigo[5];
7     float alt[5];
8     for(x=0; x<5;x++)
9     {
10         cout << "\nCodigo do atleta: ";
11         cin>>codigo[x];
12         cout<<"\nAltura: ";
13         cin>>alt[x];
14     }
15     insercao(codigo, alt, 5);
16     cout<<"\nCodigo\tAltura\n";
17     for(x=0; x<5;x++)
18         cout<<"\n" <<codigo[x]<<"\t"<<alt[x];
19     cout<<"\n";
20     system("pause");
21 }
22 void insercao(int vet[], float h[],int tam)
23 {
24     int j,i, aux; float auxf;
25     for (i=1;i<tam;i++)
26     {
27         aux = vet[i];auxf=h[i];
28         for(j=i; j>0 && aux <vet[j-1]; j--)
29             { vet[j]=vet[j-1]; h[j]=h[j-1];}
30         vet[j]=aux; h[j]=auxf;
31     }
32 }

```

```

C:\Documents and Settings\Anita Lopes\
Codigo do atleta: 23
Altura: 1.81
Codigo do atleta: 13
Altura: 1.67
Codigo do atleta: 6
Altura: 1.49
Codigo do atleta: 17
Altura: 1.56
Codigo do atleta: 4
Altura: 1.66
Codigo  Altura
4      1.66
6      1.49
13     1.67
17     1.56
23     1.81
Press any key to continue . . .

```

## Bolha

```
1 #include<iostream>
2 using namespace std;
3 void bolha(int vet[], float h[], int tam);
4 int main()
5 {
6     int x, codigo[5];
7     float alt[5];
8     for(x=0; x<5; x++)
9     {
10         cout << "\nCodigo do atleta: ";
11         cin >> codigo[x];
12         cout << "\nAltura: ";
13         cin >> alt[x];
14     }
15     bolha(codigo, alt, 5);
16     cout << "\nCodigo\tAltura\n";
17     for(x=0; x<5; x++)
18         cout << "\n" << codigo[x] << "\t" << alt[x];
19     cout << "\n";
20     system("pause");
21 }
22 void bolha(int vet[], float h[], int tam)
23 {
24     int j, i, aux; float auxf;
25     for (i=0; i<tam-1; i++)
26         for (j=tam-1; j>i; j--)
27             if (vet[j] < vet[j-1])
28             {
29                 aux=vet[j];
30                 vet[j]=vet[j-1];
31                 vet[j-1]=aux;
32
33                 auxf=h[j];
34                 h[j]=h[j-1];
35                 h[j-1]=auxf;
36             }
37 }
38
```

C:\Documents and Settings\Anita Lopes...  
Codigo do atleta: 23  
Altura: 1.81  
Codigo do atleta: 13  
Altura: 1.67  
Codigo do atleta: 6  
Altura: 1.49  
Codigo do atleta: 17  
Altura: 1.56  
Codigo do atleta: 4  
Altura: 1.66  
Codigo    Altura  
4        1.66  
6        1.49  
13       1.67  
17       1.56  
23       1.81  
Press any key to continue . . .

## 10 Método de ordenação por seleção cujo critério é nome, usando struct

Neste exemplo, resolvemos usar struct para que você perceba a grande vantagem, mesmo usando o Método de Ordenação por Seleção, ao trocarmos os elementos de lugar, pois se estivéssemos usando vetores, teríamos que usar tantos trechos de trocas quanto fossem os vetores e, no entanto, por termos usado struct, só usamos um trecho de troca.

```

1 #include<iostream>
2 using namespace std;
3 struct dados
4 {
5     char nome[31]; int codigo; float alt;
6 };
7 void maiuscula(char n[]);
8 void selecao(dados vet[],int tam);
9 int main()
10 {
11     int x; dados atleta[5];
12     for(x=0; x<5;x++)
13     {
14         cout << "\nNome do atleta: "; cin.getline(atleta[x].nome,31);
15         maiuscula(atleta[x].nome);
16         while(strlen(atleta[x].nome)<30)
17             strcat(atleta[x].nome, " ");
18         cout << "\nCodigo do atleta: ";cin>>atleta[x].codigo;
19         cout<<"\nAltura: "; cin>>atleta[x].alt; cin.get();
20     }
21     selecao(atleta, 5);
22     cout<<"\nCodigo\tNome\t\t\t\tAltura\n";
23     for(x=0; x<5;x++)
24         cout<<"\n" <<atleta[x].codigo<<"\t"<<atleta[x].nome<<"\t"<<atleta[x].alt;
25     cout<<"\n"; system("pause");
26 }
27 void maiuscula(char n[])
28 { int x;
29   for(x=0; x<strlen(n); x++) n[x]=toupper(n[x]);
30 }
31 void selecao(dados vet[],int tam)
32 {
33     int j,i, aux; dados temp;
34     for (i=0; i<tam-1; i++)
35     {
36         aux = i;
37         for(j=i+1; j<tam; j++)
38             if(strcmp(vet[aux].nome,vet[j].nome)>0 )
39                 aux=j;
40         temp=vet[aux];
41         vet[aux]= vet[i];
42         vet[i]=temp;
43     }
44 }

```

```
C:\Documents and Settings\Anita Lopes\Desktop\FAD-FDA...
Nome do atleta: Joao renato
Codigo do atleta: 23
Altura: 1.81
Nome do atleta: anita LUIZA
Codigo do atleta: 13
Altura: 1.67
Nome do atleta: ana maciel
Codigo do atleta: 4
Altura: 1.66
Nome do atleta: maria TeReza
Codigo do atleta: 6
Altura: 1.49
Nome do atleta: maria CORREA
Codigo do atleta: 17
Altura: 1.56
Codigo   None           Altura
4        ANA MACIEL     1.66
13       ANITA LUIZA    1.67
23       JOAO RENATO    1.81
17       MARIA CORREA   1.56
6        MARIA TEREZA   1.49
Press any key to continue . . .
```

## 11 Pesquisa

### Introdução

Hoje em dia, quando se fala em pesquisar, logo alguém responde com um nome de um site de busca. Às vezes, falo para meus alunos da área tecnológica que nem todas as profissões têm essa "mania" e cito como exemplo o estudante/profissional da área de Direito que pode fazer provas consultando livros desde que não tenham marcação. Você já se imaginou consultando um dicionário do Aurélio? **Claro que não!** Muitos de vocês dirão. Entretanto, se você for fazer uma prova de língua estrangeira, em muitos lugares, poderá consultar o dicionário, mas não seu notebook. Enfim, o que pretendemos passar é que não paramos para pensar como esse processo acontece, pois já está incorporado em nossas vidas, mas, nós que somos dessa área, deveríamos nos preocupar porque o processo de pesquisa é fundamental na programação.

## 12 Pesquisa sequencial

Quando uma matriz não está ordenada, a única saída para pesquisar um elemento é fazendo uma "varredura" até encontrar o elemento, ou até o final para concluir que não encontrou. A pesquisa sequencial é simples, mas ineficiente, pois fazendo uma analogia com algo do nosso dia a dia, perceberemos a grandeza do problema.

Vamos imaginar que você chegou à cidade de São Paulo e quer procurar o endereço da sua amiga num catálogo telefônico, fora de ordem. Isso é um problema sem solução, diria você, com certeza, rindo. Não que seja sem solução, visto que você poderá achar logo de primeira ou demorar dias, dias, dias ... Sei que estamos exagerando, mas queremos deixar claro que esse tipo de pesquisa só será "eficiente" para um número pequeno de dados, mas temos que começar pelo mais fácil, não é verdade?



### ***12. 1 Primeiro trecho da Pesquisa sequencial***

Esse trecho fará uma "varredura" em toda a matriz e, mesmo depois que achar, continuará até o final.

```
cout<<"\n...? ";
cin>>varProcura;
achou=0;
for(L=0;L<tamLinha;L++)
{
    if(varProcura==vetor[L])
    {
        achou=1; posicao=L;
    }
}
if(achou==1)
```

```

cout<<"\n...: "<<outroVetor[posicao]<<endl;

else

cout<<"\nDado nao achado\n";

```

## ***12.2 Segundo trecho da Pesquisa sequencial***

Para melhorar o trecho anterior poderemos incluir um teste na estrutura do for já que a linguagem C++ apresenta uma estrutura do for com possibilidades de testes permitidos na estrutura do while/do Com a inserção desse teste, assim que o elemento for achado, o for será interrompido.

```

cout<<"\n...? ";

cin>>varProcura;

achou=0;

for(L=0;L<tamLinha && achou==0;L++)

{

if(varProcura==vetor[L])

{

achou=1; posicao=L;

}

}

if(achou==1)

cout<<"\n...: "<<outroVetor[posicao]<<endl;

else

cout<<"\nDado nao achado\n";

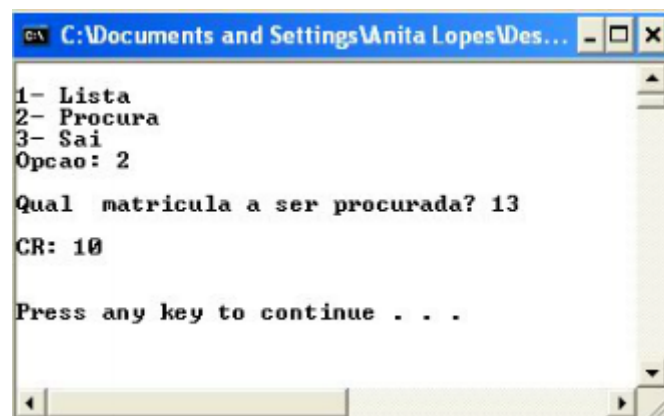
```

Como os dois trechos são bem parecidos, vamos apresentar somente uma aplicação.

```

1 #include <iostream>
2 #define tam 5
3 using namespace std;
4 int main()
5 {
6     int L,op, matProcura, pos, achou, matric[]={2,4,5,6,13};
7     float CR[]={8,9,7.8,6,10};
8     do
9     {
10         system("cls");
11         cout<<"\n1- Lista";
12         cout<<"\n2- Procura";
13         cout<<"\n3- Sai";
14         cout<<"\nOpcao: ";
15         cin>>op;
16         switch(op)
17         {
18             case 1: cout<<"\nMatriculas\n\n";
19                     for (L = 0 ; L < tam; L++)
20                         cout<<"\n"<<L+1<<" - "<<matric[L];
21                     break;
22             case 2: /*trecho de procura */
23                     cout<<"\nQual matricula a ser procurada? ";
24                     cin>>matProcura;
25                     achou=0;
26                     for(L=0;L<tam && achou==0;L++)
27                     {
28                         if(matProcura==matric[L])
29                         {
30                             achou=1; pos=L;
31                         }
32                     }
33                     if(achou==1)
34                         cout<<"\nCR: "<<CR[pos]<<endl;
35                     else
36                         cout<<"\nMatricula nao achada\n";
37                     break;
38             case 3: cout<<"\nFim\n";break;
39             default: cout<<"\nOpcao Invalida\n";
40         }
41         cout<<"\n\n": system("pause");
42     }while(op!=3);
43 }

```



```

C:\Documents and Settings\Anita Lopes\Des...
1- Lista
2- Procura
3- Sai
Opcao: 2

Qual matricula a ser procurada? 13

CR: 10

Press any key to continue . . .

```

## 13 Pesquisa binária

Todos os algoritmos que vimos no item anterior apresentam a vantagem de serem simples entretanto, a grande desvantagem é que podem procurar na matriz toda e não achar o elemento. A pesquisa binária chega para suprir essa deficiência, mas faz uma exigência: a matriz tem que estar ordenada. Utiliza a ideia de "Dividir para Conquistar", visando reduzir o espaço a ser pesquisado.

1) Suponha o vetor de 10 elementos.

2) Suponha que 18 é o número a ser procurado.

3) Vamos calcular o meio do vetor, sabendo-se que o início é 0 e o final é 9:  $(0+9)/2$  (divisão inteira). Logo, 4.

4) Como o número procurado é maior do que o meio, o início passa ser o seguinte do meio.

5) Calcula-se o novo meio:  $(5+9)/2=7$

6) Como o número procurado é menor do que o número que está na posição do meio, o meio passa a ser o fim.

Calcula-se o novo meio:  $(5+7)/2=6$  e, nesse caso, o meio coincidiu com o número procurado.



O trecho da pesquisa binária ficará assim:

```
inicio=0;
```

```
fim= tamanho - 1;
```

```
meio=(inicio+fim)/2;
```

```
while(procura != nomeVetor[meio] && inicio != fim)
```

```
{
```

```
if(procura > nomeVetor[meio])
```



```
inicio=meio+1;

else

fim=meio;

meio=(inicio+fim)/2;

}

if(nomeVetor[meio]==procura)

cout<<"\n.....: "<<outroVetor[meio]<<endl;

else

cout<<"\nDado nao encontrado\n";
```

Para finalizar, vamos aplicar o que acabamos de aprender. Entretanto, foi incluída a linha abaixo em dois lugares para dar mais um reforço ao aprendizado.

```
cout<<"\n"<<inicio<<"\t"<<meio<<"\t"<<fim; //PODE RETIRAR
```

```

1 #include <iostream>
2 #define tam 10
3 using namespace std;
4 int main()
5 {
6     int op,L,inicio,fim,meio,procura, matric[]={2,4,5,6,13,15,18,23,29,30};
7     float CR[]={8,9,7.8,6,10,9,9,9.7,9,9};
8     /* exibe e procura */
9     do
10     {
11         system("cls");
12         cout<<"\nMatriculas\n\n";
13         for (L = 0 ; L < tam ; L++)
14             cout<<"\n"<<L+1<<" - "<<matric[L];
15
16         cout<<"\n\n1- Procura";
17         cout<<"\n2- Sai";
18         cout<<"\nOpcao: ";
19         cin>>op;
20         if(op==1)
21         {
22             /* PESQUISA BINARIA */
23             cout<<"\nDigite matricula a ser procurada: ";
24             cin>>procura;
25
26             inicio=0;
27             fim=tam-1;
28             meio=(inicio+fim)/2;
29             cout<<"\n"<<inicio<<"\t"<<meio<<"\t"<<fim; //PODE RETIRAR
30             while(procura != matric[meio] && inicio != fim)
31             {
32                 if(procura > matric[meio])
33                     inicio=meio+1;
34                 else
35                     fim=meio;
36                 meio=(inicio+fim)/2;
37                 cout<<"\n"<<inicio<<"\t"<<meio<<"\t"<<fim; //PODE RETIRAR
38             }
39             if(matric[meio]==procura)
40                 cout<<"\nCR : "<<CR[meio]<<endl;
41             else
42                 cout<<"\nMatricula nao encontrada\n";
43         }
44         else if(op==2) cout<<"\nFim\n";
45         else cout<<"\nOpcao Invalida\n";
46         cout<<"\n\n"; system("pause");
47     }while(op!=2);
48 }

```

```

C:\Documents and Settings\Anita Lopes\Des...
1- Procura
2- Sai
3- Sai
Opcao: 2

Qual matricula a ser procurada? 13

CR: 10

Press any key to continue . . .

```

Nesta aula, estudamos alguns métodos de ordenação e de pesquisa que serão de muita utilidade na sua vida profissional.

Esperamos que você incorpore todo esse conteúdo e busque o aprendizado de outros métodos de ordenação, principalmente.

Procuramos, através de muitas figuras, tornar a aula mais amigável possível, mas se você não entendeu algum método, não se acanhe, fale com seu professor.

Não deixe de fazer os exercícios da Lista\_3 para reforçar o aprendizado. Para isso acesse a biblioteca de sua disciplina!

Até a próxima aula!

## O que vem na próxima aula

Na próxima aula, você estudará os seguintes assuntos:

- Compreender o conceito de lista linear sequencial;
- Implementar operações básicas em lista linear sequencial;
- Implementar métodos de ordenação e pesquisa em lista linear sequencial.

## CONCLUSÃO

Nesta aula, você:

- Compreendeu a importância da ordenação para a pesquisa;
- Aprendeu a diferença entre os métodos de ordenação;
- Aprendeu a diferença entre a pesquisa sequencial e a pesquisa binária.