

Notas de Aula de Algoritmos e Programação de Computadores

FLÁVIO KEIDI MIYAZAWA

com a colaboração de

TOMASZ KOWALTOWSKI

Instituto de Computação - UNICAMP

Versão 2000.1

Estas notas de aula não devem ser usadas como única fonte de estudo. O aluno deve ler outros livros disponíveis na literatura.

Nenhuma parte destas notas pode ser reproduzida, qualquer que seja a forma ou o meio, sem a permissão dos autores.

Os autores concedem a permissão explícita para a utilização e reprodução deste material no contexto do ensino de disciplinas regulares dos cursos de graduação sob a responsabilidade do Instituto de Computação da UNICAMP.

© Copyright 2000

Instituto de Computação
UNICAMP
Caixa Postal 6176
13083-970 Campinas-SP
{fkm,tomasz}@ic.unicamp.br

6 Variáveis Compostas Homogêneas

6.1 Vetores Unidimensionais

Até agora, vimos que uma variável está associada a uma posição de memória e qualquer referência a ela significa um acesso ao conteúdo de um pedaço de memória cujo tamanho depende de seu tipo. Nesta seção, iremos ver um dos tipos mais simples de estrutura de dados que nos possibilitará associar um identificador a um conjunto de elementos de um mesmo tipo. Naturalmente precisaremos de uma sintaxe apropriada para acessar cada elemento deste conjunto de forma precisa.

Antes de apresentar este tipo de estrutura de dados, considere o seguinte exemplo:

Exemplo 6.1 *Lêr 10 notas de alunos, calcular a média destas notas e imprimir as notas acima da média.*

Note que neste exemplo as notas devem ser lidas primeiro para depois se calcular a média das notas. Logo em seguida, cada nota deve ser comparada com a média, sendo que as maiores que a média são impressas.

Portanto, um programa para isso deveria conter pelo menos 10 variáveis apenas para guardar cada nota. O programa relativo a este exemplo se encontra na figura 18

```
program notas;
var nota1,nota2,nota3,nota4,nota5,
    nota6,nota7,nota8,nota9,nota10 : real;
    media                : real;

begin
write('Entre com a nota 1: '); readln(nota1);
write('Entre com a nota 2: '); readln(nota2);
write('Entre com a nota 3: '); readln(nota3);
write('Entre com a nota 4: '); readln(nota4);
write('Entre com a nota 5: '); readln(nota5);
write('Entre com a nota 6: '); readln(nota6);
write('Entre com a nota 7: '); readln(nota7);
write('Entre com a nota 8: '); readln(nota8);
write('Entre com a nota 9: '); readln(nota9);
write('Entre com a nota 10: '); readln(nota10);
media := (nota1+nota2+nota3+nota4+nota5+nota6+nota7+nota8+nota9+nota10)/10;
writeln('A média das notas é ',media);
if (nota1 > media) then writeln(nota1);
if (nota2 > media) then writeln(nota2);
if (nota3 > media) then writeln(nota3);
if (nota4 > media) then writeln(nota4);
if (nota5 > media) then writeln(nota5);
if (nota6 > media) then writeln(nota6);
if (nota7 > media) then writeln(nota7);
if (nota8 > media) then writeln(nota8);
if (nota9 > media) then writeln(nota9);
if (nota10 > media) then writeln(nota10);
end.
```

Figura 18: Leitura de 10 notas, cálculo da média e impressão das maiores notas, usando variáveis simples.

Note que o programa da figura 18 está cheio de duplicações e cálculos idênticos para cada nota. Agora imagine este programa feito para uma turma de 100 alunos. Certamente não seria nada agradável ler, escrever ou programar desta maneira.

Variáveis compostas homogêneas correspondem a posições de memória, identificadas por um mesmo nome, individualizadas por índices e cujo conteúdo é de mesmo tipo.

Assim, o conjunto de 10 notas pode ser associado a apenas um identificador, digamos NOTA, que passará a identificar não apenas uma única posição de memória, mas 10. A referência ao conteúdo do n -ésimo elemento do conjunto será indicada pela notação $NOTA[n]$, onde n é um valor inteiro (também podendo ser uma expressão cujo resultado é inteiro).

85	100	50	40	70	80	95	65	75	90
1	2	3	4	5	6	7	8	9	10

Figura 19: Vetor de 10 notas representado pelo identificador NOTA.

Na figura 19, a nota de valor 70, que está na quinta posição da sequência de notas é obtida como $NOTA[5]$.

A declaração de um vetor é feita usando a seguinte sintaxe:

Identificador : **array**[*Faixa_Escalar*] **of** Tipo_de_Cada_Elemento;

Onde *Faixa_Escalar* indica uma faixa de valores escalares. Naturalmente, se desejamos trabalhar com K elementos, então o vetor deve ser declarado com pelo menos K posições na faixa. Os elementos do vetor devem ser referenciados com valores de índice que pertencem a faixa de valores escalares usada na declaração do vetor.

Exemplo 6.2 *Exemplo de declaração de vetores:*

V : **array**[1..1000] **of** integer;
VetReais : **array**[-10..10] **of** real;
VetIndLetras : **array**['a'..'z'] **of** real;

A declaração do vetor NOTA, no exemplo acima, é feita da seguinte forma.

$NOTA$: **array**[1..10] **of** real;

Na figura 20, apresentamos o programa alternativo ao da figura 19, agora usando vetor.

```

program notas2;

const quant=10;

var nota    : array[1..10] of real;
    soma, media : real;
    i          : integer;

begin
    soma:=0;
    for i:=1 to quant do begin
        write('Entre com a nota ',i,' : '); readln(nota[i]);
        soma := soma+nota[i];
    end;
    media := soma/quant;
    writeln('A média das notas é ',media);
    for i:=1 to quant do
        if (nota[i] > media) then writeln(nota[i]);
end.

```

Figura 20: Leitura de 10 notas, cálculo da média e impressão das maiores notas. Versão usando vetores.

Note que agora é fácil fazer uma mudança no programa da figura 20 para que o programa faça o mesmo processamento para 100 alunos em vez de 10. Basta mudar a quantidade de alunos definida na constante *quant* de 10 para

100. Outra alternativa é declarar o vetor com quantidade suficientemente grande de elementos, usar uma variável para armazenar a quantidade de alunos e ler pelo teclado esta quantidade.

Exemplo 6.3 *O seguinte programa lê um número positivo n e n valores reais em um vetor. O programa imprime os elementos do vetor que estão acima da média. Vamos projetar nosso programa por etapas, usando a seqüência dos seguintes passos:*

1. Leia n , $n > 0$.
2. Leia os n valores reais no vetor.
3. Calcule a média M .
4. Imprima os valores maiores que M .

```
program LeImprimeVetorReal;
const MAX      = 100;
type TipoVetorReal = array[1..MAX] of real;
var n,i        : integer;
    v          : TipoVetorReal;
    M,Soma     : real;
begin
  { Passo 1: Leitura do valor n }
  write('Entre com a quantidade de elementos a ler: ');
  readln(n);
  while (n<1) or (n>MAX) do begin
    writeln('Quantidade fora dos limites possíveis [1, ',MAX,'].');
    readln(n);
  end;
  { Passo 2: Leitura dos n valores reais no vetor }
  for i:=1 to n do begin
    write('Entre com o ',i,'-esimo elemento: ');
    readln(v[i]);
  end;
  { Passo 3: Cálculo da Média dos n valores lidos }
  Soma := 0;
  for i:=1 to n do Soma := Soma + v[i];
  M := Soma/n;
  { Passo 4: Impressão dos valores maiores que M }
  writeln('Valores maiores que a média ',M,':');
  for i:=1 to n do
    if (v[i]>M) then writeln(v[i]);
end.
```

Exemplo 6.4 *Faça um programa para imprimir uma seqüência de números lidos em ordem inversa.*

```
program Inverte;
const TamMax = 100;
var i,n : Integer;
    v    : array [1..TamMax] of Integer;
begin
  Readln(n); { 0 <= n <= TamMax }
  for i:=1 to n do Read(v[i]);
  for i:=n downto 1 do Writeln(v[i]);
end.
```

Exemplo 6.5 O seguinte programa imprime o índice do maior elemento de um vetor lido. Neste exemplo, guardamos o índice onde aparece o maior elemento, inicialmente atribuído com 1. Em seguida, percorremos os demais elementos, atualizando o valor deste índice sempre que encontramos um elemento maior.

```

program maximo;
const MAX      = 100;
type TipoVetorReal = array[1..MAX] of real;
var i, n, Ind : integer;
    V      : TipoVetorReal;
begin
  { Leitura dos elementos do vetor }
  write('Entre com a quantidade de elementos a ler: '); readln(n);
  for i:=1 to n do begin
    write('Entre com o ',i,'-ésimo elemento: '); readln(V[i]);
  end;
  { Busca da posição do maior elemento no vetor, guardando na variável Ind }
  if (n>0) then begin
    Ind := 1;
    for i:=2 to n do
      if V[Ind] < V[i] then Ind:=i; {Ao encontrar um valor maior, atualiza Ind}
    end;
  { Impressão do maior valor e seu índice. }
  if (n<=0) then writeln('Não há elementos no vetor.')
  else writeln('O maior valor do vetor é ',V[Ind]:7:2,' e aparece no índice ',ind);
end.

```

Exemplo 6.6 O seguinte programa lê um vetor contendo n números reais imprime o desvio padrão, dp , dos n elementos. Obs.: $dp = \sqrt{\frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2 \right)}$

```

program ProgramaDesvioPadrao;
const MAX      = 100;
type TipoVetorReal = array[1..MAX] of real;
var i,n
    Soma,SomaQuadrado,DesvioPadrao : real;
    v      : TipoVetorReal;
begin
  { Leitura do vetor }
  write('Entre com a quantidade de elementos a ler: '); readln(n);
  for i:=1 to n do begin
    write('Entre com o ',i,'-ésimo elemento: '); readln(V[i]);
  end;
  { Cálculo do desvio padrão }
  if (n<=1) then DesvioPadrao := 0
  else begin
    Soma := 0.0; SomaQuadrado := 0.0;
    for i:=1 to n do begin
      Soma := Soma + v[i];
      SomaQuadrado := SomaQuadrado + Sqr(v[i]);
    end;
    DesvioPadrao := Sqrt( (1/(n-1)) * (SomaQuadrado-(1/n)*Sqr(Soma)));
  end;
  { Impressão do desvio padrão }
  writeln('O desvio padrão dos números lidos é: ',DesvioPadrao);
end.

```

Exemplo 6.7 (*Busca Seqüencial*) O seguinte programa lê o nome e a idade de vários alunos (no máximo 100). E em seguida o programa lê repetidamente um nome de aluno e imprime a idade deste aluno. O programa deve parar de ler quando for dado o nome 'fim', que não deve ter idade lida.

```

program LeImprimeVetorReal;
const MAX      = 100;
type
  TipoNome      = string[100];
  TipoVetorIdade = array[1..MAX] of integer;
  TipoVetorNome = array[1..MAX] of TipoNome;
var
  Vnome : TipoVetorNome;
  Vidade : TipoVetorIdade;
  n,i    : integer;
  Nome   : TipoNome;
  Achou  : boolean;
begin
  { Leitura dos nomes e idades dos alunos }
  n:=0;
  writeln('Leitura dos nomes e idades dos alunos : ');
  write('Entre com o nome de um aluno: '); readln(nome);
  while (nome<>'fim') and (n<MAX) do begin
    n:=n+1;
    Vnome[n] := nome;
    write(' Entre com a idade do aluno ',nome,' : '); readln(Vidade[n]);
    write('Entre com o nome de um aluno: '); readln(nome);
  end;
  { Leitura de nome e pesquisa (busca seqüencial) da sua idade }
  writeln('Pesquisa das idades dos alunos : ');
  write('Entre com o nome de um aluno para pesquisar: '); readln(nome);
  while (nome<>'fim') do begin
    i:=1;
    Achou := false;
    while (not achou) and (i<=n) do
      if (Vnome[i]=nome) then Achou:=true
      else i:=i+1;
    if (Achou) then writeln('A idade do aluno ',nome,' é: ',Vidade[i])
    else writeln('O nome do aluno não foi encontrado');
    write('Entre com o nome de um para pesquisar aluno: '); readln(nome);
  end;
end.

```

Exemplo 6.8 A matriz abaixo representa o Triângulo de Pascal de ordem 6 (6 linhas).

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

```

Os elementos extremos de cada linha são iguais a 1. Os outros elementos são obtidos somando-se os dois elementos que aparecem imediatamente acima e à esquerda na linha anterior (i.e., $10=4+6$). O seguinte programa lê um inteiro positivo n e imprime as n linhas do triângulo de Pascal.

```

program Pascal;
const TamMax = 100;
var n,i,j : Integer;
    prev,corr : array[0..TamMax] of Integer;
begin
  Readln(n); {0 <= n <= TamMax}
  prev[0] := 1; corr[0] := 1;
  for i:=1 to n do prev[i] := 0; {truque para evitar caso especial na diagonal}
  for i:=1 to n do begin
    for j:=1 to i do corr[j] := prev[j-1]+prev[j];
    for j:=0 to i do begin
      Writeln(corr[j]);
      prev[j] := corr[j];
    end
  end
end.

```

Obs.: Poderíamos usar “prev:=corr”, mas seriam copiados todos os valores, mesmo os que não são usados.

Exemplo 6.9 (Ordenação por Seleção) O seguinte programa lê um vetor de reais n números reais e o imprime em ordem não decrescente (menores primeiro). A estratégia usada é a mesma do algoritmo ordena3 apresentado no exemplo 7.4 na página 68. Neste caso, o algoritmo realiza $n - 1$ interações. Na primeira interação, o programa encontra um maior elemento do vetor, e troca este elemento com o elemento que está na n -ésima posição. Na segunda interação, o algoritmo encontra um segundo maior elemento (tirando o maior elemento, o segundo maior é um maior entre os demais) e troca com o elemento que está na $(n - 1)$ -ésima posição. Este processo se repete até que se tenha colocado o segundo elemento na posição correta. Naturalmente após estas $n - 1$ interações, um menor elemento deve estar necessariamente na primeira posição. Para encontrar um maior elemento, podemos usar a estratégia de busca apresentada no exemplo 6.5 (pg. 57).

```

program SelectionSort;
const MAX = 100;
type TipoVetorReal = array[1..MAX] of real;
var n,m,i,imax : integer;
    v : TipoVetorReal;
    aux : real;
begin
  { Passo 1: Leitura do valor n }
  write('Entre com a quantidade de elementos a ler: '); readln(n);
  { Passo 2: Leitura dos n valores reais no vetor }
  for i:=1 to n do begin
    write('Entre com o ',i,'-esimo elemento: '); readln(v[i]);
  end;
  { Passo 3: Ordenação do vetor }
  for m:=n downto 2 do begin
    { Encontra o m-ésimo maior elemento }
    imax := 1;
    for i:=2 to m do
      if (v[i] > v[imax]) then imax:=i;
    { Coloca o m-ésimo maior elemento na posição m, trocando valores de v[m] e v[imax] }
    aux := v[m]; v[m] := v[imax]; v[imax] := aux;
  end;
  { Passo 4: Impressão do vetor ordenado }
  for i:=1 to n do writeln(v[i]:10:3);
end.

```

Uma das vantagens de se usar este método é que ele faz no máximo $n - 1$ trocas entre elementos. Isto é vantajoso quando o tamanho de cada elemento (memória usada pelo elemento) é grande.

Exemplo 6.10 (*Intercalação de vetores*) Faça um programa que lê dois vetores ordenados, v_1 e v_2 , com n_1 e n_2 elementos, respectivamente e intercala os dois vetores gerando um terceiro vetor ordenado v_3 , com $n_3 := n_1 + n_2$ elementos da intercalação de v_1 e v_2 .

```
program Intercala;
const TamMax = 100;
      TamMaxRes = 200;
var n1,n2,i1,i2,k : Integer;
    v1,v2      : array [1..TamMax] of Integer;
    w          : array [1..TamMaxRes] of Integer;
begin
  Readln(n1,n2); {0 <= n1,n2 <= TamMax; seqüências ordenadas}
  for i1:=1 to n1 do Readln(v1[i1]);
  for i2:=1 to n2 do Readln(v2[i2]);
  { Os dois vetores devem ser lidos ordenados }
  i1 := 1; i2 := 1; k := 0;
  while (i1 <= n1) and (i2 <= n2) do begin
    k := k+1;
    if v1[i1] <= v2[i2] then begin
      w[k] := v1[i1];
      i1 := i1+1
    end else begin
      w[k] := v2[i2];
      i2 := i2+1
    end
  end;
  for i1:=i1 to n1 do begin w[k] := v1[i1]; k := k+1 end;
  for i2:=i2 to n2 do begin w[k] := v2[i2]; k := k+1 end;
  for i1:=1 to k do Writeln(w[i1])
end.
```

Exemplo 6.11 (*Busca Binária*) Note que no exemplo da busca seqüencial, descrita no exemplo 6.7 podemos ver que no pior caso, a rotina faz n comparações com elementos do vetor. Uma vez que temos um vetor já ordenado, é possível se fazer um algoritmo de busca mais eficiente.

A estratégia é a mesma que usamos para encontrar um determinado nome em uma lista telefônica (uma lista telefônica de uma grande cidade pode ter milhões de assinantes). Neste caso, partimos a lista telefônica em uma posição, verificamos se um nome desta posição é menor, maior ou igual ao elemento que estamos procurando. Se for menor ou maior, podemos descartar uma das partes da lista.

Assim, a estratégia que podemos usar é a de sempre pegar um elemento do meio do vetor (ou um dos elementos do meio) e comparar, se este elemento é menor, maior ou igual ao elemento que estamos procurando. Se for menor ou maior, podemos restringir a busca em apenas uma das partes do vetor.

```

program ProgBuscaBinaria;
const MAX      = 100;
type TipoVetorReal = array[1..MAX] of real;
var
  v                : TipoVetorReal;
  n,pos,i,esq,dir,meio : integer;
  x                : real;
begin
  writeln('Entre com o número de elementos a inserir: '); readln(n);
  { A seqüência lida deve estar ordenada. }
  for i:=1 to n do begin write('Entre com o ',i,'-ésimo elemento: '); readln(v[i]); end;
  { Pesquisa de um elemento }
  write('Entre com um elemento a pesquisar: '); readln(x);
  pos := 0; { Uma vez que o elemento tenha sido encontrado, pos terá o índice do elemento }
  esq:=1;
  dir :=n;
  while (esq<=dir) and (pos=0) do begin
    meio := (esq+dir) div 2;
    if (x<v[meio]) then dir := meio-1 { descarta parte da direita }
    else if (x>v[meio]) then esq := meio+1 { descarta parte da esquerda }
    else pos:=meio; { encontrou o elemento }
  end;
  if pos>0 then Writeln('O elemento ',x,' está na posição ',pos)
  else Writeln('Elemento não encontrado. ');
end.

```

A busca binária nos introduz a uma questão interessante. Quando usar a busca binária e quando usar a busca seqüencial?

Considere primeiro o algoritmo que usa busca seqüencial. Este algoritmo realiza no pior caso, no máximo n interações por busca. Assim, se necessitamos de m buscas, faremos em média uma quantidade de operações proporcional a $n \cdot m$ operações.

Para o caso da busca binária, note que é necessário se fazer inicialmente a ordenação do vetor. Vamos supor que usamos a estratégia do algoritmo *SelectionSort* para ordenar o vetor. Fazendo uma análise em alto nível, podemos ver que a rotina *SelectionSort* faz $n - 1$ interações, mas em cada uma delas há uma chamada da rotina *IndMaximo* que faz no máximo $n - 1$ interações. Assim, a grosso modo, a quantidade de operações total para ordenar não é maior que uma proporção de n^2 . Agora vamos analisar, também em alto nível, a busca binária no pior caso, que faz digamos k interações. Na primeira interação são considerados os n elementos. Na segunda interação são considerados $n/2$ elementos. Na terceira interação são considerados $n/4$ elementos, assim, por diante. Com isso a quantidade de operações realizadas não é maior que uma proporção de $\log_2(n)$. Considerando o tempo de ordenação mais o tempo de se fazer m buscas, temos um tempo que é delimitado por uma proporção de $n^2 + m \cdot \log_2(n)$ operações.

Assim, podemos dizer que os tempos de processamento total destas duas estratégias são próximos dos comportamentos de duas funções: $s(n, m) := c_s \cdot (n \cdot m)$ para a busca seqüencial e a função $b(n, m) := c_b \cdot (n^2 + m \cdot \log_2(n))$ para a busca binária (c_s e c_b são constantes adequadas). Note que para valores pequenos de m , a busca seqüencial faz muito menos processamento que a estratégia que usa busca binária. Por outro lado, para valores grandes de m (e.g., valores bem maiores que n), temos que a busca binária faz menos processamento que a busca seqüencial. Na seção 11 iremos estudar métodos mais eficientes para ordenação, o que faz com que métodos de busca usando a estratégia da busca binária fiquem bem atraentes.

6.2 Vetores Multidimensionais

A linguagem Pascal também permite a declaração de vetores de vetores. I.e., é um vetor onde cada elemento também é um vetor. Além disso, você pode declarar vetores de vetores de vetores ...

Vetores de vetores são comumente chamadas de matrizes, e correspondem a forma de matrizes que costumamos aprender no ensino básico. Matrizes podem ser declaradas como nas seguintes formas equivalentes:

$mat : \mathbf{array}[I1..F1] \mathbf{of} \mathbf{array}[I2..F2] \mathbf{of} \text{Tipo_de_Cada_Elemento};$

ou

$mat : \mathbf{array}[I1..F1, I2..F2] \mathbf{of} \text{Tipo_de_Cada_Elemento};$

Na figura 21 apresentamos uma representação gráfica da matriz, conforme a declaração acima. O elemento X pode ser acessado usando se a sintaxe $Mat[i, j]$.

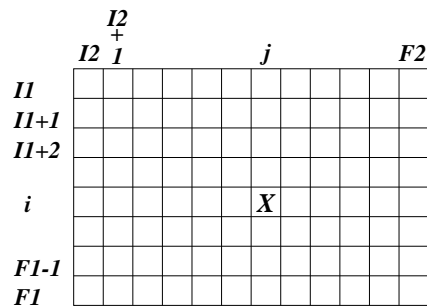


Figura 21: $Mat : \mathbf{array}[I1..F1, I2..F2] \mathbf{of} \text{Tipo_de_Cada_Elemento}$

Exemplo 6.12 O seguinte programa faz a leitura e a impressão de uma matriz de forma tabular. Para cada uma destas operações, o programa percorre todos os elementos a serem lidos/impressos com dois comandos **for** encadeados.

```

program matriz;
const MaximaDimensao = 100;
type TipoMatrizReal = array[1..MaximaDimensao,1..MaximaDimensao] of real;
    TipoMsg      = string[100];
var n,m,i,j : integer;
    mat      : TipoMatrizReal;
begin
    { Leitura das dimensões da matriz }
    write('Entre com o número de linhas da matriz: '); readln(n);
    write('Entre com o número de colunas da matriz: '); readln(m);
    { Leitura da matriz }
    writeln('Leitura dos elementos da matriz');
    for i:=1 to n do
        for j:=1 to m do begin
            write('Entre com o elemento (',i:3,', ',j:3,'): '); readln(mat[i,j]);
        end;
    { Impressão da matriz }
    for i:=1 to n do begin
        for j:=1 to m do write(mat[i,j]:5:1, ' ');
        writeln;
    end;
end.

```

Exemplo 6.13 O seguinte programa lê três inteiros, p , q e r ; lê duas matrizes A e B , de dimensões $p \times q$ e $q \times r$, respectivamente; e imprime a multiplicação da matriz A com a matriz B . Obs.: Vamos supor que temos descrito o código para ler e imprimir as matrizes, como descritos no exemplo 6.12.

```

program multmat;
const MAX = 10;
type
  tipomatrizreal = array[1..MAX,1..MAX] of real;
var
  Soma,A,B,C : TipoMatrizReal;
  p,q,r,i,j,k : integer;

begin
  { Passo 1: Fazer leitura das dimensões p,q,r }
  { Passo 2: Fazer leitura das Matrizes A e B }
  { Passo 3: Multiplicação das matrizes A e B em C }
  for i:=1 to p do
    for j:=1 to r do begin
      Soma := 0;
      for k:=1 to q do
        Soma := Soma + A[i,k]*B[k,j];
      C[i,j] := Soma;
    end;
  { Passo 4: Fazer impressão da matriz C }
end.

```

6.3 Exercícios

1. Faça um programa que lê as notas de n alunos, cada nota é um inteiro entre 0 e 100, e imprima a quantidade de vezes com que apareceu cada nota.
2. Faça um programa para ordenar um vetor de números reais de forma não crescente (i.e., os maiores primeiro).
3. Faça um programa que lê o nome, idade e salário de n pessoas (n lido). O programa deve ordenar e imprimir os dados destas pessoas ordenados pelo nome, pelo salário e pela idade (de maneira não decrescente).
4. Faça um programa que lê uma matriz 3×3 e imprime o determinante da matriz.
5. Escreva um programa que imprime um Triângulo de Pascal de ordem n (n lido). O programa deve usar apenas um vetor e apenas um comando de repetição **for**.
6. Uma matriz quadrada inteira é chamada de *quadrado mágico* se a soma dos elementos de cada linha, a soma dos elementos de cada coluna e a soma dos elementos das diagonais principal e secundária são todos iguais. Exemplo: As matrizes abaixo são quadrados mágicos:

$$\begin{bmatrix} 3 & 4 & 8 \\ 10 & 5 & 0 \\ 2 & 6 & 7 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Faça um programa que verifica se uma dada matriz quadrada lida é ou não um quadrado mágico.

7. Um número é dito ser palíndromo se a seqüência dos dígitos do número lidos da esquerda para a direita é igual a seqüência de dígitos lidos da direita para a esquerda. Por exemplo: Os seguintes números são palíndromos: 123454321, 54445, 789987, 121.
Faça duas versões de um programa que verifica se um número inteiro é ou não um número palíndromo, uma usando vetor e outra sem usar vetor.

8. Leia dois vetores v_1 e v_2 , com n_1 e n_2 elementos, respectivamente, onde cada elemento do vetor é um dígito. Faça um programa que realiza a multiplicação dos números representados por estes dois vetores colocando o resultado em um terceiro vetor. Obs.: Cada vetor de entrada tem tamanho máximo de 100 dígitos.
9. Um mapa apresenta n cidades definidas pelos números $1, 2, \dots, n$. Cada duas cidades i e j podem estar ligadas por uma estrada. Chamamos de componente um grupo maximal de cidades que estão ligadas por estradas (i.e., é possível ir de uma cidade para outra que esteja no mesmo grupo andando por estradas). Faça um programa que lê um valor n , quantidade de cidades, um valor m , quantidade de estradas, e m pares $(i_1, j_1), (i_2, j_2), \dots, (i_m, j_m)$. Cada par (i, j) indica que há uma estrada ligando as cidades i e j . O programa deve imprimir k linhas, onde k é a quantidade de componentes no mapa. Cada linha contém uma lista das cidades que estão em uma mesma componente.