4. VARIÁVEIS COMPOSTAS

Até o momento, vimos apenas variáveis que são utilizadas para o armazenamento de dados simples. Entretanto, as variáveis podem ser compostas, formadas por uma ou mais posições (campos ou entradas), podendo armazenar um ou mais dados. As variáveis compostas são classificadas em homogêneas (vetoriais) ou heterogêneas (registros). Em função de sua capacidade de armazenar diferentes valores, este tipo de variável pode ser encarado como uma "estrutura" de armazenamento. Entretanto, não deve ser confundido com as "estruturas de dados" que veremos mais adiante neste livro. O conceito estruturas de dados envolve não somente a estrutura de armazenamento, mas também regras de formação e manipulação, além de funções próprias de acesso.

4.1 Variáveis Vetoriais Unidimensionais

As variáveis vetoriais unidimensionais, chamadas muitas vezes simplesmente de variáveis vetoriais, são variáveis compostas homogêneas indexadas, as quais contêm um ou mais campos de mesmo tipo, onde cada campo é acessado pelo seu índice (posição). Sua declaração geral é definida da seguinte forma:

```
var <nome>:array[<inicio>..<fim>] of <tipo_da_variavel>;
```

Exemplo 1: Declaração de uma variável chamada A que pode armazenar 10 números inteiros.

```
var   A:array[1..10] of integer;
```

Neste exemplo, a variável A será armazenada na memória do computador em uma área equivalente a 10 vezes o que é armazenado para uma única variável inteira. Esta variável é representada pela figura 9. Nela, os números 1, 2, 3...10 são os índices (posição ou localização) de cada número inteiro que pode ser armazenado. O índice serve para localizar qual elemento do vetor estará sendo referido em determinado instante. Por exemplo, o elemento A[5] é localizado no campo de índice 5 do vetor A e, no exemplo da figura, vale -12.

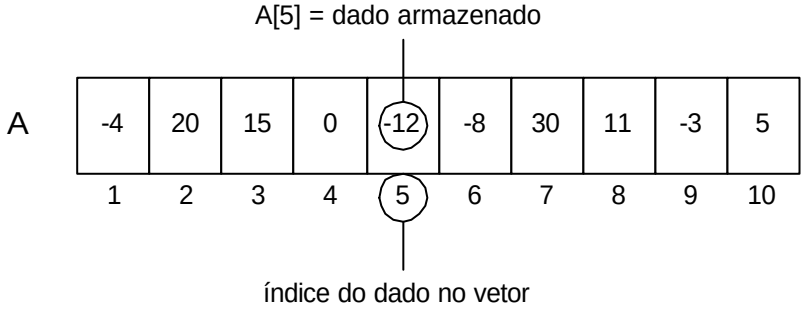


Figura 9 – Representação de uma Variável Vetorial

Para atribuir um valor qualquer para um campo de posição `i` faz-se o seguinte:

```
A[i]:=<valor_qualquer>;
```

Neste caso, diz-se que a posição i-ésima do vetor A foi indexada.

Exemplo 2: Algumas operações que podem ser realizadas com vetores.

```
B[3]:=´flor´;
```

```
A[6]:=A[8];
read(B[2]);
write(B[i]);
A[i]:=A[i+1]-C[j];
```

Exemplo 3: Trecho de programa que inicializa todos os campos de um vetor A com o valor 0 (zero). Diz-se "inicializar o vetor com zeros" ou simplesmente "zerar o vetor".

Primeira Opção: atribui-se 0 (zero) a todos os campos, um a um, da seguinte forma:

```
A[1]:=0;
A[2]:=0;
A[3]:=0;
A[4]:=0;
A[5]:=0;
A[6]:=0;
A[7]:=0;
A[8]:=0;
A[9]:=0;
A[10]:=0;
```

Segunda Opção: utiliza-se uma variável para indexar as 10 posições com um comando de repetição. A idéia aqui é usar a própria variável de contagem do comando `for` para indexar o vetor todo, da seguinte forma:

```
for i:=1 to 10 do
  A[i]:=0;
```

Neste exemplo dizemos que o vetor A foi completamente indexado (percorrido, visitado, pesquisado, varrido, ...) em todas as suas entradas e para cada uma um valor zero foi inserido. Dizemos que para cada iteração do for, ou seja, para cada valor de `i`, o elemento `i`-ésimo (elemento da vez, elemento corrente, elemento atual etc...) é zerado. A figura 10 mostra uma possível representação gráfica da indexação com comando de repetição.

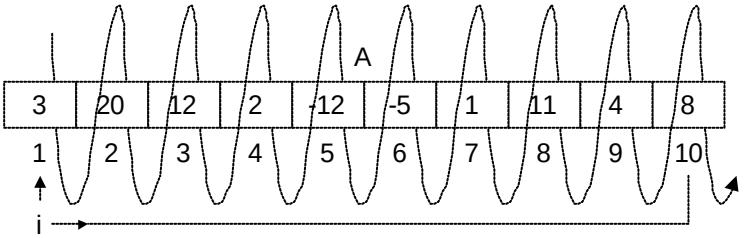


Figura 10 – Representação Gráfica da Indexação Vetorial com Repetição

Entretanto, esta varredura pode ser feita com os comandos de repetição `repeat` e `while`, conforme os modelos que seguem.

```
i:=0;
repeat
  inc(i);
  A[i]:=0;
until (i=10);
```

ou

```
i:=0;
while (i<>10) do
begin
  inc(i);
  A[i]:=0;
end;
```

Exemplo 4: Programa que lê 20 números inteiros e depois os escreve na ordem inversa. A idéia aqui é mostrar no vídeo os elementos começando da última posição e vindo em direção a primeira posição, usando para tanto o comando de repetição `for`.

```
program inversor_1;
var   conjunto:array[1..20] of integer;
      i:integer;
begin
  for i:=1 to 20 do
    read(conjunto[i]);
  for i:=1 to 20 do
    write(conjunto[21-i]);
end.
```

Observe neste exemplo que foram usados dois comandos `for`, um para a leitura do vetor e outro para a escrita do vetor. Pelo fato da escrita ser feita na ordem inversa do vetor, a indexação é feita pela expressão matemática "21-i", que dita a lei de formação da posição a ser acessada a cada iteração do `for`. Esta expressão é necessária porque a variável de contagem do `for` é incrementada a cada iteração (varia crescentemente) ao passo que a posição a ser acessada é decrementada.

Para descobrir a lei de formação, o programador deve analisar como o vetor deve ser acessado a cada iteração do laço. Note que quando i valer 1, a posição a ser acessada deverá ser a 20; quando i valer 2, a posição a ser acessada deverá ser a 19; quando i valer 3, a posição a ser acessada deverá ser a 18 e assim sucessivamente. Podemos observar que a posição a ser acessada é sempre igual a 21 menos o valor da variável i. Desta forma, a expressão matemática que rege esta indexação é 21-i, para i variando de 1 a 20. Note que com a aplicação desta fórmula a indexação do vetor resultará em uma variação decrescente de 20 até 1. Note também que se o tamanho do vetor fosse medido por uma variável n, a fórmula de indexação utilizada seria "n+1-i". Felizmente, o comando `for` pode trabalhar com variáveis decrescentes e este programa pode ser resolvido tal como o exemplo a seguir.

IMPORTANTE! Note que sempre que um vetor tiver que ser pesquisado em todas as suas entradas, o comando `for` pode ser utilizado já que o número de iterações é conhecido. Caso contrário, os comandos `repeat` e `while` são mais recomendados.

Exemplo 5: Programa que lê 20 números inteiros e depois os escreve na ordem inversa usando comando `for` decrescente. Neste caso, a indexação do vetor é diretamente feita pela variável de contagem do comando `for`.

```
program inversor_2;
var   conjunto:array[1..20] of integer;
      i:integer;
begin
  for i:=1 to 20 do
    read(conjunto[i]);
  for i:=20 downto 1 do
```

```
     write(conjunto[i]);
end.
```

Exemplo 6: Trecho de programa que calcula a média de um vetor V de 30 números inteiros. A idéia aqui é percorrer todas as entradas do vetor, usando o comando de repetição `for`, e acumular, a cada iteração do laço, o elemento que estiver sendo indexado pela variável de contagem. Podemos pensar assim: para cada valor da variável `i`, o programa acessa o elemento `V[i]` e soma-o com o valor da variável `media`, colocando o resultado final na própria variável `media`. No final, divide-se a somatória pelo número de elementos do vetor.

```
media:=0;
for i:=1 to 30 do
   media:=media+V[i];
media:=media/30;
writeln(media);
```

Exemplo 7: Trecho programa que a partir de um vetor A de n números inteiros, calcula a soma dos pares e a soma dos ímpares. A idéia aqui é semelhante a anterior, entretanto usam-se duas variáveis de somatória (`somapar` e `somaimpar`). Para cada elemento indexado, uma comparação é feita para determinar em qual das somatórias ele será acumulado.

```
somapar:=0;
somaimpar:=0;
for i:=1 to n do
   if ((A[i] mod 2)=0)
      then somapar:=somapar+A[i]
      else somaimpar:=somaimpar+A[i];
writeln(somapar,´ ´,somaimpar);
```

Exemplo 8: Trecho de programa que procura dentro de um vetor `A` de `n` inteiros o elemento mais próximo da média. Diga qual é a posição deste elemento.

```
media:=0;
for i:=1 to n do
   media:=media+A[i];
media:=media/n;
dif:= abs(A[1]-media);
pos:=1;
for i:=2 to n do
   if abs(A[i]-media)<dif)
   then begin
      dif:= abs(A[i]-media);
      pos:=i;
   end;
writeln(´A[´,pos,´]=´,A[pos]);
```

Exemplo 9: Trecho programa que a partir de um vetor numérico A de tamanho n, eleva os valores de cada campo deste vetor ao índice de sua posição. Assim, $A[1]:=(A[1])^1$, $A[2]:=(A[2])^2$, $A[3]:=(A[3])^3$ e assim sucessivamente até que $A[n]:=(A[n])^n$. A idéia aqui é percorrer o vetor com um comando `for` e para cada iteração usar um outro comando `for` para calcular uma potenciação.

```
for i:=1 to n do
   begin
```

```
    pot:=A[i];
    for j:=2 to i do
      pot:=pot*A[i];
    A[i]:=pot;
  end;
```

Exemplo 10: Trecho de programa que a partir de um vetor A de n números naturais não nulos, para cada campo A[i]=k, com 1≤i≤n, o programa lê k números inteiros e coloca a média deles novamente em A[i]. No final é impresso o vetor e a média geral de seus campos.

```
somatotal:=0;
for i:=1 to n do
  begin
    soma:=0;
    for j:=1 to A[i] do
    begin
      read(nro);
      soma:=soma+nro;
    end;
    A[i]:=(soma/A[i]);
    somatotal:=somatotal+A[i];
  end;
mediatotal:=somatotal/n;
for i:=1 to n do
  writeln(A[i],´ ´);
writeln(mediatotal);
```

Exemplo 11: Trecho de programa que calcula uma somatória de operações contidas em um vetor OP sobre 2 vetores A e B, contendo números reais. O vetor OP contém caracteres representando as quatro operações aritméticas básicas (* , + , - , /). Os elementos da somatória são obtidos aplicando as operações do vetor OP sobre os vetores numéricos, campo a campo, conforme mostra a figura 11. A idéia aqui é indexar os 3 vetores simultaneamente usando a mesma variável, já que os 3 vetores apresentam o mesmo alcance e a ıésima posição de um é associada com a ıésima posição do outro. Neste sentido, o comando for é bastante adequado.

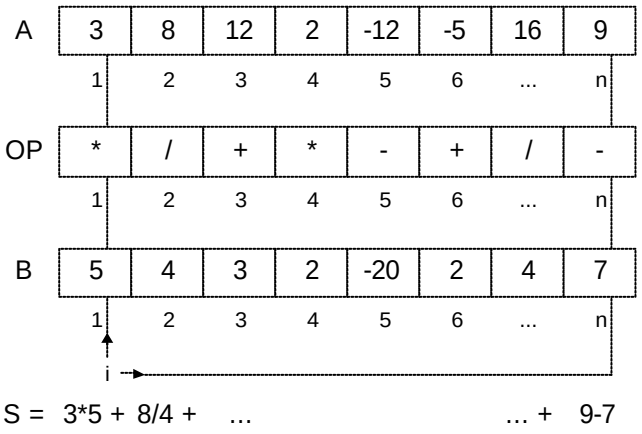


Figura 11 – Vetor de Operações Aplicado a Dois Outros Vetores

```
Soma:=0;
for i:=1 to n do
  begin
    case OP[i] of
      ´*´: soma:=soma+ A[i]*B[i];
```

```
       ´+´: soma:=soma+ A[i]+B[i];
       ´-´: soma:=soma+ A[i]-B[i];
       ´/´: soma:=soma+ A[i]/B[i];
     end;
  end;
writeln(soma);
```

Exemplo 12: Trecho de programa que procura um número `x` dentro de um vetor de `n` posições. Se encontrado, a sua posição dentro do vetor é mostrada. Observe que apenas a primeira ocorrência de `x` é procurada. A idéia aqui é utilizar um comando de repetição e uma variável de indexação para percorrer o vetor, campo a campo, até que o elemento seja encontrado ou até que o vetor tenha sido varrido em todas as suas entradas.

```
achou:=false;
acabou:=false;
i:=0;
read(x);
repeat
  inc(i);
  achou:=(x=V[i]);
  acabou:=(i=n);
until (acabou)or(achou);
if (achou)
  then writeln(´Encontrado na Posição ´,i)
  else writeln(´Nao Encontrado!´);
```

Neste exemplo foram utilizadas duas variáveis lógicas `achou` e `acabou`. A variável `achou` será verdadeira quando a primeira ocorrência de `x` for encontrada. A variável `acabou` será verdadeira quando a indexação do vetor atingir o máximo, ou seja, quando a variável `i,` que indexa o vetor `V,` atingir o valor de `n`. Estas variáveis são utilizadas para controlar o término da pesquisa, já que não é necessário pesquisar todo o vetor e, também, não é permitido acessar um campo além do limite de validade dos dados do vetor. Pelo fato de que o número de iterações não é conhecido, o comando `for` não é utilizado e sim o `repeat`. Poderia também ser usado um `while`. Mais uma observação é necessária: o uso das variáveis `achou` e `acabou` pode ser eliminado se as condições que elas representam, `x=V[i]` e `i=n`, forem colocadas diretamente na cláusula `until`. O motivo pelo qual estas variáveis são usadas é para deixar o programa mais legível/compreensível.

Conforme dito, este trecho de código também pode ser escrito sem as variáveis lógicas, usando tanto o comando `while` quanto o `repeat`, conforme abaixo:

```
read(x);
i:=1;
while (x<>V[i])and(i<=n) do
  inc(i);
if (i>n)
  then writeln(´Nao Encontrado!´)
  else writeln(´Encontrado na Posição ´,i)
```

ou `repeat`

```
read(x);
i:=0;
repeat
  inc(i)
```

```
until (x=V[i])or(i>n);
if (i>n)
   then writeln(´Nao Encontrado!´)
   else writeln(´Encontrado na Posição ´,i)
```

Exemplo 13: Trecho de programa que contabiliza a quantidade de ocorrências do número x dentro de um vetor de n posições. Quando encontrado uma ocorrência, a sua posição dentro do vetor é mostrada. Observe que neste caso todas as entradas devem ser pesquisadas, já que o elemento x poderá estar em qualquer posição válida.

```
read(x);
quant:=0;
for i:=1 to n do
begin
   if (x=V[i])
   then begin
      inc(quant);
      writeln(´uma ocorrencia encontrada em ´,i);
   end;
end;
writeln(´encontrado ´,quant,´ valores iguais a ´,x);
```

Exemplo 14: Trecho de programa que procura um número x dentro de um vetor de n posições cujos elementos estão dispostos em ordem crescente. Se encontrado, a sua posição dentro do vetor é mostrada. Observe que apenas a primeira ocorrência de x é procurada. Observe também que a pesquisa somente é feita enquanto o elemento x é maior que o elemento que está sendo visitado no vetor a cada iteração, pois como o vetor está ordenado, uma vez que um elemento maior que x é alcançado, a procura não é mais necessária.

```
achou:=false;
acabou:=false;
i:=0;
read(x);
repeat
   inc(i);
   achou:=(x=V[i]);
   acabou:=(i=n)or(x<V[i]);
until (acabou)or(achou);
if (achou)
   then writeln(´Encontrado na Posição ´,i)
   else writeln(´Nao Encontrado!´);
```

Neste exemplo, além das condições de finalização já apresentadas no exemplo 7, a pesquisa pelo elemento pode finalizar-se também quando a posição corrente do vetor (aquela que estiver sendo comparada com o elemento x) for maior do que x. Neste caso, não adianta mais procurar pois o vetor está disposto em ordem crescente e os próximos elementos a serem percorridos no vetor são todos maiores ou iguais a x.

Exemplo 15: Semelhante ao exemplo 9, entretanto os elementos repetidos deverão ser contados. Neste caso, o programa deve retornar o índice da primeira ocorrência de x e a quantidade de repetições que o mesmo apresenta na continuidade do vetor. A idéia aqui é encontrar a primeira ocorrência de x e depois continuar a percorrer o vetor enquanto existirem elementos iguais a x ou o vetor acabar.

```
achou:=false;
acabou:=false;
i:=0;
read(x);
repeat
  inc(i);
  achou:=(x=V[i]);
  acabou:=(i=n)or(x<V[i]);
until (acabou)or(achou);
if (achou)
  then begin
     j:=i;
     while (V[j]=x)and(j<=n) do
       inc(j);
     quant:=j-i;
     writeln(´Achou ´,quant,´ ocorrencias de ´,x,´ a partir da posicao ´,i);
  end
  else writeln(´Nao achou ´,x,´ no vetor´);
```

Exemplo 16: Trecho de programa que a partir um vetor numérico A, de tamanho na, soma cada k elementos adjacentes, colocando os resultados em um outro vetor B, tal como representado na figura 12. Note que na não precisa ser múltiplo de k, ou seja, a última seqüência de elementos somados de A pode não conter k elementos.

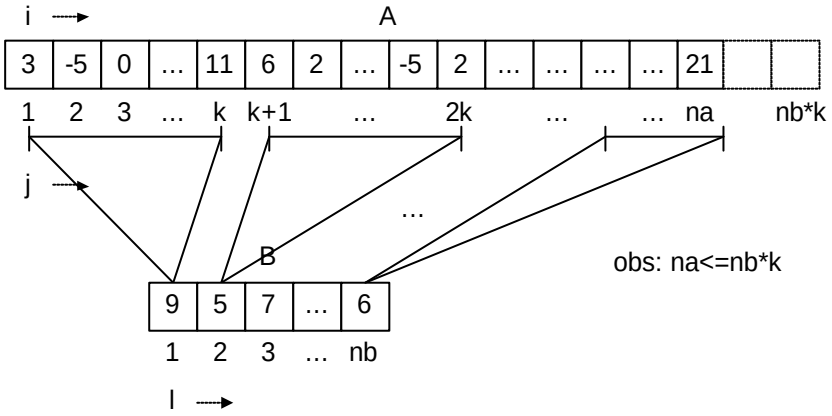


Figura 12 – Redução Vetorial por Junção de Elementos Adjacentes

```
read(k);
i:=0;
j:=0;
l:=0;
soma:=0;
repeat
  inc(i);
  inc(j);
  soma:=soma+A[i];
  if (j=k)
  then begin
    inc(l);
    B[l]:=soma;
    j:=0;
    soma:=0;
  end;
until (i=na);
```

```
if (soma<>0)
then begin
   inc(l);
   B[l]:=soma;
end;
nb:=l;
```

Exemplo 17: Trecho de programa que localiza a primeira ocorrência de uma seqüência de `nb` elementos (vetor `B`) dentro de uma seqüência de `na` elementos (vetor `A`), onde `na>nb`.

Exemplo 18: Trecho de programa para calcular o valor da função a seguir onde todos os elementos `ai`, com `1≤i≤n`, estão localizados em um vetor de `n` elementos:

$$f(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + ... + a_{n-1} x^2 + a_n x^1$$

```
read(x);
fx:=0;
for i:=1 to n do
   begin
      pot:=1;
      for j:=1 to (n-i+1) do
         pot:=pot*x;
      fx:=fx+A[i]*pot;
   end;
writeln(fx);
```

Exemplo 19: Trecho de programa que a partir de 2 vetores `A` e `B` (fontes), de tamanhos `na` e `nb`, respectivamente, gera um vetor concatenado `C` (destino) composto pela intercalação dos elementos de `A` e `B`, um a um. Em uma primeira solução, conforme exemplifica a figura 13 (note que não necessariamente o B é menor do que A), tenta-se primeiramente efetuar a intercalação enquanto existir elementos em ambos vetores fontes. Nesta parte do código, a variável `i` é usada pra indexar ambos os vetores `A` e `B`, já que a mesma posição de ambos é acessada a cada iteração do `for`. Note o vetor `C` é indexado usando as expressões matemáticas "`2*i-1`", para inserir elementos do vetor `A`, e "`2*i`", para inserir elementos do vetor `B`. Para se obter estas expressões a seguinte análise foi feita: quando `i` valer 1, as posições de `C` deverão ser 1 e 2; quando `i` valer 2, as posições de `C` deverão ser 3 e 4; quando `i` valer 3, as posições de `C` deverão ser 5 e 6 e assim, genericamente, para qualquer valor de `i`, as posições de `C` a serem acessadas serão `2*i-1` (o dobro menos 1) e `2*i` (o dobro).

Além disso, como os tamanhos `na` e `nb` podem ser diferentes, um dos vetores poderá acabar primeiro, e o restante do outro deverá ser descarregado sozinho. Para realizar este descarregamento, usa-se o comando `for` para indexar os elementos restantes do vetor que ainda não acabou. Como não se sabe qual dos dois vetores contém o restante, o programa implementa o descarregamento de ambos os vetores, entretanto, somente um deles é executado pois o outro tem a variável do comando `for` fora de alcance e não é executado. Note que a próxima posição a ser indexada no vetor restante é `i`-ésima e a próxima posição a ser inserida em C é (`2*i-1`)-ésima.

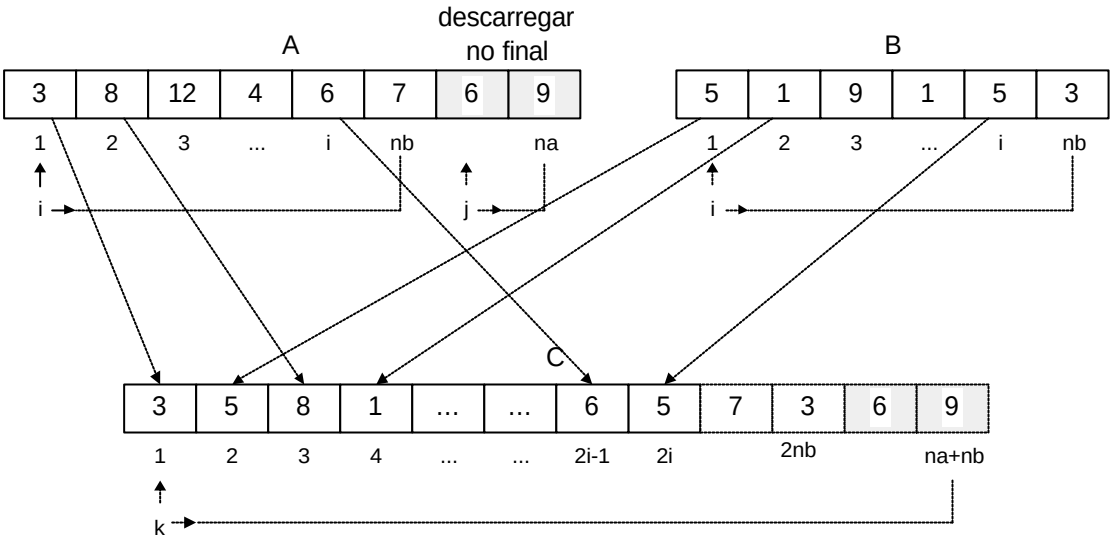


Figura 13 – Intercalação de Vetores: Primeira Solução

```
i:=1;
while (i<=na)and(i<=nb) do
begin
   C[2*i-1]:=A[i];
   C[2*i]:=B[i];
   inc(i);
End;
K:=2*i-1;
for j:=i to (nb) do
begin
   C[k]:=B[j];
   inc(k);
end;
for j:=i to (na) do
begin
   C[k]:=A[j];
   Inc(k);
end;
nc:=na+nb;
```

Em uma segunda solução, usa-se somente um comando de repetição `for`, com a variável `k`, já que é conhecido o número de elementos de C. Entretanto, cada vetor fonte é indexado separadamente com variáveis `i` e `j`. A idéia aqui é semelhante a anterior, com a diferença de que o descarregamento do restante é feito dentro do mesmo laço. As variáveis i e j somente serão incrementadas caso o respectivo vetor fonte ainda não tiver acabado. Para saber de qual vetor fonte o elemento é retirado, o programa verifica se um ou outro indexador i e j ainda não passou dos limites.

```
i:=1;
j:=1;
nc:=na+nb;
for k:=1 to nc do
begin
   if (i<=na)
   then begin
      C[k]:=A[i];
      inc(i);
   end;
   if (j<=nb)
   then begin
```

```
     C[k]:=B[j];
     inc(j);
   end;
end;
```

Exemplo 20: Trecho de programa que "dobra" ao meio um vetor de n elementos, sobrepondo a segunda metade do vetor sobre a primeira metade, de forma a somar os elementos opostos correspondentes, tal como mostra a figura 14. A idéia de solução aqui é percorrer a primeira metade do vetor (até `n div 2`) com uma variável de indexação e para cada elemento percorrido somar o seu oposto correspondente. Note que o oposto correspondente de 1 é o n, de 2 é o n-1, de 3 é o n-2, de 4 é o n-3 e assim, pode-se deduzir que o oposto correspondente do elemento i é o n-(i-1), ou seja, o n-i+1. Observe entretanto que o número de elementos do vetor pode ser par ou ímpar. No caso de ser ímpar, ficará um elemento central sem ter sido varrido, devendo este ser tratado a parte.

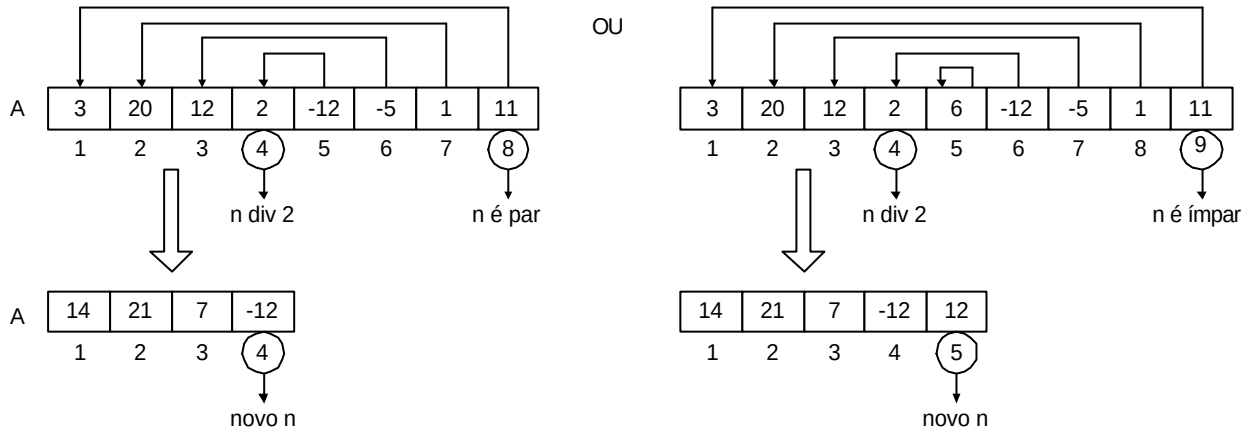


Figura 14 – Associação de Elementos Opostos Correspondentes

```
for i:=1 to (n div 2) do
  A[i]:=A[i]+A[n-i+1];
if ((n mod 2)=1)and(n>1)  {n eh impar: faltou o elemento central}
     then begin
       A[i+1]:=2*A[i+1];
       n:=(n div 2)+1; {poderia ser i+1}
     end
     else if (n>1)
            then n:=(n div 2); {poderia ser i}
```

Exemplo 21: Parte de programa que estende o exemplo anterior para reduzir o vetor em uma única posição. A idéia de solução aqui é muito simples. Basta inserir uma repetição mais externa ao código anterior de forma que o vetor A seja quebrado até que n=1.

```
repeat
for i:=1 to (n div 2) do
  A[i]:=A[i]+A[n-i+1];
if ((n mod 2)=1)and(n>1)  {n eh impar: faltou o elemento central}
     then begin
       A[i+1]:=2*A[i+1];
       n:=(n div 2)+1; {poderia ser i+1}
     end
     else if (n>1)
            then n:=(n div 2); {poderia ser i}
until n=1;
```

Exemplo 22: Trecho de programa que dado um vetor A de números com n elementos, altera seus campos, trocando entre eles os elementos opostos correspondentes.

```
for i:=1 to (n div 2) do
  begin
    aux:=A[i];
    A[i]:=A[n-i+1];
    A[n-i+1]:=aux;
  end;
```

Exemplo 23: Trecho de programa que faz a união de dois conjuntos. O objetivo é juntar (concatenar) os vetores A e B em um terceiro C (união) eliminado as repetições entre eles. Supõe-se que não existem repetições internamente em cada vetor. A idéia de solução aqui é descarregar primeiramente o vetor A no vetor C. Depois, descarregar o B verificando se cada um de seus elementos já não estava em A. Para descarregar um vetor em outro, varre-se todos os elementos do primeiro, usando um comando de repetição e uma variável de indexação que aumenta de 1 a cada iteração (o for neste caso é indicado) e a cada entrada alcançada do primeiro copia a mesma na próxima entrada do segundo vetor.

```
for i:=1 to na do       {Aqui o A é descarregado em C usando a mesma indexação}
  C[i]:=A[i];           {já que o C está vazio. na é o numero de elementos de ª}
nc:=na;                 {Neste momento C tem o mesmo número de elementos de A}
for i:=1 to nb do       {Vai descarregar o B agora}
  begin
    j:=1;               {indexa o A deste o início}
    while (B[i]<>A[j])and(j<=na) do  {procura B[i] dentro de A}
      inc(j);
    if (j>na)
      then begin        {percorreu todo o vetor A e não encontrou B[i] dentro de A}
        inc(nc);
        C[nc]:=B[j];    {Descarrega B[i] na próxima posição de C}
      end;
  end;
```

Exemplo 24: Trecho de programa que dado 2 vetores A e B de números ordenados, de tamanhos na e nb, junta-os em um único vetor concatenado C, ordenado e de tamanho na+nb. Neste exemplo, o fato dos vetores estarem previamente ordenados deve ser aproveitado. Note que o maior elemento de cada um está na posição final de cada vetor (em na e em nb), enquanto que os menores elementos estão nas primeiras posições. A idéia aqui é usar um comando de repetição para executar, repetidamente, a remoção do menor elemento entre A e B, e a sua respectiva inserção no vetor C, indexando o vetor A com i, o vetor B com j e o vetor C com k. A cada repetição, os elementos localizados nas posições correntes i e j são comparados, o menor elemento entre eles é descarregado na posição k do vetor C, o contador k é incrementado, o contador (i ou j) do vetor que contiver o menor elemento é incrementado para a próxima posição e o contador do vetor que contiver o maior elemento fica inalterado. Tal como o exemplo 16, aqui também são apresentadas duas soluções. A primeira solução utiliza um comando de repetição para concatenar A e B até que um dos vetores termine. Então, o restante é descarregado usando outro comando de repetição, tal como segue.

```
i:=1;
j:=1;
k:=1;
while(i<=na)and(j<=nb)do
```

```
begin
   if A[i]<B[j]
      then begin
         C[k]:=A[i];
         inc(i);
      end
      else begin
         C[k]:=B[j];
         inc(j);
      end
   inc(k);

end;
for l:=i to na do
begin
   C[k]:=A[l];
   inc(k);
end;
for l:=j to nb do
begin
   C[k]:=B[l];
   inc(k);
end;
nc:=na+nb;
```

Se neste ponto, o leitor estiver atento e realmente estiver conseguindo acompanhar este livro, notará que o trecho final do algoritmo anterior pode ser substituído pelo trecho a seguir.

```
for l:=i to na do
   C[k+l-i]:=A[l];
for l:=j to nb do
   C[k+l-j]:=B[l];
nc:=na+nb;
```

A segunda solução utiliza um único comando de repetição. Neste caso, torna-se necessário o uso de uma posição sentinela no final de cada vetor. A sentinela deverá ser um elemento maior do que todos os elementos dos dois vetores. A sentinela é utilizada para que nenhum vetor acabe antes do outro, garantindo que sempre exista pelo menos um elemento de cada vetor para serem comparados. No momento em que um dos vetores acabaria primeiro, ainda restará a sentinela para ser comparada com os elementos restante do vetor que ainda não acabou. No entanto, como a sentinela é maior do que todos os elementos válidos do vetor restante, os elementos deste serão menores e assim serão descarregados em `C`, até que restem somente as duas sentinelas para serem comparadas. Neste momento, a repetição termina porque somente `na+nb` elementos serão colocados em `C`.

```
sentinela:=abs(A[na])+abs(B[nb])+1;
A[na+1]:=sentinela;
B[nb+1]:=sentinela;
i:=1;
j:=1;
nc:=na+nb;
for k:=1 to nc do
begin
   if A[i]<B[j]
      then begin
         C[k]:=A[i];
         inc(i);
```

```
     end
     else begin
       C[k]:=B[j];
       inc(j);
     end
   inc(k);
end;
```

Exemplo 25: Trecho de programa que transfere os elementos do vetor A para o vetor B, descartando os elementos repetidos. Com isso, o vetor B poderá ser menor do que A, caso exista pelo menos 1 elemento repetido. A idéia de solução aqui é varrer o vetor A, elemento por elemento, comparando cada um com aqueles que já estiverem em B (inicialmente vazio). Caso o elemento ainda não estiver em B, ele será inserido em B. Caso contrário, ele será desconsiderado e o próximo elemento de A será analisado da mesma forma e assim sucessivamente. Para isto, é usado o comando `for` com uma variável de contagem `i` para varrer todo o vetor A. Para percorrer o vetor B, a procura de uma repetição, um outro comando de repetição é usado em conjunto com a variável de indexação `j`. O número de elementos de A é `na` e o número de elementos de B é `nb`, que inicialmente é nulo e a cada nova inserção ele é aumentado de 1. A figura 15 representa o esquema ora mencionado.

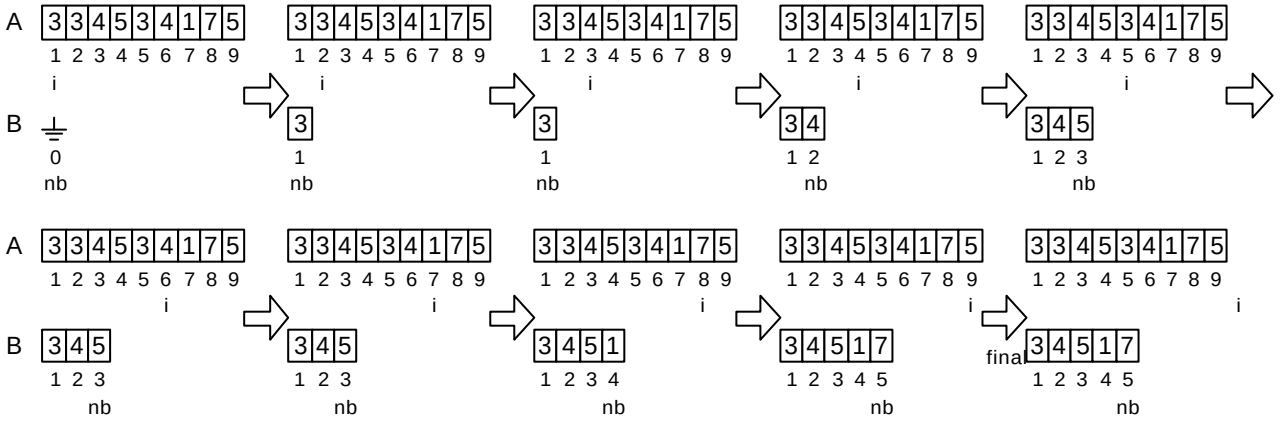


Figura 15 – Esquema de Eliminação de Repetições Usando 2 Vetores

```
nb:=0;
for i:=1 to na do
begin
  j:=1;
  while (j<=nb)and(A[i]<>B[j]) do  {enquanto não encontrou nenhum elemento}
    inc(j);                        {igual a A[i], continua percorrendo B}
  if (j>nb)
    then begin   {então A[i] nao esta em nenhuma posicao de B}
      inc(nb);
      B[nb]:=A[i];  { incrementa o tamanho de B de 1 e insere A[i]}
    end;
end;
```

Note que não existe a necessidade de se usar o vetor B. A compactação pode ser feita no mesmo vetor A, e B somente foi utilizado aqui por que este autor acredita que seja mais fácil de entender. Para modificar o programa de modo a efetuar a compactação no mesmo vetor basta substituir as ocorrências de B por A.

Exemplo 26: Programa que lê n nomes de pessoas e suas respectivas idades. Depois, o programa encontra a pessoa mais nova e a mais velha, informando seus respectivos nomes. Este problema é

resolvido utilizando duas variáveis vetoriais: uma para armazenar os nomes (tipo literal) e outra para armazenar as idades (tipo inteiro) de forma que a i-ésima posição do vetor de nomes esteja associada a i-ésima posição do vetor de idades, ou seja, estas entradas representam as informações de uma mesma pessoa. O vetor de idades é pesquisado para saber quais são a pessoa mais nova e a mais velha, guardando suas respectivas posições no vetor. Nestas mesmas posições do vetor de nomes estarão os respectivos nomes das pessoas mais nova e mais velha. A solução algorítmica é a mesma que foi usada na sessão 3.3.8, exemplo 4, com a diferença de que os números aqui não precisam ser lidos pois já estão no vetor.

```
program acha_novo_velho;
var  i, pos_novo,pos_velho,mais_novo,mais_velho:integer;
     idades:array[1..30] of integer;
     nomes:array[1..30] of string[64];
begin
   repeat
     read(n);
   until (n>0);
   for i:=1 to n do
     readln(nomes[i],idades[i]);
   mais_novo:=idades[1];
   mais_velho:=idades[1];
   for i:=2 to n do
     if  (mais_novo>idades[i])
     then begin
       mais_novo:=idades[i];
       pos_novo:=i;
     end
     else if (mais_velho<idades[i])
           then begin
             mais_velho:=idades[i];
             pos_velho:=i;
           end;
   writeln(´Mais Velho: ´,nomes[pos_velho]);
   writeln(´Mais Novo: ´,nomes[pos_novo]);
end.
```

Note que o uso das variáveis `mais_velho` e `mais_novo` é dispensável já que o programa mantém as posições onde estão o mais velho e o mais novo. Basta acessar as entradas do vetor `idades` indexadas por estas posições. O objetivo foi o de facilitar o entendimento do programa. Como sugestão, este autor sugere que você modifique o programa de forma a retirar as variáveis mencionadas.

Exemplo 27: Programa que dado um vetor A de número naturais, de tamanho n, imprime os elementos de forma ordenada, usando 3 soluções: 1) procura o menor elementos e o imprime, marcando-o como "inválido" para repetir a operação com os elementos restantes; 2) tal como a primeira solução, entretanto ao invés de marcar o elemento como "inválido" remove-o deslocando os elementos seguintes, reduzindo de 1 o tamanho do vetor; 3) ao invés de marcar o campo como inválido remova-o, preenchendo a posição do mesmo com o último elemento do vetor, reduzindo de 1 o tamanho do vetor e 4) troque de posição o menor com aquele da primeira posição. Repita o procedimento para a segunda posição. E assim sucessivamente.

4.2 Variáveis Vetoriais Multidimensionais

São variáveis homogêneas que possuem duas ou mais dimensões, ou se preferir, seus campos podem ser acessados por mais de 1 índice. Podemos imaginar como sendo tabelas de dados distribuídas no espaço. Conforme a dimensão, fica difícil a sua representativa gráfica. Sua sintaxe é definida abaixo:

```
var <nome>:array[<interv1>,<interv2>,....,<intervn>] of <tipo> ;
```

Onde `<intervi>` corresponde ao intervalo discreto da dimensão i, comumente composto de valores indicando o início e o fim da seqüência de elementos discretos que compõe o intervalo. A sintaxe da especificação do intervalo é:

```
<inicio>..<fim>
```

onde `<início>` e `<fim>` podem ser constantes ou variáveis contendo normalmente valores inteiros ou caracteres. Outros tipos menos comuns também são permitidos. Um exemplo de vetor Tetra-Dimensional capaz de armazenar valores reais é declarado abaixo, mas a sua representação gráfica é complicada.

Exemplo 1:

```
var   tetra:array[1..10,1..20,1..15,1..5] of real;
```

Já no caso de variável vetorial bidimensional, também chamada de matriz, sua representação pode ser vista no exemplo a seguir:

Exemplo2:

```
var  A:array[1..6,1..5] of integer;
```

Neste exemplo, o primeiro intervalo da dimensão é normalmente chamado de linha e o segundo de coluna, por questões de simplicidade de entendimento, já que estamos acostumados com esta representação tradicional da matemática escolar. No entanto, não existe esta obrigatoriedade e nada impede o programador de entender que o segundo intervalo seja a linha e o primeiro a coluna. O importante neste caso é padronizar o uso dentro do programa. Para todos os efeitos, melhor o entendimento tradicional.

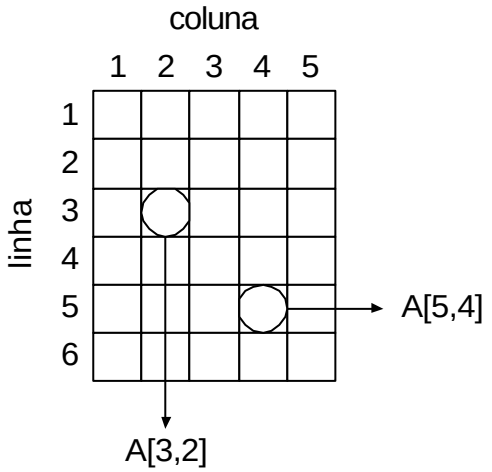


Figura 16 – Representação de uma variável matricial 6x5

Os campos de uma variável multidimensional podem ser acessados pela combinação de seus índices, também chamada de coordenada. O exemplo da figura anterior deixa em destaque dois campos da variável A, posicionados nas coordenadas (3,2) e (5,4). Cada campo pode ser manipulado da mesma forma que as variáveis de seu tipo. Para

Exemplo 3: Programa que preenche uma matriz mxn com elementos nulos.

```
program matriz_nula;
var  m,n,lin,col:integer;
     Max:array[1..200,1..200] of integer;
begin
   read(m,n);   {m linhas e n colunas}
   for lin:=1 to m do
     for col:=1 to n do
       Max[lin,col]:=0;
end.
```

Este exemplo mostra a forma mais comum de percorrer uma matriz mxn, usando dois comandos **`for`** para indexar todas as linhas e colunas da matriz. Note que o **`for`** mais externo é o responsável pela indexação das linhas e o mais interno das colunas. Assim, conforme feito neste exemplo, para cada linha o algoritmo percorre os elementos de todas as colunas e atribui a ele o valor zero.

Uma outra observação se faz necessária. Note que a matriz Max foi declarada com 40.000 campos inteiros. A declaração reserva espaço de memória para os campos mas o uso destes não é obrigatório. No programa, o que determina o tamanho válido da matriz é o controle que fazemos dela. Neste caso, usamos `m` e `n` para delimitar os campos válidos. Normalmente devemos reservar espaço suficiente para as situações de uso máximo, mas todo cuidado é pouco. Neste exemplo, 40.000 valores inteiros são 80.000 bytes, ou seja, aproximadamente 80K bytes. Imaginem agora se o tipo básico dos campos da matriz fosse real. Então teríamos uma reserva de 240K bytes, bem mais do que a memória disponível nos primeiros computadores. Este uso deve ser controlado, reservando o mínimo possível de memória, a fim de facilitarmos o trabalho o sistema operacional e permitir uma execução mais rápida do programa gerado.

Exemplo 4: Programa que lê uma matriz quadrada de ordem n e posteriormente imprime somente os elementos da diagonal principal.

```
program matriz_quadrada;
var  n,lin,col:integer;
     MQ:array[1..50,1..50] of integer;
begin
   read(m,n);   {n linhas e n colunas}
   for lin:=1 to n do
     for col:=1 to n do
       begin
         write(´Entre com o elemento [´,i,´,´,j,´] = ´);
         readln(MQ[lin,col]);
       end;
   for lin:=1 to n do
     for col:=1 to n do
       if (lin=col)
         then writeln(´Elemento [´,i,´,´,j,´] = ´,MQ[lin,col]);
end.
```

Ou

```
program matriz_quadrada;
var  n,lin,col,diag:integer;
     MQ:array[1..50,1..50] of integer;
begin
   read(n);   {n linhas e n colunas}
   for lin:=1 to n do
     for col:=1 to n do
       begin
         write(´Entre com o elemento [´,i,´,´,j,´] = ´);
         readln(MQ[lin,col]);
       end;
   for diag:=1 to n do
     then writeln(´Elemento [´,i,´,´,j,´] = ´,MQ[diag,diag]);
end.
```

Exemplo 5: Trecho de programa que lê uma matriz quadrada de ordem 3 e calcula o seu determinante.

```
for lin:=1 to 3 do
   for col:=1 to 3 do
      readln(A[lin,col]);
parte1:=A[1,1]*A[2,2]*A[3,3]+ A[1,2]*A[2,3]*A[3,1]+ A[1,3]*A[2,1]*A[3,2];
parte2:=A[1,3]*A[2,2]*A[3,1]+ A[1,2]*A[2,1]*A[3,3]+ A[1,1]*A[2,3]*A[3,2];
det := parte1-parte2;
writeln(´Determinante = ´,det);
```

Exemplo 6: Trecho de programa que lê uma matriz quadrada de ordem n e imprime a soma dos elementos de cada coluna.

```
read(n);
for lin:=1 to n do
   for col:=1 to n do
      readln(A[lin,col]);

for col:=1 to n do
   begin
     somacol:=0;
     for lin:=1 to n do
      begin
        somacol:=somacol+A[lin,col]);
      end;
     writeln(´Coluna ´,col,´ soma ´,somacol);
   end;
```

Exemplo 7: Trecho de programa que calcula a soma entre duas matrizes quadradas de ordem n.

```
for lin:=1 to n do
   for col:=1 to n do
      C[lin,col]:=A[lin,col]+B[lin,col];
```

Exemplo 7: Trecho de programa que verifica se uma matriz quadrada de ordem n é simétrica.

```
sim:=true;
```

```
for lin:=1 to n do
   for col:=1 to n do
     if A[lin,col] <> A[col,lin]
       then sim:=false;
if (sim=true)
   then writeln(´matriz simétrica´)
   else writeln(´matriz assimétrica´);
```

ou

```
sim:=true;
for lin:=1 to n do
   for col:=(lin+1) to n do
     if A[lin,col] <> A[col,lin]
       then sim:=false;
if (sim=true)
   then writeln(´matriz simétrica´)
   else writeln(´matriz assimétrica´);
```

ou ainda

```
lin:=0;
repeat
   inc(lin);
   col:=lin;
   repeat
     inc(col);
     sim:=(A[lin,col]=A[col,lin]);
   until (col=n)or(not sim);
until (lin=n)or(not sim);
if (sim)
   then writeln(´matriz simetrica´)
   else writeln(´matriz assimetrica´);
```

Exemplo 8: Trecho de programa que calcula o produto entre 2 matrizes Amxn e Bnxp.

```
for lin:=1 to m do
 for col:=1 to p do
    begin
      prod:=0;
      for k:=1 to n do
        prod:=prod+A[lin,k]*B[k,col];
      C[lin,col]:=prod;
    end;
```

8: Faça um programa que leia uma matriz quadrada de ordem N e calcule a soma de todos os elementos que estão abaixo da diagonal principal.

12: Escrever um programa que gere a seguinte matriz:

1	1	1	1	1	1
1	2	2	2	2	1
1	2	3	3	2	1
1	2	3	3	2	1

1	2	2	2	2	1
1	1	1	1	1	1

13: Faça um programa que leia uma matriz quadrada de ordem N, e posteriormente altere seus elementos, dividindo cada elemento de uma linha pelo elemento desta linha que pertence a diagonal principal. Faça isso para todas as linhas. No final, deve ser impresso a matriz resultante.

Exercícios

1. Faça um programa em Pascal que leia um vetor de tamanho N e calcule a média, encontre a posição do menor e do maior. O programa também deve localizar o elemento mais próximo da média.
2. Dado 2 vetores A e B de mesmo tamanho, faça um programa que troque os elementos dos vetores entre si, ou seja, s elementos de A são colocados em B e os de B em A.
3. Calcule o produto de 2 vetores de inteiros A e B de forma a gerar um único valor numérico. Multiplique campo a campo correspondente entre os dois vetores e some com as demais multiplicações dos outros campos correspondentes.
4. Faça um programa para multiplicar 2 matrizes.
5. Faça um programa em Pascal que leia um vetor A de números inteiros positivos até que seja digitado um número negativo. Após, o vetor deve ser reduzido pela metade, da seguinte forma. O termo A[1] deve ser somado com o termo A[N], o A[2] com o A[N-1] e assim por diante. Se houver elemento central este deverá ficar inalterado.
6. Faça um programa em pascal que leia um vetor de tamanho N e inverta a ordem dos elementos.
7. Faça o mesmo com uma matriz B. Neste caso leia a ordem da matriz (MxN) antes de ler os seus elementos. Depois, faça com cada linha da matriz aquilo que foi feito na questão 1. Neste caso, a matriz resultante deverá conter o mesmo número de linhas, mas terá reduzida pela metade o número de colunas.
8. Ídem ao exercício 2 com as colunas. Neste caso, a matriz resultante deverá conter o mesmo número de colunas, mas terá reduzida pela metade o número de linhas.
9. Faça um programa em Pascal que leia M, N e uma matriz A (MxN). Depois, o programa deverá trocar as linhas opostas correspondentes entre si. Por exemplo, a linha 1 deve ser trocada com a linha M, a linha 2 com a linha (M-1) e assim por diante. Cuidado para não destrocar aquilo que acabou de ser trocado. Se houver uma linha central, ela deverá permanecer inalterada.
10. Ídem ao 4 mas com as colunas.
11. Faça um programa em Pascal para descarregar uma matriz de ordem MxN em um vetor, linha por linha.
12. Ídem ao anterior, mas coluna por coluna.
13. Ídem aos exercícios 6 e 7, mas diagonal por diagonal (não somente a principal, mas todas as secundárias também). Neste caso, considere uma matriz quadrada NxN.
14. Faça um programa em Pascal para ordenar um vetor de tamanho N.
15. Ídem ao anterior, embora separando os números pares e ímpares, da seguinte forma. Os números pares devem ser ordenados no início do vetor e os números ímpares logo após.
16. Faça um programa que dado 3 vetores A, B e C de tamanhos M, N e P já ordenados, crie um quarto vetor R de tamanho (M+N+P) formado pela concatenação dos vetores A, B e C, de forma que R também permaneça ordenado (semelhante a aquele feito em sala de aula com 2 vetores). Sugestão: percorra cada vetor com um contador (indexador) diferente (i, j e k). Em cada momento você terá que comparar 3 valores (um de cada vetor). Compare-os e insira o menor no vetor resultante. Avance o contador do vetor que tinha o menor valor e do vetor resultante. Observe que os vetores irão se

esgotando em tempos diferentes. Inicialmente você terá 3 vetores com dados, depois restarão 2, que ainda precisarão ser comparados e no final restará somente 1 para descarregar o restante dos dados.

17. UltraEspeciais: Supondo uma matriz MxN, onde: todos os elementos de cada linha estão ordenados; as linhas não estão ordenadas entre si e não existe nenhuma relação entre elas. A partir desta matriz gere um vetor de tamanho N^2 formado pela concatenação das linhas da matriz. Da mesma forma que o exercício anterior, você deverá percorrer cada linha com um contador específico (faça um vetor de contadores i[1..M]). Insira o menor deles no vetor resultado e avance o contador do vetor resultante e da linha onde estava o menor elemento.

18. Suponha duas matrizes A e B com o mesmo número de linhas. Coloque as duas juntas lateralmente e ordene-as de forma que os elementos são ordenados dentro de cada linha e entre as linhas e, além disso, entre duas linhas adjacentes, os elementos da matriz posicionada a direita são posteriores aos da matriz posicionada a esquerda.

Variáveis Compostas Heterogêneas

São variáveis que podem armazenar um conjunto de informações de tipos diferentes. São também chamadas registros e sua sintaxe é mostrada abaixo:

```
Var   <nome_do_registro>:Record
                            <campo1>:<tipo1>;
                            <campo2>:<tipo2>;
                                :       :
                         End;
```

Exemplo 1: Declarar um Variável que possa armazenar o cadastro de um aluno.

```
Var   cadastro:Record
                 nome,
                 endereço,
                 RA:String;
                 idade:Integer;
               End;
```

A Variável acima possui quatro campos e pode ser representada abaixo:

cadastro

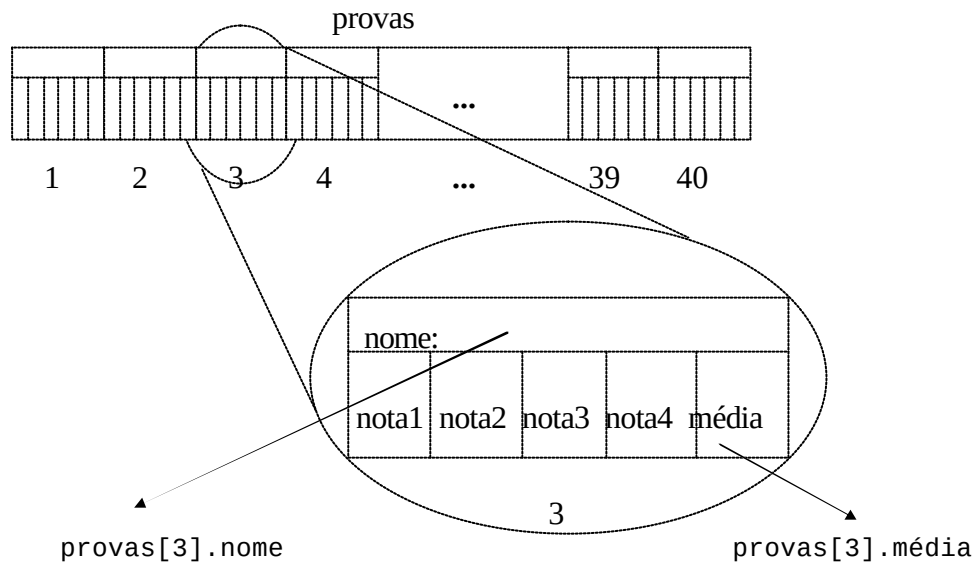
nome:	
endereço:	
RA:	idade:

Para acessarmos um determinado campo devemos indicar o nome do registro seguido de ponto e do nome do campo, conforme exemplos abaixo:

```
cadastro.nome := ´Roberta Miranda´;
write(cadastro.idade);
end := cadastro.endereço;
```

Exemplo 2: Declare uma estrutura que possa armazenar as notas de uma turma de quarenta alunos. Represente graficamente.

```
Var   provas:Array[1..40] of Record
                               nome: String;
                               nota1,nota2,nota3,nota4,media:Real;
                             End;
```



Exemplo 3: Faça um trecho de programa que leia as notas dos alunos, armazenando as informações na variável declarada na questão anterior, e calcule a média.

```
                :
For i:=1 To 40 Do
  Begin
    Read(provas[i].nome);
    Read(provas[i].nota1);
    Read(provas[i].nota2);
    Read(provas[i].nota3);
    Read(provas[i].nota4);
    provas[i].média:=(provas[i].nota1+provas[i].nota2+
                      provas[i].nota3+provas[i].nota4)/4;
  End;
```

O Comando `with`

Para diminuir o tamanho dos nomes das variáveis que utilizam registros, pode ser utilizado o comando **With**, que possui o seguinte formato:

```
With <nome_do_registro> Do
 Begin
    <lista_de_comandos>
 End;
```

Exemplo 1: Refaça o exemplo anterior usando `with`.

```
              :
for i:=1 to 40 do
  with  provas[i] Do
    begin
      read(nome,nota1,nota2,nota3,nota4);
      média:=(nota1+nota2+nota3+nota4)/4;
    End;
```

Neste exemplo, as variáveis 'nome, nota1, nota2, nota3, nota4 e média' se referem aos campos do registro provas[i].

Exercício: Faça um programa que leia um conjunto de nomes com suas respectivas idades e posteriormente mostre o nome do mais velho e do mais novo, usando registros.

```
Program Mais_Velho_Mais_Novo;
Var  N,i,mais_velho,mais_novo,pos_velho,pos_novo:Integer ;
     Cadastro:array[1..100] of record
                                 nome: String ;
                                 idade: Integer ;
                               End ;
Begin
  read(N);
  for i:=1 to N do
    begin
      read(cadastro[i].nome);
      read(cadastro[i].idade);
    end;
  pos_novo:=1;
  pos_velho:=1;
  for i:=2 to N do
    if cadastro[pos_velho].idade<cadastro[i].idade
      then pos_velho:=i
      else if cadastro[pos_novo].idade>cadastro[i].idade
              then pos_novo:=i;
  write(cadastro[pos_novo].nome,cadastro[pos_velho].nome);
end.
```

Definição de Constantes

Constantes contém valores que não podem ser alterados durante a execução do programa. Sua finalidade principal é facilitar a alteração futura do programa. Sua sintaxe básica pode ser vista abaixo:

```
const <nome> = <valor>;
```

Exemplo1:

```
Const tam = 40 ;
      N = 20 ;
```

IMPORTANTE!!! Para facilitar a manutenção de programas devemos sempre utilizar a constante ao invés do valor propriamente dito, pois assim, sempre que desejarmos alterar todos os valores iguais alteraremos somente a definição da constante.

Definição de Tipos

Para facilitar a estruturação e legibilidade dos programas, bem como facilitar a declaração de variáveis de tipos complexos iguais, podemos utilizar o recurso de definição de tipos. Baseado nos tipos primitivos (inteiro, real, string e lógico) podemos definir tipos mais complexos para declarar as variáveis.

Para definirmos um tipo temos a seguinte sintaxe básica:

```
type <nome>=<tipo>;
```

Exemplo: Definir um tipo Recorde declarar variáveis com este tipo.

```
program Exemplo ;
const  N = 100 ;
type  tipo_reg = record
                   nome: string[20];
                   idade: integer;
                 end ;

var  ficha:tipo_reg;
     fichario:array[1..N] of tipo_reg;
```

Devemos ter em mente que a declaração de tipos não ocasiona a reserva de espaço de memória, mas somente uma informação para quando o programas for processado. No exemplo anterior, 'tipo_reg' somente informa a estrutura das variáveis 'ficha' e 'fichário'. Poderíamos ter também definido um tipo para a Var 'fichário', da seguinte forma:

```
program Exemplo2 ;
const  N = 100 ;
type   tipo_reg = record
                    nome: string [20];
                    idade: integer;
                  end ;
       tipo_fichario = Array[1..N] of tipo_reg ;

var  ficha1,ficha2:tipo-reg ;
     fichário1, fichário2:tipo_fichário ;
```

Exercícios:

1) Declare uma variável registro para cada entidade abaixo:

a) professor b) automóvel c) cheque bancário

2) Defina um tipo registro, que possa armazenar dados de sobre computadores, que possa conter: CPU (286, 386, 486, Pentium etc...), Vídeo (EGA, CGA, VGA, SuperVGA etc...), Velocidade (33, 40, 66, 100MHz etc...), RAM (4Mb, 8Mb, 16MB etc...), HD (120Mb, 240Mb, 540Mb etc...) e Impressora (matricial, jato de tinta, laser etc...). Defina agora, um tipo vetorial do tipo registro definido anteriormente, com 500 campos.

3) Faça um programa que declare uma variável do tipo vetorial definido na questão anterior e que leia repetidamente dados sobre computadores até que seja lido o valor 111 para CPU. Quando isto ocorrer, os outros campos do registro não devem ser lidos.

4) Faça um trecho de programa que utilize o vetor lido na questão anterior e imprima a relação de CPU diferentes com suas respectivas quantidades.