Introdução a Programação em OX

Leonardo Costa
Reinaldo Marques
Departamento de Estatística - UFMG

-- Versão ainda não editada --

A palavra **“algoritmo”** em Ox é dado pelo comando **main()**. É necessário carregar alguns pacotes antes de rodar um algoritmo em Ox, no nosso caso não será tão necessário ainda porque não estaremos utilizando pacotes específicos, esses pacotes são carregados quando se ativa o **include**<>

É importante saber que toda linha de programação em Ox quando terminada (comando de for e if não precisa) deve ser acompanhada de “;”.

Declaração de variáveis em Ox.

```
decl n, a, nota;
```

É possível utilizar um identificador como uma constate. Por exemplo, pode-se querer definir o identificador “nome” como sendo igual a “Bruna”, ou “Reinaldo”. Assim, torna-se impossível alterar o conteúdo do identificador nome. Essa constante deverá ser criada na declaração de variáveis e seu comando será:

```
const decl nome =  “Bruna”;

main(){
}
```

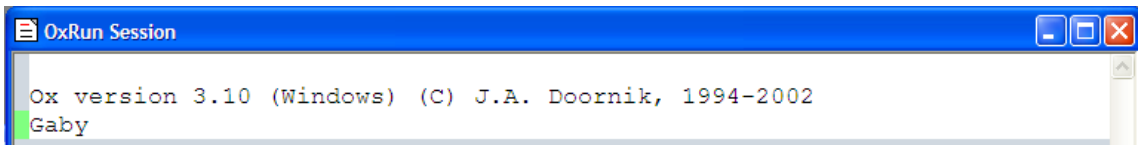
Esse comando de declaração de variáveis constantes deve ser antes da entrada em main(), dessa forma, todas vez que se utilizar o identificador nome dentro do programa, ele conterá o string “Bruna” que não poderá ser modificado ao longo do algoritmo criado.

Ex.

```
#include <oxstd.h>

const decl nome = "Gaby";
main()
{
   print(nome);
}
```

A saída será:



não é necessário dizer qual o tipo de cada variável (inteiro, numérico, literal ou lógico)

Comando de atribuição

O comando de atribuição no Ox será dado por “=”
Nota = 10
Nome = “cachorro”; etc.

Obs.: O Ox diferencia as variáveis "nome" de "Nome", ou seja, deve-se tomar cuidado na digitação das variáveis utilizadas no programa.

Operadores Lógicos

Os operadores lógicos são utilizados para comparação entre um ou mais elementos. Eles são utilizados em operações matriciais, estruturas condicionais e estruturas de repetição. Abaixo apresenta-se um quadro com os principais operadores lógicos.

Operador	Operações
<	menor que
>	maior que
<=	menor igual a
>=	maior igual a
==	igual a
!=	não igual a
&&	E
||	OU
!	NÃO

Vetores e Matrizes

vetor e matriz podem ser declarado normalmente como uma variável qualquer e, posteriormente, utilizado como matriz como nos exemplos abaixo:

Ex.1

```
decl x;
x = <1:3>;
print(x) # a saída será um vetor linha de forma (1.0000,
2.0000, 3.0000)
```

Obs.: separador decimal no Ox é o ponto (1.0 = 1)

Ex.2

```
decl x;
x = <1,2,3>
print(x)  # a saída será um vetor linha de forma (1.0000,
2.0000, 3.0000)
```

Ex. 3

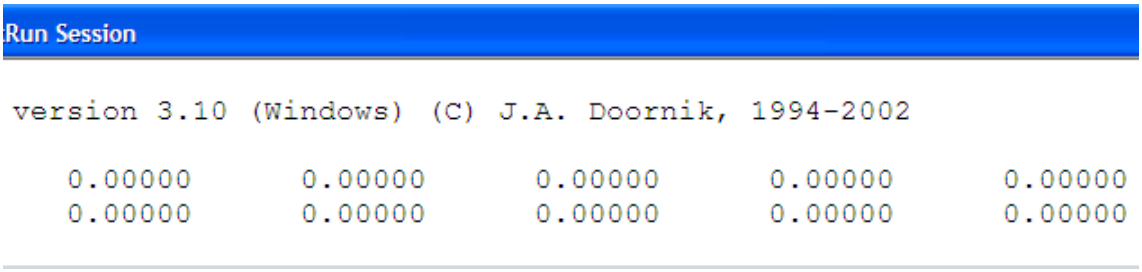
```
decl x;
x = <1;2;3>
print(x) # a saída será uma matriz de 3
colunas e uma linha:
```

```
x
1.0000
2.0000
3.0000
```

Um forma de se obter as dimensões da matriz que você acabou de criar é a utilização dos comando **sizec(matriz)** ou **columns(matriz)** # para verificar o número de colunas e o comando **sizer(matriz)** ou **rows(matriz)** # para verificar o número de linhas.

Ex. 4

`print(zeros(2,5)) # o comando` **zeros(b, c)** `cria uma matriz com b linhas e c colunas todos os elementos sendo zero. No exemplo acima a saída será:`

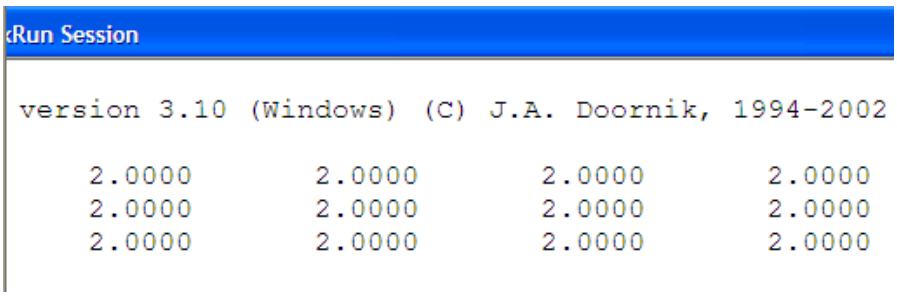


Demais funções com matriz:

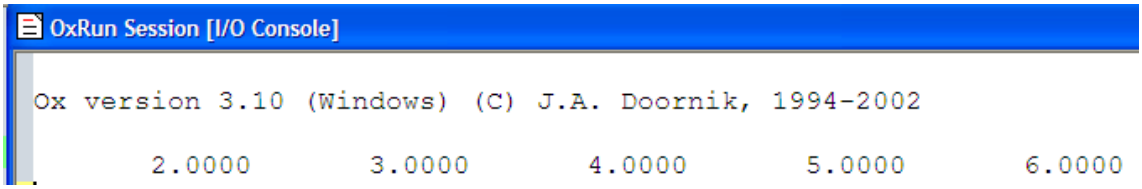
ones(b,c) cria uma matriz com b linhas e c colunas (todos os elementos da matriz iguais a 1)

unit(p) cria uma matriz identidade com p linha e p colunas

constant(k,b,c): cria uma matriz de constantes k de dimensão b x c. Abaixo uma saída utilizando o comando constant(2,3,4). Foi criado uma matriz com 3 linhas e 4 colunas.



range(i, j): cria um vetor com elementos que vão de i ao j. Saída obtida com o comando range(2,6):



Concatenação em matrizes

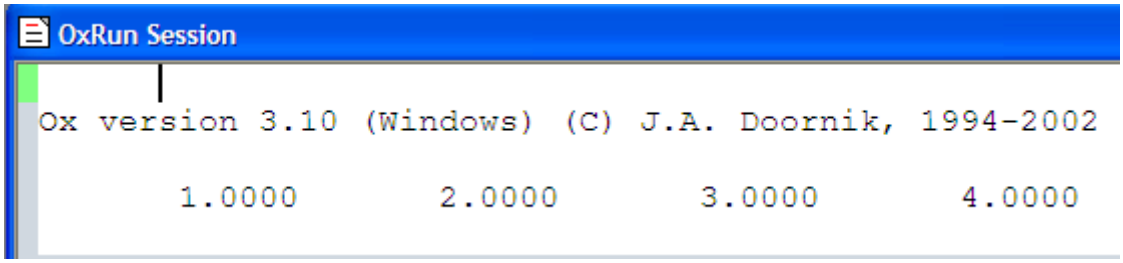
Existem dois comandos para concatenar elementos de matriz, “~” e “|”. O comando “~” faz uma concatenação horizontal e o comando “|” faz uma concatenação vertical.

Exemplos:

Ex. 1:

Declare x, e y, e construa dois vetores, o vetor x será: (1,2) e o vetor y será o vetor (3,4), utilizando então o comando “~” de concatenação vertical, teremos o seguinte algoritmo e resutlado:

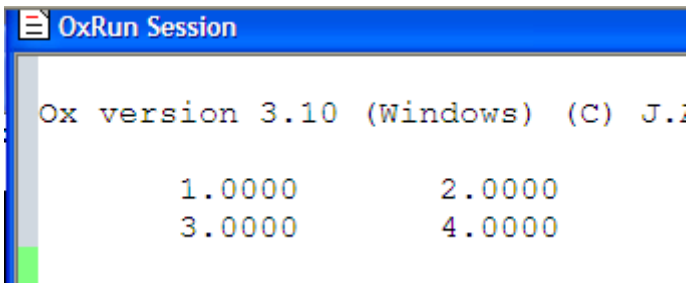
```
decl x,y;
x = <1,2>;
y = <3,4>;
print(x ~ y);
```



Ex. 2:

Construindo os mesmos vetores x e y, ou seja, x = (1,2) e y = (3,4) mas utilizando agora o comando de concatenação “|”. O algoritmo e a saída serão:

```
decl x,y;
x = <1,2>;
y = <3,4>;
print(x | y);
```



Ou seja, no primeiro exemplo, o Ox “colou” os vetores como linhas, primeiro o vetor x e depois o vetor y. No segundo exemplo, “colou-se” primeiro o vetor x, e depois como uma outra linha, o vetor y.

Algumas funções de operações com vetores e matrizes:

Operadores	Operação
‘	transpor, X´y (ou usar o comando X'*y
^	elevar a matriz a uma potência
*	Multiplicação matricial
/	divisão matricial
+	somar
-	subtrair
~	concatenar horizontalmente
|	concatenar verticalmente
invert(X)	inversa da matriz X
invertgen(X)	inversa generalizada da matriz X

Operações elemento por elemento

Operadores	Operação
.^	elevar elemento por elemento
.*	multiplicar elemento por elemento
./	dividir elemento por elemento
+	somar
-	subtrair

Estrutura Condicional

A estrutura condicional simples, no Ox é feita por

if(condição)
{
  Seqüência de comandos
}

Estrutura condicional composta:

if(condição)
{
  Seqüência de comandos A
}
else
{
  Seqüência de comandos B
}

if(condição)
{
  Seqüência de comandos A
}
else if(condição)
{
  Seqüência de comandos B
}

Estrutura de repetição

Existem basicamente dois tipos de estruturas para repetição no Ox, while e for. Primeiramente veremos o comando de repetição for.

for(inicialização; condição(enquanto); incrementação)
{
  Seqüência de comandos
}

Exemplo:

```
decl i;
for(i = 0;  i < 4; ++i)
{
   print(i, " ");
}
```

Saída deste comando:

Observe que o incremento da função for pode ser feita de uma outra forma um pouco mais intuitiva.

Exemplo:

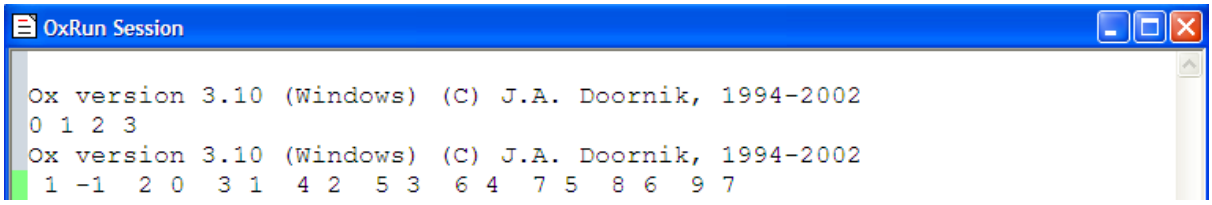
```
decl i;
for(i = 0; i < 4; i = i + 1)
{
   print(i, " ");
}
```

A saída será a mesma apresentada na figura acima.
Dentro da estrutura do for, é possível se obter dois índices simultâneos.

Exemplo:

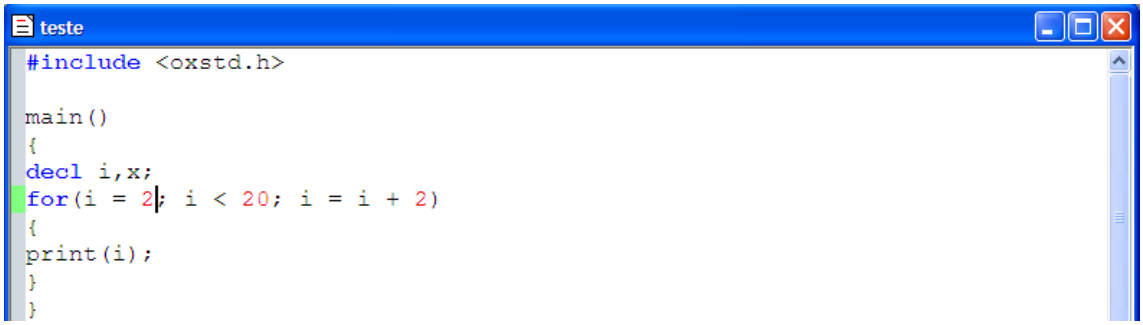
```
main()
{
decl i,j;
for(i = 1, j = -1; i < 10 && j <=9; ++i, ++j)
     {
   print("",i);
   print(“ ”);
   print(j,” “);
     }
}
```

Saída do algoritmo acima:

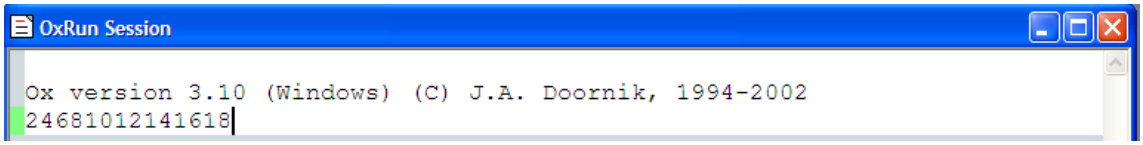


O índice do for pode ser alterado conforme o desejo do programador, exemplo, caso queira se incrementar o índice de 2 em 2.

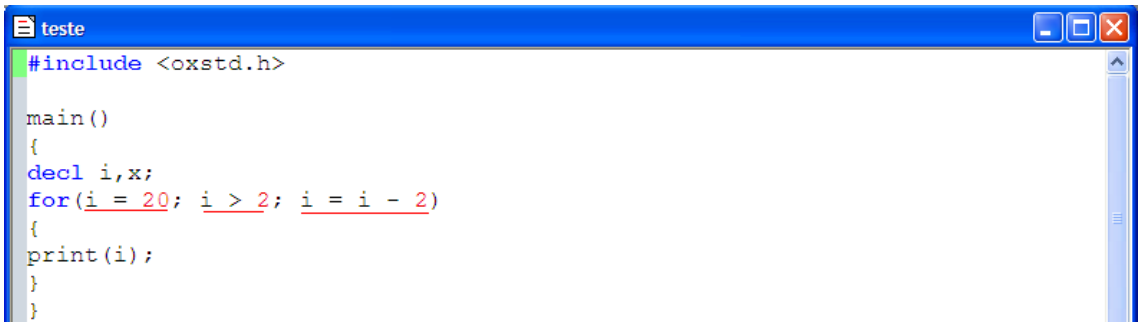
Caso queira-se, por exemplo, escrever os números pares entre 2 e 20 (incluindo-se o 2 e excluindo-se o 20), poderíamos fazer:



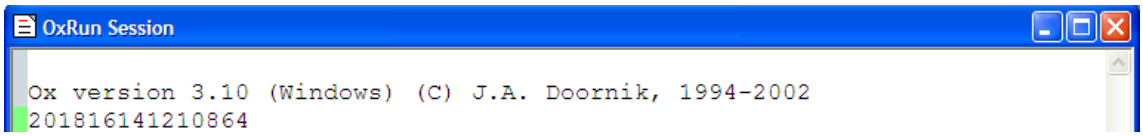
O resultado para o algoritmo acima é:



Ou poderíamos fazer o inverso, escrevermos todos os pares entre 20 (inclusive) até dois (exclusive), o algoritmo seria:



e a saída para o algoritmo acima, como esperado, será:



Outra forma de se criar um comando de repetição no Ox, é usando o comando While. Abaixo, apresentaremos a estrutura do comando while e, será apresentada uma sugestão de solução dos algoritmos apresentados para o comando de repetição utilizando for, mas, agora, na estrutura do while.

while(*condição de parada)*
{
Seqüência de comandos
}

O primeiro exemplo apresentado no comando de repetição (for) foi, resolver o problema de escrever os valores de 0 a 3 no Ox. O algoritmo utilizando a estrutura do while para obtermos a mesma saída será:

```
main()
{
decl i;
i = 0;
while(i < 4)
{
print(i," ");
i = i + 1;
}
```

Observe que é necessário atribuir um valor inicial pra variável antes de se utilizar o while. É importante observar que o incremento da variável i (i = i + 1) é importante para que o algoritmo não caia em um loop infinito, isto é, caso tenha se omitido a linha de incremento, o software iria escrever tela o número zero (valor atribuído à variável i antes do while) até que o algoritmo seja interrompido pelo usuário.

Observe que a saída desse algoritmo será idêntica à saída apresentada para o comando de repetição (for).

Obs. O incremento da variável i, poderia ser dar pelo comando i++

Para resolução do problema de dois índices simultâneos utilizados no for, terá solução semelhante, utilizando o while dado por:

```
main()
{
decl i,j;
i = 1;
j = -1;
while(i < 10 && j <=9)
{
   print(" ", i);
   print(" ");
   print(j," ");
   i = i + 1;
   j = j + 1;
}
}
```

O cálculo utilizando o while para variação de índices com padrões diferentes é bem mais claro. Por exemplo, se quiser escrever os pares entre 2 (inclusive) e 20 (exclusive) podemos utilizar o seguinte algoritmo:

```
main()
{
decl i;
i = 2;
while(i < 20)
     {
print(i);
i = i + 2;
     }
}
```

Ou para fazer o inverso poderemos utilizar o algoritmo:

```
main()
{
decl i;
i = 20;
while(i > 2)
     {
print(i);
i = i - 2;
     }
}
```

Para todos os exemplos acima, a saída será idêntica aos exemplos utilizando o for.

Break e Continue

Ao se utilizar os comandos de repetição, é necessário muito cuidado ao montar-se a estrutura para evitar criar um loop infinito. Em alguns casos, o programador pode simplesmente não saber se, o algoritmo criado irá cair nesse loop ou não.

O comando **break** é utilizado para "forçar" o comando de repetição criado a parar caso certa condição seja satisfeita. Um exemplo de utilização do **break** o caso onde um programador deseja verificar se existe um número negativo em uma matriz muito grande e, se isso acontecesse a primeira vez o algoritmo disparasse uma mensagem de erro.

Algoritmo

repita

Leia o elemento da linha i e coluna j da matriz
Se o elemento for menor que zero, então **pare**

fim repita

fim Algoritmo

Exemplo: Uma atuário deseja verificar se o banco de dados recebido por ele apresenta alguma inconsistência. Ele recebe um vetor com n elementos correspondentes ao benefício futuro de um segurado. Se esse benefício for igual ou menor que zero, é porque o banco apresenta algum problema. Esse atuário deseja verificar então se existe algum valor no seu banco de dados com essa característica.

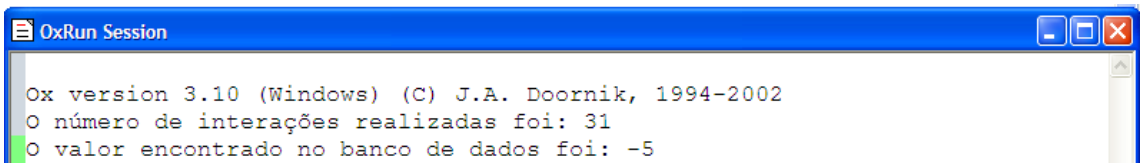
Abaixo, criou-se um banco de dados de nome "*Banco_Dados*" como um vetor de 201 posições e, substitui-se a posição 30 (lembrando que o Ox começa a contar da posição zero, ou seja, existem 30 posições abaixo da posição 30) que continha o valor 31 por -5 e a posição 70 por 0 (zero). Observe que, se o comando de repetição fosse mantido até percorrer todo o vetor, o comando de repetição faria 201 interações, enquanto que, com o comando **break**, foi necessário fazer apenas 31 interações. Abaixo o algoritmo utilizado e sua saída.

```
main()
{
decl n,Banco_Dados,Tamanho_Banco;
n = 0;
Tamanho_Banco = 200;
Banco_Dados = <1:200>;
Banco_Dados[30] = -5;
Banco_Dados[70] = 0;

 while(n < Tamanho_Banco)
 {
   if(Banco_Dados[n] <= 0)
```

```
   {
     print("O número de interações realizadas foi: ");
     print(n + 1);
      print("\n");
      print("O valor encontrado no banco de dados foi: ");
     print(Banco_Dados[n]);
     break
   }
   n = n + 1;
 }

 }
```



Exemplo: Imagine que o mesmo atuário, já ciente do elemento na posição 30, deseja verificar se existe mais um valor no banco que apresenta problemas. Por algum motivo, o atuário não pôde alterar o valor do seu banco de dados na posição 30. Uma dentre várias alternativas para a solução desse problema seria a utilização do **continue.**

```
main()
{
decl n,Banco_Dados,Tamanho_Banco;
n = -1;
Tamanho_Banco = 200;
Banco_Dados = <1:200>;
Banco_Dados[30] = -5;
Banco_Dados[70] = 0;

 while(n < Tamanho_Banco)
   {
   n = n + 1;
   if(Banco_Dados[n] <= 0)
     {

     if(n == 30)
     continue

      print("O número de interações realizadas foi: ");
     print(n + 1);
      print("\n");
      print("O valor encontrado no banco de dados foi: ");
     print(Banco_Dados[n]);
      break
     }
     }
}
```

Observe pelo algoritmo apresentado acima, que ao encontrar o comando **continue** no algoritmo, o programa finaliza a execução do comando de repetição para o índice n e recomeça novamente o comando de execução. Veja que, o algoritmo encontrou o valor negativo quando n assume o valor 30 (nesse caso o banco de dados contém o valor -5). Porém, a saída do código indicando o número de interações e o valor -5 não foi obtida porque antes de executar essas linhas de comando o software executou o comando condicional que perguntava se n == 30. Ao executar o comando **continue**, o software pára automaticamente de executar o comando de repetição e retorna ao início (mantendo-se o índice com o mesmo valor que o do momento de execução do **continue**).

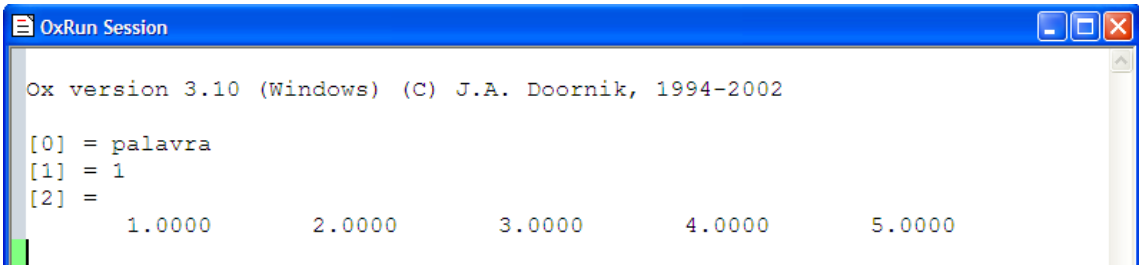
ARRAYS

Arrays são "matrizes" com mais dimensões e mais flexíveis quanto aos elementos possíveis que ela comporta. Matrizes e vetores no Ox não aceitam, por exemplo, elementos que sejam letras (strings). Isso não ocorre no arrays.
Cada elemento de um array pode conter, por exemplo, uma matriz, enquanto o outro contém um único elemento numérico, outro elemento lógico (VERDADEIRO, FALSO) num mesmo array.
Os arrays no Ox são declarados gerados de forma semelhante a matrizes e vetores, porém, o identificador do array é **{}.**

Exemplo: Deseja-se criar um array que contenha em seu primeiro elemento a palavra "palavra", o segundo elemento seja a variável lógica "VERDADEIRO" e, o último elemento do array seja um vetor que contenha os número de 1 a 5. O algoritmo no Ox poderá ser:

```
main()
{
decl palavra,logic,vetor,x;
palavra = "palavra";
logic = TRUE;
vetor = <1:5>;
x = {palavra, logic, vetor};
print(x);
}
```

Cuja saída do algoritmo será:



Para acessar o primeiro elemento desse array, ou seja, para obter a saída o string "palavra" é semelhante a acessar o primeiro elemento de uma matriz, basta utilizar o comando **x[0]**. Para acessar o elemento lógico do array basta utilizar o comando **x[1]**.

A dimensão do array para o último elemento do array é diferente do que os padrões encontrados na matriz. Para acessar, por exemplo, o número dois do vetor que está na segunda posição do array (no Ox as posições são contadas a partir de zero) utiliza-se o comando **x[2][1].**

Observa-se pela saída acima que, o padrão para se acessar um elemento em um array segue o seguinte padrão, o primeiro elemento a ser acessado no array (no exemplo acima de valor 2) corresponde à posição a ser acessada no array, ou seja, o n-ésimo elemento do array. Uma vez acessado esse elemento (um vetor de 5 posições) acessa-se o elemento do vetor, no exemplo acima o valor correspondente é o número 1.

Se ao invés de um vetor, atribuíssemos uma matriz, os elementos poderiam ser acessados da seguinte forma:

array[posição da matriz no array][número da linha da matriz][número da coluna da matriz]

Uma outra forma de se declarar um array é usando o comando **new array**. Com esse comando o algoritmo acima pode ser feito como:

```
main()
{
decl palavra,logic,vetor,x;
palavra = "palavra";
logic = TRUE;
vetor = <1:5>;
x = new array[3];
x[0] = palavra;
x[1] = logic;
x[2] = vetor;
print(x);
}
```

A saída desse algoritmo é a mesma apresentada para o exemplo anterior.

Leitura de Dados no Ox

Leitura de Strings

```
main()
{
decl nome,altura,i,aux_nome, aux_altura;

   scan("%s",&i);
   nome = i;
   print(i);
```

```
}
```

O comando “%s” indica ao Ox que a variável lida será um string.

FUNÇÃO

No Ox, assim como em outros softwares, existem muitas funções já implementadas como zeros, ones, units, log, sqrt entre outras, porém o usuário do Ox pode criar as suas próprias funções.

As funções do Ox podem ser criadas da seguinte maneira (maneira mais simples):

NOMEDAFUNCAO(variáveis de entrada)
{
Declaração de variáveis;
Seq. de comandos;
return(variável de saída);
}

Exemplo: vamos criar uma função que cria um vetor conforme a função **constant** já enunciada neste material mas, neste nosso exemplo a função criará apenas vetores e não matrizes. Chamaremos nossa função de CONSTANTE.

```
CONSTANTE(constante,numero de colunas)
{
decl resultado,i;
resultado = constante;
for( i=0, i<n-1, i-i+1)
     resultado = resultado ~ constante;
return(resultado);
}
```

Observe que as variáveis de entrada não precisam ser declaradas como as demais variáveis da função.

OBSERVAÇÃO: As variáveis utilizadas dentro da função são chamadas **variáveis locais** e, as demais variáveis do algoritmo que estão “fora” das funções são chamadas **variáveis globais.**

Uma variável declarada dentro da função (variável local) não pode ser usada fora dela, ou seja, a variável só existe dentro da função, ou seja, imagine o seguinte algoritmo abaixo:

função*(x)*
{
print*(x);*
}
main*()*

{
***decl** x,y;*
x = 7;
y = 3
***funcao**(y);*
}

No algoritmo acima, apesar de, dentro da função existir a sentença “print(x)”, o “x” de dentro da função (variável local) não é o mesmo x do algoritmo (variável global), então, ao se usar a função o resultado obtido será 3 que é o valor de y.

Algumas funções já implementadas no Ox estão ilustradas abaixo:

Leitura de dados no Ox

load.mat(“nome do arquivo”)

lembrando que a primeira linha do arquivo deve ter o número de linhas e colunas a serem lidas.

GRÁFICOS

A função **DrawTMatrix()** gera o gráfico da série no tempo e a função **DrawXMatrix()** gera o gráfico de uma série contra outra. Uma chamada à função **DrawTMatrix()** tem a seguinte forma `DrawTMatrix(posição, série, “rótulo”, ano inicial, período inicial, periodicidade, tipo e cor de linha);` . Enquanto a função **DrawXMatrix()** tem os seguintes argumentos:

`DrawXMatrix(posição,Y, “rótulo de Y”, X, “rótulo de X”, símbolo, tipo e cor de símbolos);`

Exemplo:

```
main()
{
decl x;
x = rann(1,10);
DrawTMatrix(0,x,"X ~ N(0,1)",1,1,1);
ShowDrawWindow();
}
```