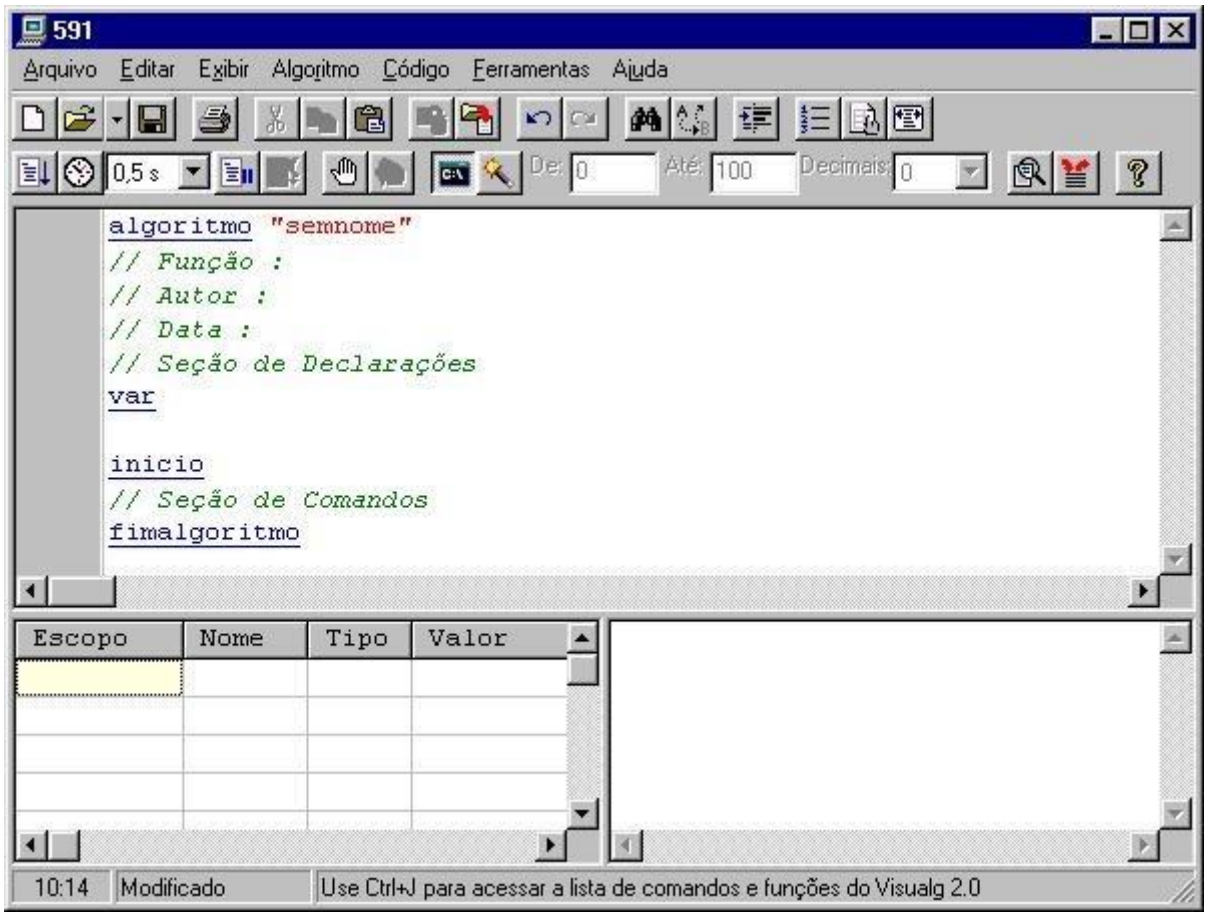
A Linguagem de Programação do VisuAlg

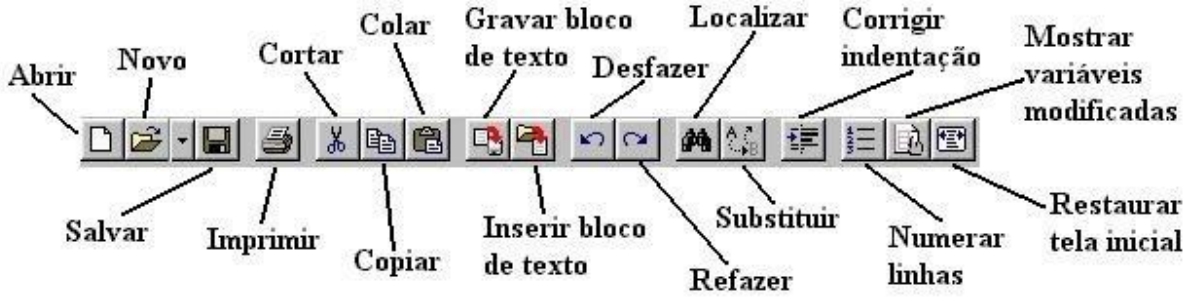
A Tela Principal do VisuAlg

A tela do VisuAlg compõe-se da barra de tarefas, do editor de textos (que toma toda a sua metade superior), do quadro de variáveis (no lado esquerdo da metade inferior), do simulador de saída (no correspondente lado direito) e da barra de *status*. Quando o programa é carregado, já apresenta no editor um "esqueleto" de pseudocódigo, com a intenção de poupar trabalho ao usuário e de mostrar o formato básico que deve ser seguido. Explicaremos a seguir cada componente da interface do VisuAlg.

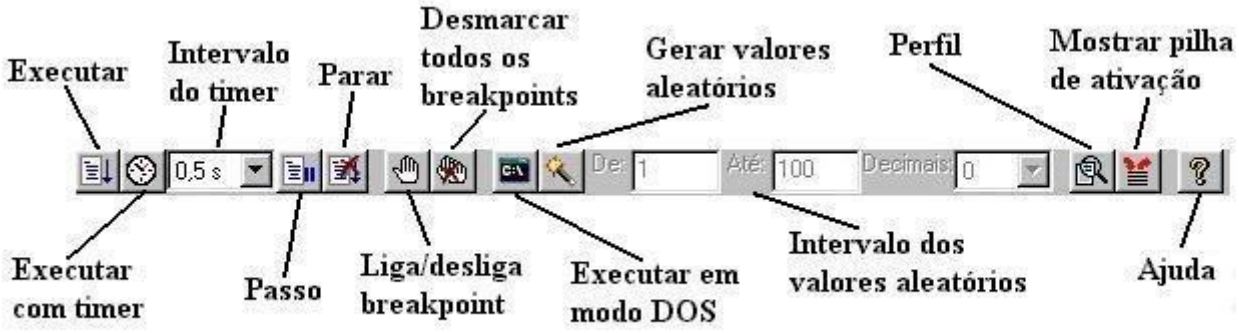


A Barra de Tarefas

Contém os comandos mais utilizados no VisuAlg (estes comandos também podem ser acessados pelo menu ou por atalhos no teclado).



Abrir (Ctrl-A): Abre um arquivo anteriormente gravado, substituindo o texto presente no editor. Se este tiver sido modificado, o VisuAlg pedirá sua confirmação para salvá-lo antes que seja sobreposto.
Novo (Ctrl-N): Cria um novo "esqueleto" de pseudocódigo, substituindo o texto presente no editor. Se este tiver sido modificado, o VisuAlg pedirá sua confirmação para salvá-lo antes que seja sobreposto.
Salvar (Ctrl-S): Grava imediatamente o texto presente no editor. Na primeira vez que um novo texto é gravado, o VisuAlg pede seu nome e localização.
Imprimir: Imprime imediatamente na impressora padrão o texto presente no editor. Para configurar a impressão, use o comando Imprimir do menu Arquivo (acessível também pelo atalho *Ctrl-P*).
Cortar (Ctrl-X): Apaga texto selecionado, armazenando-o em uma área de transferência.
Copiar (Ctrl-C): Copia o texto selecionado para a área de transferência.
Colar (Ctrl-V): Copia texto da área de transferência para o local em que está o cursor.
Gravar bloco de texto: Permite a gravação em arquivo de um texto selecionado no editor. A extensão sugerida para o nome do arquivo é *.inc*.
Inserir bloco de texto: Permite a inserção do conteúdo de um arquivo. A extensão sugerida para o nome do arquivo é *.inc*.
Desfazer (Ctrl-Z): Desfaz último comando efetuado.
Refazer (Shift-Ctrl-Z): Refaz último comando desfeito.
Localizar (Ctrl-L): Localiza no texto presente no editor determinada palavra especificada.
Substituir (Ctrl-U): Localiza no texto presente no editor determinada palavra especificada, substituindo-a por outra.
Corrigir Indentação (Ctrl-G): Corrige automaticamente a *indentação* (ou tabulação) do pseudocódigo, tabulando cada comando interno com espaços à esquerda.
Numerar linhas: Ativa ou desativa a exibição dos números das linhas na área à esquerda do editor. A linha e a coluna do editor em que o cursor está em um determinado momento também são mostradas na barra de *status* (parte inferior da tela). Por motivos técnicos, esta opção é automaticamente desativada durante a execução do pseudocódigo, mas volta a ser ativada logo em seguida.
Mostrar variáveis modificadas: Ativa ou desativa a exibição da variável que está sendo modificada. Como o número de variáveis pode ser grande, muitas podem estar fora da janela de visualização; quando esta característica está ativada, o VisuAlg rola a grade de exibição de modo que cada variável fique visível no momento em está sendo modificada. Este recurso é especialmente útil quando se executa um pseudocódigo passo a passo. Por questões de desempenho, a configuração padrão desta característica é *desativada,* quando o pseudocódigo está sendo executado automaticamente. No entanto, basta clicar este botão para executá-lo automaticamente com a exibição ativada. No final da execução, a configuração volta a ser *desativada*.
Restaurar tela inicial: Retorna a divisão da tela ao formato inicial, caso você tenha mudado o tamanho da área do editor de texto, quadro de variáveis ou simulador de saída.



Executar (F9): Inicia (ou continua) a execução automática do pseudocódigo.
Executar com *timer* (Shift-F9): Insere um atraso (que pode ser especificado no intervalo ao lado) antes da execução de cada linha. Também realça em fundo azul o comando que está sendo executado, da mesma forma que na execução passo a passo.

Intervalo do *timer*: Atraso em cada linha, para quando se deseja executar o pseudocódigo com *timer*.
Passo (F8): Inicia (ou continua) a execução linha por linha do pseudocódigo, dando ao usuário a oportunidade de acompanhar o fluxo de execução, os valores das variáveis e a pilha de ativação dos subprogramas.
Parar (Ctrl-F2): Termina imediatamente a execução do pseudocódigo. Evidentemente, este botão fica desabilitado quando o pseudocódigo não está sendo executado.
Liga/desliga *breakpoint* (F5): Insere/remove um ponto de parada na linha em que esteja o cursor. Estes pontos de parada são úteis para a depuração e acompanhamento da execução dos pseudocódigos, pois permitem a verificação dos valores das variáveis e da pilha de ativação de subprogramas.
Desmarcar todos os *breakpoints* (Ctrl-F5): Desativa todos os *breakpoints* que estejam ativados naquele momento.
Executar em modo DOS: Com esta opção ativada, tanto a entrada como a saída-padrão passa a ser uma janela que imita o DOS, simulando a execução de um programa neste ambiente.
Gerar valores aleatórios: Ativa a geração de valores aleatórios que substituem a digitação de dados. A faixa padrão de valores gerados é de 0 a 100 inclusive, mas pode ser modificada (basta alterar intervalo ao lado). Para a geração de dados do tipo caractere, não há uma faixa preestabelecida: os dados gerados serão sempre *strings* de 5 letras maiúsculas.
Intervalo dos valores aleatórios: Faixa de valores que serão gerados automaticamente, quando esta opção estiver ativada.
Perfil (F7): Após a execução de um pseudocódigo, exibe o número de vezes que cada uma das suas linhas foi executada. É útil para a análise de eficiência (por exemplo, nos métodos de ordenação).
Mostrar pilha de ativação (Ctrl-F3): Exibe a pilha de subprogramas ativados num dado momento. Convém utilizar este comando em conjunto com *breakpoints* ou com a execução passo a passo.
Ajuda (F1): Possibilita acesso às páginas de ajuda e às informações sobre o VisuAlg.

Quadro de Variáveis

É formado por uma grade na qual são mostrados o escopo de cada variável (se for do programa principal, será global; se for local, será apresentado o nome do subprograma onde foi declarada), seus nomes (também com os índices, nos casos em que sejam vetores), seu tipo ("I" para inteiro, "R" para real, "C" para caractere e "L" para lógico) e o seu valor corrente. A versão atual do VisuAlg permite a visualização de até 500 variáveis (contando individualmente cada elemento dos vetores).

A Barra de *Status*

Situada na parte inferior da tela, esta barra contém dois painéis: o primeiro mostra a linha e a coluna onde o cursor está, e o segundo mostra a palavra *Modificado* no caso em que o pseudocódigo tenha sido alterado desde que foi carregado ou salvo pela última vez. Nesta barra, há ainda um terceiro painel disponível, que ainda não tem um uso específico na atual versão.

Introdução

A linguagem que o VisuAlg interpreta é bem simples: é uma versão portuguesa dos pseudocódigos largamente utilizados nos livros de introdução à programação, conhecida como "Portugol". Há ainda alguns comandos novos, com o intuito de criar facilidades específicas para o ensino de técnicas de elaboração de algoritmos.

A linguagem do VisuAlg permite apenas **um comando por linha**.

Importante: para facilitar a digitação e evitar confusões, todas as palavras-chave do VisuAlg foram implementadas sem acentos, cedilha, etc. Portanto, o tipo de dados lógico é definido como logico, o

comando se..então..senão é definido como se..entao..senao, e assim por diante. O VisuAlg também não distingue maiúsculas e minúsculas no reconhecimento de palavras-chave e nomes de variáveis.

Formato Básico do Pseudocódigo e Inclusão de Comentários

O formato básico do nosso pseudocódigo é o seguinte:

```
algoritmo "semnome"
// Função :
// Autor :
// Data :
// Seção de Declarações

inicio
// Seção de Comandos

fimalgoritmo
```

A primeira linha é composta pela palavra-chave algoritmo seguida do seu nome delimitado por aspas duplas. Este nome será usado como título nas janelas de leitura de dados. A seção que se segue é a de declaração de variáveis, que termina com a linha que contém a palavra-chave `inicio`. Deste ponto em diante está a seção de comandos, que continua até a linha em que se encontre a palavra-chave `fimalgoritmo`. Esta última linha marca o final do pseudocódigo: todo texto existente a partir dela é ignorado pelo interpretador.

O VisuAlg permite a inclusão de comentários: qualquer texto precedido de "//" é ignorado, até se atingir o final da sua linha. Por este motivo, os comentários não se estendem por mais de uma linha: quando se deseja escrever comentários mais longos, que ocupem várias linhas, cada uma delas deverá começar por "//".

Tipos de Dados

O VisuAlg prevê quatro tipos de dados: **inteiro**, **real**, **cadeia de caracteres** e **lógico** (ou *booleano*). As palavras-chave que os definem são as seguintes (observe que elas não tem acentuação):

- `inteiro`: define variáveis numéricas do tipo inteiro, ou seja, sem casas decimais.
- `real`: define variáveis numéricas do tipo real, ou seja, com casas decimais.
- `caractere`: define variáveis do tipo *string*, ou seja, cadeia de caracteres.
- `logico`: define variáveis do tipo *booleano*, ou seja, com valor VERDADEIRO ou FALSO.

Nomes de Variáveis e sua Declaração

Os nomes das variáveis devem começar por uma letra e depois conter letras, números ou *underline*, até um limite de 30 caracteres. Não pode haver duas variáveis com o mesmo nome.

A seção de declaração de variáveis começa com a palavra-chave var, e continua com as seguintes sintaxes:

```
<lista-de-variáveis> : <tipo-de-dado>
```

Na <lista-de-variáveis>, os nomes das variáveis estão separados por vírgulas.

Exemplos:

```
var
     a: inteiro
     Valor1, Valor2: real
     nome_do_aluno: caractere
     sinalizador: logico
```

Note que não há a necessidade de ponto e vírgula após cada declaração: basta pular linha. O número total de variáveis suportado pelo VisuAlg é 500 (cada elemento de um vetor é contado individualmente).

Constantes e Comando de Atribuição

O VisuAlg tem três tipos de constantes:

- *Numéricos:* são valores numéricos escritos na forma usual das linguagens de programação. Podem ser inteiros ou reais. Neste último caso, o separador de decimais é o ponto e não a vírgula, independente da configuração regional do computador onde o VisuAlg está sendo executado. O VisuAlg também não suporta separadores de milhares.
- *Caracteres*: qualquer cadeia de caracteres delimitada por aspas duplas (").
- *Lógicos*: admite os valores VERDADEIRO ou FALSO.

A atribuição de valores a variáveis é feita com o operador **<-**. Do seu lado esquerdo fica a variável à qual está sendo atribuído o valor, e à sua direita pode-se colocar qualquer expressão (constantes, variáveis, expressões numéricas), desde que seu resultado tenha tipo igual ao da variável.

Alguns exemplos de atribuições, usando as variáveis declaradas acima:

```
a <- 3
Valor1 <- 1.5
Valor2 <- Valor1 + a
nome_do_aluno <- "José da Silva"
sinalizador <- FALSO
```

Operadores Aritméticos

`+,-`	Operadores unários, isto é, são aplicados a um único operando. São os operadores aritméticos de maior precedência. Exemplos: `-3, +x`. Enquanto o operador unário `-` inverte o sinal do seu operando, o operador `+` não altera o valor em nada o seu valor.
`\`	Operador de divisão inteira. Por exemplo, `5 \ 2 = 2`. Tem a mesma precedência do operador de divisão tradicional.
`+,-,*, /`	Operadores aritméticos tradicionais de adição, subtração, multiplicação e divisão. Por convenção, `*` e `/` têm precedência sobre `+` e `-`. Para modificar a ordem de avaliação das operações, é necessário usar parênteses como em qualquer expressão aritmética.
`MOD ou %`	Operador de módulo (isto é, resto da divisão inteira). Por exemplo, `8 MOD 3 = 2`. Tem a mesma precedência do operador de divisão tradicional.
`^`	Operador de potenciação. Por exemplo, `5 ^ 2 = 25`. Tem a maior precedência entre os operadores aritméticos binários (aqueles que têm dois operandos).

Operadores de Caracteres

+	Operador de concatenação de *strings* (isto é, cadeias de caracteres), quando usado com dois valores (variáveis ou constantes) do tipo "caractere". Por exemplo: "Rio " + " de Janeiro" = "Rio de Janeiro".

Operadores Relacionais

=, <, >, <=, >=, <>	Respectivamente: igual, menor que, maior que, menor ou igual a, maior ou igual a, diferente de. São utilizados em expressões lógicas para se testar a relação entre dois valores do mesmo tipo. Exemplos: 3 = 3 (3 é igual a 3?) resulta em VERDADEIRO ; "A" > "B" ("A" está depois de "B" na ordem alfabética?) resulta em FALSO.

Importante: No VisuAlg, as comparações entre *strings* **não diferenciam** as letras maiúsculas das minúsculas. Assim, "ABC" é igual a "abc". Valores lógicos obedecem à seguinte ordem: FALSO < VERDADEIRO.

Operadores Lógicos

nao	Operador unário de negação. nao VERDADEIRO = FALSO, e nao FALSO = VERDADEIRO. Tem a maior precedência entre os operadores lógicos.
ou	Operador que resulta VERDADEIRO quando um dos seus operandos lógicos for verdadeiro.
e	Operador que resulta VERDADEIRO somente se seus dois operandos lógicos forem verdadeiros.
xou	Operador que resulta VERDADEIRO se seus dois operandos lógicos forem diferentes, e FALSO se forem iguais.

Comandos de Saída de Dados

escreva (<*lista-de-expressões*>)

Escreve no dispositivo de saída padrão (isto é, na área à direita da metade inferior da tela do VisuAlg) o conteúdo de cada uma das expressões que compõem <*lista-de-expressões*>. As expressões dentro desta lista devem estar separadas por vírgulas; depois de serem avaliadas, seus resultados são impressos na ordem indicada.

É possível especificar o número de espaços no qual se deseja escrever um determinado valor. Por exemplo, o comando escreva(x:5) escreve o valor da variável x em 5 espaços, alinhando-o à direita. Para variáveis reais, pode-se também especificar o número de casas fracionárias que serão exibidas. Por exemplo, considerando y como uma variável real, o comando escreva(y:6:2) escreve seu valor em 6 espaços colocando 2 casas decimais.

escreval (<*lista-de-expressões*>).

Idem ao anterior, com a única diferença que pula uma linha em seguida.

Exemplos:

```
algoritmo "exemplo"

var
  x: real
  y: inteiro
  a: caractere
  l: logico
```

```
inicio
  x <- 2.5
  y <- 6
  a <- "teste"
  l <- VERDADEIRO
  escreval ("x", x:4:1, y+3:4) // Escreve: x 2.5    9
  escreval (a, "ok")           // Escreve: testeok (e depois pula
  linha)
  escreval (a, " ok")          // Escreve: teste ok (e depois pula
  linha)
  escreval (a + " ok")         // Escreve: teste ok (e depois pula
  linha)
  escreva (l)                  // Escreve: VERDADEIRO
 fimalgoritmo
```

Note que o VisuAlg separa expressões do tipo numérico e lógico com um espaço à esquerda, mas não as expressões do tipo caractere, para que assim possa haver a concatenação. Quando se deseja separar expressões do tipo caractere, é necessário acrescentar espaços nos locais adequados.

Comando de Entrada de Dados

```
leia (<lista-de-variáveis>)
```

Recebe valores digitados pelos usuário, atribuindo-os às variáveis cujos nomes estão em *<lista-de-variáveis>* (é respeitada a ordem especificada nesta lista).

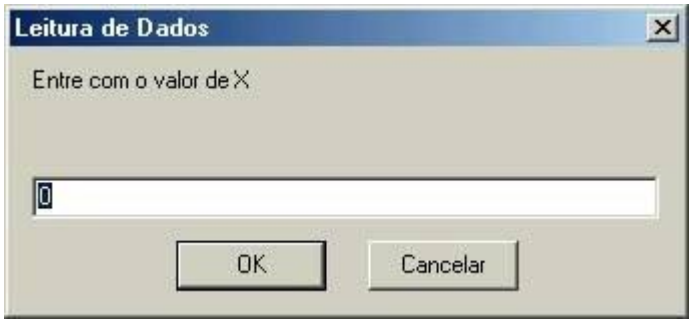
Veja no exemplo abaixo o resultado:

```
algoritmo "exemplo 1"
  var
       x: inteiro;
  inicio
      leia (x)
      escreva (x)
  fimalgoritmo
```

O comando de leitura acima irá exibir uma janela como a que se vê ao lado, com a mensagem padrão:

"Entre com o valor de *<nome-de-variável>*"

Se você clicar em *Cancelar* ou teclar *Esc* durante a leitura de dados, o programa será imediatamente interrompido.



Comando de Desvio Condicional

Condicional Simples

```
se <expressão-lógica> entao
   <sequência-de-comandos>
fimse
```

Ao encontrar este comando, o VisuAlg analisa a *<expressão-lógica>*. Se o seu resultado for VERDADEIRO, todos os comandos da *<sequência-de-comandos>* (entre esta linha e a linha com fimse) são executados. Se o resultado for FALSO, estes comandos são desprezados e a execução do algoritmo continua a partir da primeira linha depois do fimse.

Condicional Composto

```
se <expressão-lógica> entao
     <sequência-de-comandos-1>
senao
     <sequência-de-comandos-2>
fimse
```

Nesta outra forma do comando, se o resultado da avaliação de *<expressão-lógica>* for VERDADEIRO, todos os comandos da *<sequência-de-comandos-1>* (entre esta linha e a linha com senao) são executados, e a execução continua depois a partir da primeira linha depois do fimse. Se o resultado for FALSO, estes comandos são desprezados e o algoritmo continua a ser executado a partir da primeira linha depois do senao, executando todos os comandos da *<sequência-de-comandos-2>* (até a linha com fimse).

O VisuAlg permite o aninhamento desses comandos de desvio condicional.

Comando de Seleção Múltipla

A sintaxe é a seguinte:

```
escolha <expressão-de-seleção>
     caso <exp11>, <exp12>, ..., <exp1n>
          <sequência-de-comandos-1>
     caso <exp21>, <exp22>, ..., <exp2n>
          <sequência-de-comandos-2>
     ...
     outrocaso
          <sequência-de-comandos-extra>

fimescolha
```

Veja o exemplo a seguir, que ilustra bem o que faz este comando:

```
algoritmo "Times"
   var
        time: caractere
   inicio
        escreva ("Entre com o nome de um time de futebol: ")
        leia (time)
        escolha time
             caso "Flamengo", "Fluminense", "Vasco", "Botafogo"
                  escreval ("É um time carioca.")
             caso "São Paulo", "Palmeiras", "Santos", "Corinthians"
                  escreval ("É um time paulista.")
             outrocaso
                  escreval ("É de outro Estado.")
        fimescolha
   fimalgoritmo
```

Comandos de Repetição

O VisuAlg implementa as três estruturas de repetição usuais nas linguagens de programação: o laço contado `para...ate...faca` e os laços condicionados `enquanto...faca` e `repita...ate`. A sintaxe destes comandos é explicada a seguir.

Para ... faça

Esta estrutura repete uma sequência de comandos um determinado número de vezes.

```
para <variável> de <valor-inicial> ate <valor-limite> [passo
<incremento/decremento>] faca
     <sequência-de-comandos>
fimpara
```

<variável >	É a variável contadora que controla o número de repetições do laço. Na versão atual, deve ser necessariamente uma variável do tipo `inteiro`, como todas as expressões deste comando.
<valor-inicial>	É uma expressão que especifica o valor de inicialização da variável contadora antes da primeira repetição do laço.
<valor-limite >	É uma expressão que especifica o valor máximo que a variável contadora pode alcançar.
<incremento/decremento>	É opcional. Quando presente, precedida pela palavra `passo`, é uma expressão que especifica o incremento que será acrescentado à variável contadora em cada repetição do laço. Quando esta opção não é utilizada, o valor padrão de <*incremento>* é 1. Vale a pena ter em conta que também é possível especificar valores negativos *<decremento>*.
`fimpara`	Indica o fim da sequência de comandos a serem repetidos. Cada vez que o programa chega neste ponto, é acrescentado à variável contadora o valor de *<incremento>*, e comparado a *<valor-limite>*. Se for menor ou igual (ou maior ou igual, quando *<decremento>* negativo), a sequência de comandos será executada mais uma vez; caso contrário, a execução prosseguirá a partir do primeiro comando que esteja após o `fimpara`.

<valor-inicial>, <valor-limite> e *<incremento/decremento>* são avaliados uma **única vez** antes da execução da primeira repetição, e **não se alteram durante a execução do laço**, mesmo que variáveis eventualmente presentes nessas expressões tenham seus valores alterados.

No exemplo a seguir, os números de 1 a 10 são exibidos em ordem crescente.

```
algoritmo "Números de 1 a 10"
var
     j: inteiro
inicio
    para j de 1 ate 10 faca
       escreva (j:3)
    fimpara
fimalgoritmo
```

Importante: Se, logo no início da primeira repetição, *<valor-inicial>* for maior que *<valor-limite>* (ou menor, quando for negativo *<decremento>*), o laço não será executado nenhuma vez. O exemplo a seguir não imprime nada.

```
algoritmo "Numeros de 10 a 1 (não funciona)"
 var
      j: inteiro
 inicio
      para j de 10 ate 1 faca
         escreva (j:3)
      fimpara
 fimalgoritmo
```

Este outro exemplo, no entanto, funcionará por causa do **passo -1**:

```
algoritmo "Numeros de 10 a 1 (este funciona)"
 var
      j: inteiro
 inicio
      para j de 10 ate 1 passo -1 faca
          escreva (j:3)
      fimpara
 fimalgoritmo
```

Enquanto ... faça

Esta estrutura repete uma sequência de comandos enquanto uma determinada condição (especificada através de uma expressão lógica) for satisfeita.

```
enquanto <expressão-lógica> faca
      <sequência-de-comandos>
fimenquanto
```

<expressão-lógica>	Esta expressão que é avaliada antes de cada repetição do laço. Quando seu resultado for VERDADEIRO, *<sequência-de-comandos>* é executada.
`fimenquanto`	Indica o fim da *<sequência-de-comandos>* que será repetida. Cada vez que a execução atinge este ponto, volta-se ao início do laço para que *<expressãológica>* seja avaliada novamente. Se o resultado desta avaliação for VERDADEIRO, a *<sequência-de-comandos>* será executada mais uma vez; caso contrário, a execução prosseguirá a partir do primeiro comando após `fimenquanto`.

O mesmo exemplo anterior pode ser resolvido com esta estrutura de repetição:

```
algoritmo "Números de 1 a 10 (com enquanto...faca)"
var
      j: inteiro
inicio
      j <- 1
      enquanto j <= 10 faca
           escreva (j:3)
           j <- j + 1
      fimenquanto
fimalgoritmo
```

Importante: Como o laço `enquanto...faca` testa sua condição de parada **antes** de executar sua sequência de comandos, esta sequência poderá ser executada **zero ou mais vezes**.

Repita ... até

Esta estrutura repete uma sequência de comandos até que uma determinada condição (especificada através de uma expressão lógica) seja satisfeita.

```
repita
     <sequência-de-comandos>
ate <expressão-lógica>
```

`repita`	Indica o início do laço.
`ate` *<expressão-lógica>*	Indica o fim da *<sequência-de-comandos>* a serem repetidos. Cada vez que o programa chega neste ponto, *<expressão-lógica>* é avaliada: se seu resultado for FALSO, os comandos presentes entre esta linha e a linha `repita` são executados; caso contrário, a execução prosseguirá a partir do primeiro comando após esta linha.

Considerando ainda o mesmo exemplo:

```
algoritmo "Números de 1 a 10 (com repita)"
var
     j: inteiro
inicio
     j <- 1
     repita
          escreva (j:3)
          j <- j + 1
     ate j > 10
fimalgoritmo
```

Importante: Como o laço `repita...ate` testa sua condição de parada **depois** de executar sua sequência de comandos, esta sequência poderá ser executada **uma ou mais vezes**.

Variáveis Estruturadas Homogêneas

Variáveis Indexadas Unidimensionais (Vetores)

Variáveis indexadas com uma única dimensão, também conhecidas como vetores, são referenciadas por um único índice. A sintaxe para declaração é:

```
<identificador> : vetor [<tamanho>] de < tipo >

Tamanho [VI..VF]=> VI = Valor inicial do índice e VF valor Final do índice.
```

Exemplos:

```
IDADE: VETOR [1..5] DE INTEIRO
NOMES: VETOR [1..5] DE CARACTERE
```

A declaração acima corresponde a cinco nomes: nomes[1], nomes[2], nomes[3], nomes[4], nomes[5] e cinco idades: idades[1], idades[2], idades[3], idades[4], idades[5], onde serão armazenados os nomes e idades das 5 pessoas lidas.

Para se atribuir um valor a um elemento do vetor devemos utilizar o seguinte padrão:

```
< identificador>[<posição>] <- <valor>
```

Exemplos:

```
nomes[1] <- "João da Silva"
idades[1] <- 35
nomes[3] <- "Maria Aparecida"
idades[3] <- idades[1]
i <- 5
idades[i] <- 45
```

Variáveis Indexadas Bidimensionais (Matrizes)

Variáveis indexadas com duas dimensões, também conhecida como matrizes, são referenciadas por dois índices, cada qual começando por 1. A sintaxe para declaração é:

```
<identificador> : vetor [<tamanho1>,<tamanho2>] de < tipo >

Tamanho [VI..VF]=> Vi= Valor inicial do índice e VF valor Final do índice.
```

Exemplo:

```
PESSOAS: VETOR [1..2,1..3] DE CARACTERE
```

A declaração acima corresponde às 6 posições: PESSOAS[1,1], PESSOAS[1,2], PESSOAS[1,3], PESSOAS[2,1], PESSOAS[2,2] e PESSOAS[2,3], onde serão armazenados os nomes das pessoas.

Para se atribuir um valor a um elemento do vetor devemos utilizar o seguinte padrão:

```
< identificador>[<posição 1>,<posição 2>] <- <valor>
```

Exemplo:

```
PESSOAS[1,3]<- "Adriana"
```