Resumo VISUALG

O VisuAlg é um programa simples, esta ferramenta permite aos alunos iniciantes em programação o exercício dos seus conhecimentos num ambiente próximo da realidade ***http://eletrica.ufpr.br/~rogerio/visualg/Help/linguagem.htm***

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 lógico, 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:

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

inicio
// Seção de Comandos
fimalgoritmo
```

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 têm 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.***

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

<lista-de-variáveis>:<tipo-de-dado>
<lista-de-variáveis>: vetor "["<lista-de-intervalos>"]" de <tipo-de-dado>

Nota: Na <lista-de-variáveis>, os nomes das variáveis estão separados por vírgulas. Na <lista-de-intervalos>, os <intervalo> são separados por vírgulas, e têm a seguinte sintaxe:
<intervalo>:<valor-inicial>..<valor-final>

Operadores Aritméticos: é o conjunto de símbolos que representa as operações básicas da matemática.

+,-	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.
\ *ou div*	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 relacionais: é o conjunto de símbolos utilizados para comparar valores, o resultado de uma expressão relacional é um valor booleano (VERDADEIRO ou FALSO).

=, <, >, <=, >=, <>	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: é o conjunto de símbolos utilizados para operações básicas em portas lógicas (e, ou, nao, xou), o resultado da operação é um valor booleano (VERDADEIRO ou FALSO).

nao	Operador unário de negação. nao VERDADEIRO = FALSO, e nao FALSO = VERDADEIRO. Tem a maior precedência entre os operadores lógicos. Equivale ao NOT do Pascal.
ou	Operador que resulta VERDADEIRO quando um dos seus operandos lógicos for verdadeiro. Equivale ao OR do Pascal.
e	Operador que resulta VERDADEIRO somente se seus dois operandos lógicos forem verdadeiros. Equivale ao AND do Pascal.
xou	Operador que resulta VERDADEIRO se seus dois operandos lógicos forem diferentes, e FALSO se forem iguais. Equivale ao XOR do Pascal.

Operador de caracter ou concatenação: utilizado para juntar duas cadeias 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".

Comandos de Saída de Dados

escreva (<lista-de-expressões>)

Nota: 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.

escreval (<lista-de-expressões>)

OBS.: Idem ao anterior, com a única diferença que pula uma linha em seguida.
Para formatar numeração utilize: <var >: <quantidade de números antes da virgula> : <quantidade de números depois da virgura>

Comando de Entrada de Dados

leia (<lista-de-variáveis>)

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

Comando de Desvio Condicional

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

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

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

Note que não há necessidade de delimitadores de bloco como em algumas linguagens, pois as sequências de comandos já estão delimitadas pelas palavras-chave senao e fimse. O VisuAlg permite o aninhamento desses comandos de desvio condicional.

Comando de Seleção Múltipla

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

Comandos de Repetição

Para ... faça

para <variável> de <valor-inicial> ate <valor-limite> [passo <incremento>] faca
<seqüência-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 >	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 para ***<incremento>***. Por outro lado, se a avaliação da expressão ***<incremento>*** resultar em valor nulo, a execução do algoritmo será interrompida, com a impressão de uma mensagem de erro.
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 ***<incremento>*** for 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***.

Enquanto ... faça

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 ***<expressão-lógica>*** 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***.

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

repita
<seqüência-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.

Importante: Como o laço repita...ate testa sua condição de parada depois de executar sua sequência de comandos, esta se-

quência poderá ser executada uma ou mais vezes.

Comando Interrompa

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

As três estruturas de repetição acima permitem o uso do comando interrompa, que causa uma saída imediata do laço.

Procedimentos e Funções

Subprograma é um programa que auxilia o programa principal através da realização de uma determinada ***subtarefa***. Também costuma receber os nomes de ***sub-rotina***, ***procedimento***, ***método*** ou ***módulo***. Os subprogramas são chamados dentro do corpo do programa principal como se fossem *comandos*. Após seu término, a execução continua a partir do ponto onde foi chamado. É importante compreender que a chamada de um subprograma simplesmente gera um **desvio provisório no fluxo de execução**.
Há um caso particular de subprograma que recebe o nome de ***função***. Uma ***função***, além de executar uma determinada tarefa, ***retorna um valor para quem a chamou***, que é o resultado da sua execução. Por este motivo, a chamada de uma função aparece no corpo do programa principal como uma ***expressão***, e não como um comando.

Cada ***subprograma***, além de ter acesso às variáveis do programa que o chamou (são as ***variáveis globais***), pode ter suas próprias variáveis (são as ***variáveis locais***), que existem apenas durante sua chamada.

Ao se chamar um subprograma, também é possível passar-lhe determinadas informações que recebem o nome de ***parâmetros*** (são valores que, na linha de chamada, ficam entre os parênteses e que estão separados por vírgulas). A quantidade dos parâmetros, sua sequência e respectivos tipos não podem mudar: devem estar de acordo com o que foi especificado na sua correspondente declaração.

Para se criar subprogramas, é preciso descrevê-los após a declaração das variáveis e antes do corpo do programa principal. O VisuAlg possibilita declaração e chamada de subprogramas nos moldes da linguagem Pascal, ou seja, procedimentos e funções com passagem de parâmetros por valor ou referência. Isso será explicado a seguir.

Procedimentos

Em VisuAlg, procedimento é um subprograma que não retorna nenhum valor (corresponde ao ***procedure*** do Pascal). Sua declaração, que deve estar entre o final da declaração de variáveis e a linha inicio do programa principal, segue a sintaxe abaixo:

```
procedimento <nome-de-procedimento>[(<sequência-de-declarações-de-parâmetros>)]
// Seção de Declarações Internas
inicio
// Seção de Comandos
fimprocedimento
```

O ***<nome-de-procedimento>*** obedece as mesmas regras de nomenclatura das variáveis. Por outro lado, a ***<sequência-de-declarações-de-parâmetros>*** é uma sequência de

```
[var] <sequência-de-parâmetros>: <tipo-de-dado>
```

separadas por ponto e vírgula. A presença (opcional) da palavra-chave var indica passagem de parâmetros por referência; caso contrário, a passagem será por valor.

Por sua vez, ***<sequência-de-parâmetros>*** é uma sequência de nomes de parâmetros (também obedecem a mesma regra de nomenclatura de variáveis) separados por vírgulas.

De modo análogo ao programa principal, a seção de declaração interna começa com a palavra-chave var, e continua com a seguinte sintaxe:

<lista-de-variáveis> : <tipo-de-dado>

Funções

Em VisuAlg, função é um subprograma que retorna um valor (corresponde ao ***function*** do Pascal). De modo análogo aos procedimentos, sua declaração deve estar entre o final da declaração de variáveis e a linha inicio do programa principal, e segue a sintaxe abaixo:

```
funcao <nome-de-função>(<sequência-de-declarações-de-parâmetros>)]: <tipo-de-dado>
// Seção de Declarações Internas
inicio
// Seção de Comandos
fimfuncao
```

O **<nome-de-função>** obedece as mesmas regras de nomenclatura das variáveis. Por outro lado, a ***<sequência-de-declarações-de-parâmetros>*** é uma sequência de

[var] <sequência-de-parâmetros>: <tipo-de-dado>

separadas por ponto e vírgula. A presença (opcional) da palavra-chave var indica passagem de parâmetros por referência; caso contrário, a passagem será por valor.

Por sua vez, ***<sequência-de-parâmetros>*** é uma sequência de nomes de parâmetros (também obedecem a mesma regra de nomenclatura de variáveis) separados por vírgulas.

O valor retornado pela função será do tipo especificado na sua declaração (logo após os dois pontos). Em alguma parte da função (de modo geral, no seu final), este valor deve ser retornado através do comando retorne.

De modo análogo ao programa principal, a seção de declaração interna começa com a palavra-chave var, e continua com a seguinte sintaxe:

<lista-de-variáveis>:<tipo-de-dado>

Voltando ao exemplo anterior, no qual calculamos e imprimimos a soma entre os valores 4 e –9, vamos mostrar como isso poderia ser feito através de uma ***função sem parâmetros***. Ela também utiliza uma variável local aux para armazenar provisoriamente o resultado deste cálculo, antes de atribuí-lo à variável global ***res***:

```
funcao soma: inteiro
var aux: inteiro
inicio
// n, m e res são variáveis globais
aux <- n + m
retorne aux
fimfuncao
```

No programa principal deve haver os seguintes comandos:

```
n <- 4
m <- -9
res <- soma
escreva(res)
```

Se realizássemos essa mesma tarefa com uma ***função com parâmetros passados por valor***, poderia ser do seguinte modo:

```
funcao soma (x,y: inteiro): inteiro
inicio
retorne x + y
fimfuncao
```

No programa principal deve haver os seguintes comandos:

```
n <- 4
m <- -9
res <- soma(n,m)
escreva(res)
```

Passagem de Parâmetros por Referência

Há ainda uma outra forma de passagem de parâmetros para subprogramas: é a passagem por referência. Neste caso, o subprograma não recebe apenas um valor, mas sim o ***endereço*** de uma variável global. Portanto, qualquer modificação que for realizada no conteúdo deste parâmetro afetará também a variável global que está associada a ele. Durante a execução do subprograma, os parâmetros passados por referência são análogos às variáveis globais. No VisuAlg, de forma análoga a Pascal, essa passagem é feita através da palavra ***var*** na declaração do parâmetro.

Voltando ao exemplo da soma, o procedimento abaixo realiza a mesma tarefa utilizando passagem de parâmetros por referência:

```
procedimento soma (x,y: inteiro; var result: inteiro)
inicio
result <- x + y
fimprocedimento
```

No programa principal deve haver os seguintes comandos:

```
n <- 4
m <- -9
soma(n,m,res)
escreva(res)
```

Recursão e Aninhamento

A atual versão do VisuAlg permite recursão, isto é, a possibilidade de que um subprograma possa chamar a si mesmo. A função do exemplo abaixo calcula recursivamente o fatorial do número inteiro que recebe como parâmetro:

```
funcao fatorial (v: inteiro): inteiro
inicio
se v <= 2 entao
retorne v
senao
retorne v * fatorial(v-1)
fimse
fimfuncao
```

Em Pascal, é permitido o aninhamento de subprogramas, isto é, cada subprograma também pode ter seus próprios subprogramas. No entanto, esta característica dificulta a elaboração dos compiladores e, na prática, não é muito importante. Por este motivo, ela não é permitida na maioria das linguagens de programação (como C, por exemplo), e o VisuAlg não a implementa.

Comando Timer

Embora o VisuAlg seja um interpretador de pseudocódigo, seu desempenho é muito bom: o tempo gasto para interpretar cada linha digitada é apenas uma fração de segundo. Entretanto, por motivos educacionais, pode ser conveniente exibir o fluxo de execução do pseudocódigo comando por comando, em "câmera lenta". O comando timer serve para este propósito: insere um atraso (que pode ser especificado) antes da exe-

cução de cada linha. Além disso, realça em fundo azul o comando que está sendo executado, da mesma forma que na execução passo a passo.

Sua sintaxe é a seguinte:

timer on	Ativa o timer.
timer <tempo-de-atraso>	Ativa o timer estabelecendo seu tempo de atraso em milissegundos. O valor padrão é 500, que equivale a meio segundo. O argumento deve ser uma constante inteira com valor entre 0 e 10000. Valores menores que 0 são corrigidos para 0, e maiores que 10000 para 10000.
timer off	Desativa o timer

Nota: Ao longo do pseudocódigo, pode haver vários comandos timer. Todos eles devem estar na seção de comandos. Uma vez ativado, o atraso na execução dos comandos será mantido até se chegar ao final do pseudocódigo ou até ser encontrado um comando timer off.

Comando Aleatório

aleatorio [on]	Ativa a geração de valores aleatórios que substituem a digitação de dados. A palavra-chave on é opcional. A faixa padrão de valores gerados é de 0 a 100 inclusive. Para a geração de dados do tipo caractere, não há uma faixa préestabelecida: os dados gerados serão sempre strings de 5 letras maiúsculas.
aleatorio <valor1> [, <valor2>]	Ativa a geração de dados numéricos aleatórios estabelecendo uma faixa de valores mínimos e máximos. Se apenas < valor1> for fornecido, a faixa será de 0 a inclusive; caso contrário, a faixa será de a inclusive. Se for menor que , o VisuAlg os trocará para que a faixa fique correta. Importante: e devem ser constantes numéricas, e não expressões.
aleatorio off	Desativa a geração de valores aleatórios. A palavra-chave off é obrigatória.

Nota: No VisuAlg, sempre que um comando leia for encontrado, a digitação de valores numéricos e/ou caracteres é substituída por uma geração aleatória.

Comando Arquivo

A sintaxe do comando é: arquivo ***<nome-de-arquivo>***
<nome-de-arquivo> é uma constante caractere (entre aspas duplas).

Nota: Muitas vezes é necessário repetir os testes de um programa com uma série igual de dados. Para casos como este, o VisuAlg permite o armazenamento de dados em um arquivo-texto, obtendo deles os dados ao executar os comandos leia.

Comando Pausa

Sua sintaxe é simplesmente: pausa

Nota: Este comando insere uma interrupção incondicional no pseudocódigo. Quando ele é encontrado, o VisuAlg pára a execução do pseudocódigo e espera alguma ação do programador.

Comando Debug

Sua sintaxe é: ***debug <expressão-lógica>***

Nota: Se a avaliação de <expressão-lógica> resultar em valor VERDADEIRO, a execução do pseudocódigo será interrompida como no comando pausa.

Comando Eco

Sua sintaxe é: ***eco on | off***

Nota: Este comando ativa (eco on) ou desativa (eco off) a impressão dos dados de entrada na saída-padrão do VisuAlg, ou seja, na área à direita da parte inferior da tela.

Comando Cronômetro

Sua sintaxe é: ***cronometro on | off***

Nota: Este comando ativa (cronometro on) ou desativa (cronometro off) o cronômetro interno do VisuAlg. Quando o comando cronometro on é encontrado, o VisuAlg imprime na saída-padrão a informação "Cronômetro iniciado.", e começa a contar o tempo em milissegundos.

Comando Limpatela

Sua sintaxe é: ***limpatela***

Nota: Este comando simplesmente limpa a tela DOS do Visualg (a simulação da tela do computador). Ele não afeta a "tela" que existe na parte inferior direita da janela principal do Visualg.

Funções Pré-Programadas

Abs(expressão)	Retorna o valor absoluto de uma expressão do tipo inteiro ou real.
ArcCos(expressão)	Retorna o ângulo (em radianos) cujo co-seno é representado por expressão.
ArcSen(expressão)	Retorna o ângulo (em radianos) cujo seno é representado por expressão.
ArcTan(expressão)	Retorna o ângulo (em radianos) cuja tangente é representada por expressão.
Cos(expressão)	Retorna o co-seno do ângulo (em radianos) representado por expressão.
CoTan(expressão)	Retorna a co-tangente do ângulo (em radianos) representado por expressão.
Exp(base, expoente)	Retorna o valor de base elevado a expoente, sendo ambos expressões do tipo real.
GraupRad(expressão)	Retorna o valor em radianos ao valor em graus representado por expressão.
Int(expressão)	Retorna a parte inteira do valor representado por expressão.
Log(expressão)	Retorna o logaritmo na base 10 do valor representado por expressão.
LogN(expressão)	Retorna o logaritmo neperiano (base e) do valor representado por expressão.
Quad(expressão)	Retorna quadrado do valor representado por expressão.
RadpGrau(expressão)	Retorna o valor em graus ao valor em radianos representado por expressão.
RaizQ(expressão)	Retorna a raiz quadrada do valor representado por expressão.
Rand	Retorna um número real gerado aleatoriamente, maior ou igual a zero e menor que um.
Randi(limite)	Retorna um número inteiro gerado aleatoriamente, maior ou igual a zero e menor que limite.
Sen(expressão)	Retorna o seno do ângulo (em radianos) representado por expressão.
Tan(expressão)	Retorna a tangente do ângulo (em radianos) representado por expressão.
Asc (s : caracter)	Retorna um inteiro com o código ASCII do primeiro caracter da expressão.
Carac (c : inteiro)	Retorna o caracter cujo código ASCII corresponde à expressão.
Caracpnum (c : caracter)	Retorna o inteiro ou real representado pela expressão.
Compr (c : caracter)	Retorna um inteiro contendo o comprimento (quantidade de caracteres) da expressão.
Copia (c : caracter ; p, n : inteiro)	Retorna um valor do tipo caracter contendo uma cópia parcial da expressão, a partir do caracter p, contendo n caracteres.
Maiusc (c : caracter)	Retorna um valor caracter contendo a expressão em maiúsculas.
Minusc (c : caracter)	Retorna um valor caracter contendo a expressão em minúsculas.
Numpcarac (n : inteiro ou real)	Retorna um valor caracter contendo a representação de n como uma cadeia de caracteres.
Pos (subc, c : caracter)	Retorna um inteiro que indica a posição em que a cadeia subc se encontra em c, ou zero se subc não estiver contida em c.