UNIDADE III

Linguagem de Programação do VisuAlg

NDDigital

ALGORITMO - PORTUGOL

LAGES
(2014)

Sumário

1 UNIDADE 3 – MOSTRANDO TODOS OS COMANDOS

Nessa unidade daremos continuidades aos comandos da linguagem de programação do visualg, portugol.

1.1 COMANDO DE DESVIO CONDICIONAL

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

Ao encontrar este comando, o VisuAlg analisa a <expressão-lógica>(Condições). 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.

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

Nesta outra forma do comando, se o resultado da avaliação de <expressão-lógica> for VERDA-DEIRO, todos os comandos da <seqüência-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 <seqüência-de-comandos-2> (até a linha com fimse).

Estes comandos equivalem ao if...else do C#. Note que não há necessidade de delimitadores de bloco pois as sequências de comandos já estão delimitadas pelas palavras-

chave senao e fimse. O VisuAlg permite o alinhamento desses comandos de desvio condicional.

1.2 COMANDO DE SELEÇÃO MÚLTIPLA

O VisuAlg implementa (com certas variações) o comando switch case do C#. A sintaxe é a seguinte:

```
escolha <expressão-de-seleção>
caso <exp11>, <exp12>, ..., <exp1n>
<seqüência-de-comandos-1>
caso <exp21>, <exp22>, ..., <exp2n>
   <seqüência-de-comandos-2>
...
outrocaso
<seqüência-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", "Corínthians"
   escreval("É um time paulista.")
caso "Internacional", "Gremio"
   escreval("É um time gaucho.")
caso "Inter de Lages"
   escreval("É o time de LAGES.(Leão Baio)")
outrocaso
   escreval("É de outro estado.")
fimescolha
fimalgoritmo
```

1.3 COMANDOS DE REPETIÇÃO

O VisuAlg implementa as três estruturas de repetição usuais nas linguagens de programação: o laço contado para...ate...faca (similar ao for... do C#), e os laços condicionados enquanto...faca (simi-lar ao while...) e repita...ate (similar ao repeat...until). A sintaxe destes comandos é explicada a seguir.

1.3.1 Para ... faça

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

```
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.

<valor-inicial >, <valor-limite > e <incremento > 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 <incremento> for negativo), o laço não será executado nenhuma vez. O exemplo a seguir não imprime nada.

```
algoritmo "Numeros de 10 a 1 (não funcio-na)"
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
```

1.3.2 Enquanto ... faça

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

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

<expressão-lógica> Esta expressão que é avaliada antes de cada repetição do laço. Quando seu resultado for VERDADEIRO, <seqüência-de-comandos> é executada.

fimenquanto Indica o fim da <seqüência-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 <seqüência-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 enquan-to...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.

1.3.3 Repita ... até

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

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

repita Indica o início do laço.

ate <expressão-lógica> Indica o fim da <seqüência-de-comandos> a serem repetidos. Cada vez que o

<expressão-lógica> é avaliada: se seu resultado for programa chega neste ponto,

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 interrompa)"
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.

1.3.4 Comando Interrompa

As três estruturas de repetição acima permitem o uso do comando interrompa, que causa uma saída imediata do laço. Embora esta técnica esteja de certa forma em desacordo com os princípios da programação estruturada, o comando interrompa foi incluído no VisuAlg por ser encontrado na literatura de introdução à programação e mesmo em linguagens como o Object Pascal (Delphi/Kylix), C#, Clipper, VB, etc. Seu uso é exemplificado a seguir:

```
algoritmo "Números de 1 a 10 (com interrompa)"
var x:inteiro
inicio
x <- 0
repita
   x<-x+1
   escreva(x:3)
   se x = 10 entao
      interrompa
   fimse
ate falso
fimalgoritmo
```

O VisuAlg permite ainda uma forma alternativa do comando repita...ate, com a seguinte sintaxe:

```
algoritmo "Números de 1 a 10 (com interrompa) II"
var x: inteiro
inicio
x <- 0
repita
   x <- x + 1
   escreva(x:3)
   se x = 10 entao
      interrompa
   fimse
   fimrepita
fimalgoritmo
```

Com esta sintaxe alternativa, o uso do interrompa é obrigatório, pois é a única maneira de se sair do laço repita...fimrepita; caso contrário, este laço seria executado indeterminadamente.

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.

2 PROGRAMAÇÃO PROCEDURAL

2.1 PROCEDIMENTOS

Em VisuAlg, procedimento é um subprograma que não retorna nenhum valor (corresponde ao proce-dure 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> [(<seqüência-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 <seqüência-de-declarações-de-parâmetros> é uma seqüência de

```
[var] <seqüência-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, <seqüência-de-parâmetros> é uma seqüência 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 internas começa com a palavra-chave var, e continua com a seguinte sintaxe:

<lista-de-variáveis> : <tipo-de-dado>

Nos próximos exemplos, através de um subprograma soma, será calculada a soma entre os valores 4 e –9 (ou seja, será obtido o resultado 13) que o programa principal imprimirá em seguida. No primeiro caso, um procedimento sem parâmetros utiliza uma variável local aux para armazenar provisoriamente o resulta-do deste cálculo (evidentemente, esta variável é desnecessária, mas está aí apenas para ilustrar o exemplo), antes de atribuí-lo à variável global res:

```
procedimento soma
var aux: inteiro
inicio
// n, m e res são variáveis globais aux <- n + m
res <- aux
fimprocedimento
```

No programa principal deve haver os seguintes comandos:

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

A mesma tarefa poderia ser executada através de um procedimento com parâmetros, como descrito abaixo:

```
procedimento soma (x,y: inteiro)
inicio
// res é variável global
res <- x + y
fimprocedimento
```

No programa principal deve haver os seguintes comandos:

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

A passagem de parâmetros do exemplo acima chama-se passagem por valor. Neste caso, o subprograma simplesmente recebe um valor que utiliza durante sua execução. Durante essa execução, os parâmetros passados por valor são análogos às suas variáveis locais, mas com uma única diferença: receberam um valor inicial no momento em que o subprograma foi chamado.

2.2 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>[(<seqüência-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 <seqüênciade-declarações-de-parâmetros> é uma seqüência de

[var] <seqüência-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 internas 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, va-mos 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 inici
// 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, pode-ria 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
escreva(res)
```

2.3 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)
```

2.4 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 alinhamento 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.

O VisuAlg implementa algumas extensões às linguagens "tradicionais" de programação, com o intuito principal de ajudar o seu uso como ferramenta de ensino. Elas são mostradas a seguir.

2.5 COMANDO ALEATÓRIO

Muitas vezes a digitação de dados para o teste de um programa torna-se uma tarefa entediante. Com o uso do comando aleatorio do VisuAlg, sempre que um comando leia for encontrado, a digitação de valo-res numéricos e/ou caracteres é substituída por uma geração

aleatória. Este comando não afeta a leitura de variáveis lógicas: com certeza, uma coisa pouco usual em programação...

Este comando tem as seguintes sintaxes:

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 preestabelecida: 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 < va-lor1> for fornecido, a faixa será de 0 a <valor1> inclusive; caso contrário, a faixa será de <valor1> a <valor2> inclusive. Se <valor2> for menor que <valor1>, o VisuAlg os trocará para que a faixa fique correta.

Importante: <valor1> e <valor2> devem ser constantes numéricas, e não expressões.

aleatorio off Desativa a geração de valores aleatórios. A palavra-chave off é obrigatória.

2.6 COMANDO ARQUIVO

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.

Esta característica funciona da seguinte maneira:

1. Se não existir o arquivo com nome especificado, o VisuAlg fará uma leitura de dados através da digitação, armazenando os dados lidos neste arquivo, na ordem em que forem fornecidos.
2. Se o arquivo existir, o VisuAlg obterá os dados deste arquivo até chegar ao seu fim. Daí em diante, fará as leituras de dados através da digitação.

3. Somente um comando arquivo pode ser empregado em cada pseudocódigo, e ele deverá estar na seção de declarações (dependendo do "sucesso" desta característica, em futuras versões ela poderá ser melhorada...).
4. Caso não seja fornecido um caminho, o VisuAlg irá procurar este arquivo na pasta de trabalho corrente (geralmente, é a pasta onde o programa VISUALG.EXE está). Este

comando não prevê uma extensão padrão; portanto, a especificação do nome do arquivo deve ser completa, inclusive com sua extensão (por exemplo, .txt, .dat, etc.).

A sintaxe do comando é:

`arquivo <nome-de-arquivo>`

<nome-de-arquivo> é uma constante caractere (entre aspas duplas).

Veja o exemplo a seguir:

```
algoritmo "lendo do arquivo"
arquivo "teste.txt"
var x,y: inteiro
inicio
para x de 1 ate 5 faca
   leia (y)
fimpara
fimalgoritmo
```

2.6 COMANDO TIMER

Embora o VisuAlg seja um interpretador de pseudocódigo, seu desempenho é muito bom: o tempo gas-to 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 execu-ção 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-traso> Ativa o timer estabelecendo seu tempo de atraso em milissegundos. O valor padrão é

500, que equivale a meio segundo. O argumento <tempo-de-atraso> 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.

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.

2.7 COMANDOS DE DEPURAÇÃO

Nenhum ambiente de desenvolvimento está completo se não houver a possibilidade de se inserir pon-tos de interrupção (breakpoints) no pseudocódigo para fins de depuração. VisuAlg implementa dois comandos que auxiliam a depuração ou análise de um pseudocódigo: o comando pausa e o comando debug.

2.8 COMANDO PAUSA

Sua sintaxe é simplesmente: pausa

Este comando insere uma interrupção incondicional no pseudocódigo. Quando ele é encontrado, o Vi-suAlg para a execução do pseudocódigo e espera alguma ação do programador. Neste momento, é possível: analisar os valores das variáveis ou das saídas produzidas até o momento; executar o pseudocódigo passo a passo (com F8); prosseguir sua execução normalmente (com F9); ou simplesmente terminá-lo (com Ctrl-F2). Com exceção da alteração do texto do pseudocódigo, todas as funções do VisuAlg estão disponíveis.

2.9 COMANDO DEBUG

Sua sintaxe é:

```
debug <expressão-lógica>
```

Se a avaliação de <expressão-lógica> resultar em valor VERDADEIRO, a execução do pseudocódigo será interrompida como no comando pausa. Dessa forma, é possível a inserção de um breakpoint condicional no pseudocódigo.

```
Comando Eco

Sua sintaxe é:

eco on | off
```

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. Esta característica pode ser útil quando houver uma grande quantidade de dados de entrada, e se deseja apenas analisar a saída produzida. Convém utilizá-la também quando os dados de entrada provêm de um arquivo já conhecido.

2.10 COMANDO CRONÔMETRO

Sua sintaxe é:

```
cronometro on | off
```

Este comando ativa (cronometro on) ou desativa (cronometro off) o cronômetro interno do Vi-suAlg. 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. Quando o comando cronometro off é encontrado, o VisuAlg imprime na saída-padrão a informação "Cronômetro terminado. Tempo decorri-do: xx segundo(s) e xx ms". Este comando é útil na análise de desempenho de algoritmos (ordenação, busca, etc.).

2.11 COMANDO LIMPATELA

Sua sintaxe é:

```
limpatela
```

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.

2.12 AUTO DIGITAÇÃO E SUGESTÃO DE DIGITAÇÃO

O VisuAlg tem uma característica para a criação de pseudocódigos que pode aumentar a rapidez da digitação e também diminuir a possibilidade de erros: é a auto digitação. Para utilizar esta característica, basta escrever uma abreviatura da palavra-chave ou do comando a ser digitado e teclar Ctrl-Espaço. O VisuAlg completa então o comando automaticamente, colocando o cursor no ponto adequado para se continuar a digi-tação (nos exemplos abaixo, este ponto é indicado através de um *). Eis a lista de abreviaturas com os res-pectivos comandos:

! - (Ponto de exclamação) Cria um modelo de pseudocódigo.

algoritmo "semnome"

* inicio fimal-goritmo

- Cria um cabeçalho de programa.

// Algoritmo : *

// Função :

// Autor :

// Data :

ale, aof, aon - Inserem os comandos aleatorio, aleatorio off ou aleatorio on, respecti-vamente.

alg - Insere a linha algoritmo e pede a digitação do seu nome. algoritmo "*"

arq - Insere o comando arquivo e pede a digitação do seu nome. arquivo "*"

cof, con - Inserem os comandos cronometro off ou cronometro on, respecti-vamente.

dcc - Insere uma declaração de variáveis caracteres.

var * : caractere

dcl - Insere uma declaração de variáveis lógicas.

var * : logico

dcr - Insere uma declaração de variáveis reais.

var * : real

deb - Insere o comando debug.

eof, eon - Inserem os comandos eco off ou eco on, respectivamente.

esc - Insere o comando escreva.

escl - Insere o comando escolha (sem a cláusula outrocaso).

escolha * caso
fimescolha
esco - Insere o comando escolha (com a cláusula outroca-so).
escolha * caso outrocaso
fimescolha
enq - Insere o comando enquanto.
enquanto * faca
fimenquanto
fal - Insere a linha fimalgoritmo.
ini - Insere a linha inicio.
int - Insere o comando interrompa. lep - Insere o comando leia. leia (*)
par - Insere o comando para.
para * de 1 ate faca
fimpara
parp - Insere o comando para com passo.
para * de ate passo faca fimpara
rep - Insere o comando repita.
repita * ate
repf - Insere o comando repita com fimrepita.
repita *
fimrepita
see - Insere o comando se sem a alternativa senao.
se * entao fimse
ses - Insere o comando se completo.
se * entao senao fimse
tim - Insere os comandos timer on e timer off.
timer on * timer off
tof, ton - Inserem os comandos timer on ou timer off, respectivamente.

2.13 EXERCÍCIOS: FIXAÇÃO

1) Faça um algoritmo que receba um número e mostre uma mensagem caso este número seja maior que 10.

2) Escrever um algoritmo que leia dois valores inteiro distintos e informe qual é o maior.

3) Faça um algoritmo que receba um número e diga se este número está no intervalo entre 100 e 200.

4) Escrever um algoritmo que leia o nome e as três notas obtidas por um aluno durante o semestre. Calcular a sua média (aritmética), informar o nome e sua menção aprovado (media >= 7), Reprovado (media <= 5) e Recuperação (media entre 5.1 a 6.9).

5) Ler 80 números e ao final informar quantos número(s) est(á)ão no intervalo entre 10 (inclusive) e 150 (inclusive).

6) Faça um algoritmo que receba a idade de 75 pessoas e mostre mensagem informando “maior de idade” e “menor de idade” para cada pessoa. Considere a idade a partir de 18 anos como maior de idade.

7) Escrever um algoritmo que leia o nome e o sexo de 56 pessoas e informe o nome e se ela é homem ou mulher. No final informe total de homens e de mulheres.

8) A concessionária de veículos “CARANGO VELHO” está vendendo os seus veículos com desconto. Faça um algoritmo que calcule e exiba o valor do desconto e o valor a ser pago pelo cliente de vários carros. O desconto deverá ser calculado de acordo com o ano do veículo. Até 2000 - 12% e acima de 2000 - 7%. O sistema deverá perguntar se deseja continuar calculando desconto até que a resposta seja: “(N) Não” Informar total de carros com ano até 2000 e total geral.

9) Escrever um algoritmo que leia os dados de “N” pessoas (nome, sexo, idade e saúde) e informe se está apta ou não para cumprir o serviço militar obrigatório. Informe os totais.

10) Faça um algoritmo que receba o preço de custo e o preço de venda de 40 produtos. Mostre como resultado se houve lucro, prejuízo ou empate para cada produto. Informe média de preço de custo e do preço de venda.

12) Faça um algoritmo que receba um número e mostre uma mensagem caso este número sege maior que 80, menor que 25 ou igual a 40.

13) Faça um algoritmo que receba “N” números e mostre positivo, negativo ou zero para cada número

14) Escrever um algoritmo que lê um conjunto de 4 valores i, a, b, c, onde i é um valor inteiro e positivo e a, b, c, são quaisquer valores reais e os escreva. A seguir:

a) Se i=1 escrever os três valores a, b, c em ordem crescente.
b) Se i=2 escrever os três valores a, b, c em ordem decrescente.
c) Se i=3 escrever os três valores a, b, c de forma que o maior entre a, b, c fique dentre os dois.

15) Escrever um algoritmo que lê um valor em reais e calcula qual o menor número possível de notas de 100, 50, 10, 5 e 1 em que o valor lido pode ser decomposto. Escrever o valor lido e a relação de notas necessárias.

Desafio:

. Escrever um algoritmo que lê o número de identificação, as 3 notas obtidas por um aluno nas 3 verificações e a média dos exercícios que fazem parte da avaliação. Calcular a média de aproveitamento, usando a fórmula:

MA = (Nota1 + Nota2 x 2 + Nota3 x 3 + ME)/7

A atribuição de conceitos obedece a tabela abaixo:

Média de Aproveitamento	Conceito
9,0	A
7,5 e < 9,0	B
6,0 e < 7,5	C
4,0 e < 6,0	D
< 4,0	E

O algoritmo deve escrever o número do aluno, suas notas, a média dos exercícios, a média de aproveitamento, o conceito correspondente e a mensagem: APROVADO se o conceito for A,B ou C e REPROVADO se o conceito for D ou E.

Bons estudos!