

Web Games

Prof: Luciano Alves da Silva

1) Fundamentos básicos sobre jogos

Os jogos nada mais são do que programas de computador destinados ao entretenimento. Atualmente existem jogos dos mais diversos gêneros, ação, aventura, esportes e etc. Confira alguns títulos abaixo:



Sonic – The hedgehog (Ação)



Mortal Kombat Trilogy (Luta)



Top Gear 3000 (corrida)

Para se criar um jogo, de qualquer gênero que seja, é preciso primeiro antes de tudo ter todo o enredo (história) do jogo. Nesta parte do planejamento do jogo definimos do que se trata o jogo, quais personagens e entre outros elementos vão participar nesse jogo, o objetivo de cada personagem (ou elemento) no jogo e etc.

O outro passo importante é o recrutamento dos profissionais que vão participar do desenvolvimento do jogo (envolvendo artistas e programadores) e a escolha da ferramenta (IDE) utilizada.

Para este módulo estaremos desenvolvendo jogos voltados para a Web. Atualmente a ferramenta de desenvolvimento de jogos voltados para esse tipo de plataforma é o Adobe Flash Professional. Essa ferramenta é bastante fácil de se trabalhar e favorece muito o desenvolvedor durante a criação dos seus jogos.

2) Fundamentos básicos sobre linguagens de programação e algoritmos

Bom, primeiramente precisamos saber que o “Game Developer” é o profissional responsável por dar “vida” ao jogo. Ele fará uso de artes e elementos prontos que constituem um jogo e dará movimento, ação a ele através do uso de instruções de computador. Aqui neste módulo e nos próximos iremos sempre estar vendo lógica de programação, e o que iremos trabalhar daqui deste módulo por diante é o que nós chamamos de “linguagem de programação”. Mas afinal de contas, o que vem a ser uma “linguagem de programação” ? Uma linguagem de programação consiste em um conjunto de regras que precisam ser seguidas para que um programa de computador possa ser construído. Atualmente existem diversas linguagens de programação disponíveis hoje em dia como C++, Java, Pascal, Visual Basic e etc., cada uma com sua sintaxe. Para o desenvolvimento dos nossos jogos no Flash, vamos estudar a linguagem de programação ActionScript, que é a linguagem base da plataforma.

Um programa de computador nada mais é do que um conjunto de instruções que são executados pelo computador para realizar uma determinada tarefa proposta (Ex: Word, Excel, Jogo).

Para entendermos melhor o processo de desenvolvimento de jogos usando uma linguagem de programação , precisamos entender que um programa (como um jogo), nada mais é do que um “algoritmo”.

Mas afinal, o que vem a ser a ser um algoritmo ? Um algoritmo nada mais é do que um conjunto de passos ordenados cujo objetivo é solucionar um determinado problema proposto.

Para entendermos melhor o que vem a ser esse conceito, vamos demonstrar um algoritmo da vida real cujo objetivo é : fazer café. Vamos supor que temos todos os ingredientes necessários para fazer um café. Veja os passos abaixo:

- 1 – Encher a chaleira de água
- 2 – Colocar a chaleira para ferver
- 3 – Preparar o porta-filtro com o filtro sobre o bule
- 4 – Colocar duas colheres de sopa de pó de café no filtro

5 – Após a água ter fervido, acrescentar aos poucos meio litro de água sobre o filtro

6 – Aguarde coar

7 – Adoçar a gosto.

Se seguirmos os passos acima, o objetivo “fazer café” será realizado. Isso é um algoritmo.

Agora vamos supor que o café não esteja presente, como faremos um algoritmo para tratar dessa situação ? Veja como ficou abaixo:

1 – Encher a chaleira de água

2 – Colocar a chaleira para ferver

3 – Enquanto a chaleira estiver no fogo

4 – Preparar o porta-filtro com o filtro sobre o bule

5 – Se houve café, coloque duas colheres de sopa de pó de café no filtro de água sobre o filtro

6 – Se não houver café, desligar o fogo e ir para o supermercado para comprar café.

7 – Repetir os passos 2, 3 e 5.

8 – Após a água ter fervido, acrescentar aos poucos meio litro de água sobre o filtro

9 – Aguarde coar

10 – Adoçar a gosto.

Se observamos acima, os dois algoritmos acima realizam o mesmo objetivo: fazer um café. O primeiro realizou esse objetivo em 7 passos e o segundo realizou o mesmo objetivo em 10 passos. Existem certos algoritmos que podem ter mais ou menos linhas de código para realizar uma determinada tarefa, mas o importante é que o algoritmo atinja o objetivo proposto.

3) Trabalhando com algoritmos na prática

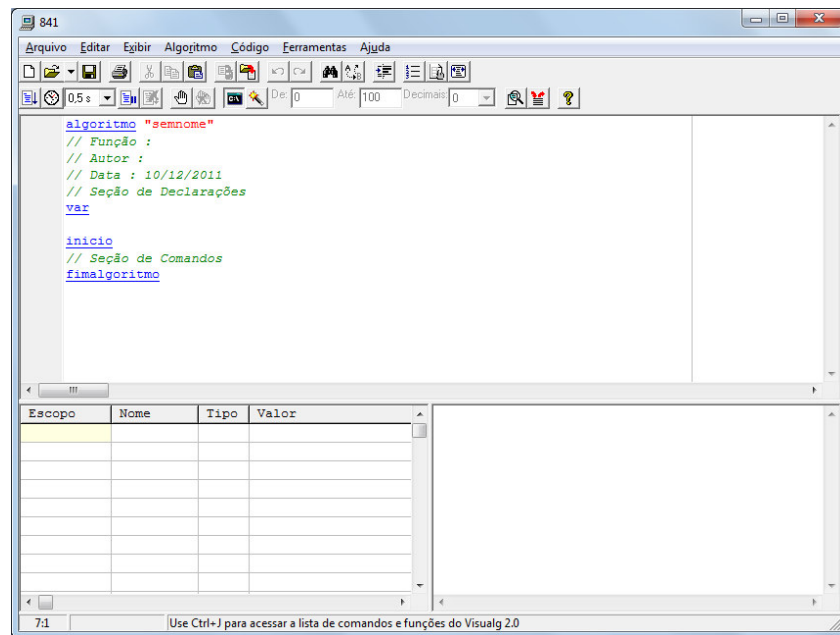
Como estamos começando nossa jornada no “mundo da programação” é preciso conhecermos como funciona a lógica de programação e toda a sua “essência” , pois, levaremos tudo isso para as futuras linguagens de programação que iremos encontrar pela frente (ActionScript, C# e entre outras).

Antes de manipularmos o Flash (ActionScript) por definitivo, irei abordar alguns conceitos essenciais sobre algoritmos demonstrando aqui o uso do “Portugol” (também conhecido como português estruturado). Esse será o nosso “ponto de partida” para o aprendizado sobre lógica de programação.

Irei demonstrar o aprendizado sobre algoritmos através da ferramenta de programação VisuAlg, que pode ser baixado “livremente” no link abaixo:

<http://www.baixaki.com.br/download/visualg.htm>

Depois de baixar e efetuar sua instalação execute o aplicativo. Veja-o na figura abaixo:



VisuAlg

Observe que dentro da ferramenta existe um “código” pré-definido onde nele iremos construir nosso programa (algoritmo). Para desenvolvermos um programa nessa

ferramenta, precisamos antes de tudo, conhecer a sintaxe e os comandos que essa linguagem de algoritmos nos oferece. Primeiramente vamos estudar o código gerado para entendermos como funciona o processo de desenvolvimento de um programa.

Vejamos a primeira linha de código do nosso algoritmo:

algoritmo “sem nome”

Observe que a palavra “algoritmo” está sublinhada “em azul”. O que significa isso ? Isso significa que a palavra “algoritmo” é uma palavra reservada. Uma palavra reservada “em programação” é uma palavra que já possui uma finalidade em específico dentro dela. Para esta linguagem, “algoritmo” nada mais é do que uma palavra reservada utilizada para definir o nome do nosso programa. Por padrão o nome do nosso programa está definido como “sem nome” . Todo texto escrito entre aspas fica destacado com cor vermelho. Isso é normal na maioria das linguagens de programação (algumas linguagens destacam com cor azul, isso varia da ferramenta a ser utilizada).

Lembre-se que é preciso conhecer as regras da linguagem para construirmos um programa. Na linha acima, a palavra reservada algoritmo vem seguido pelo nome do programa colado entre “aspas”. Isso é uma regra dessa linguagem.

Em algumas linhas seguintes existem algumas frases que levam antes delas duas barras // e elas ficam destacadas com cor verde. Isso indica um comentário. Um comentário é útil quando queremos descrever o funcionamento de um determinado trecho do código ou quando queremos fazer alguma observação/anotação qualquer.

Todo comentário , nessa linguagem (e em algumas das linguagens mais conhecidas) devem começar com duas barras // seguido da frase que você deseja digitar.

Mais à frente , encontramos a palavra chave “inicio”, destacada em azul. Ela é uma palavra reservada que marca o inicio do bloco de comandos que vai constituir meu programa. Em seguida, encontramos a palavra chave “finalgoritmo”, que define o fim do meu bloco de comandos da linguagem. Logo, todo o código do meu programa deve estar entre “inicio” e “finalgoritmo”.

Bom, vamos construir nosso primeiro programa cuja finalidade é exibir uma mensagem na tela. Para realizar esta operação, preciso conhecer qual comando da minha linguagem realiza tarefa. O comando que faz isso é o comando “escreva”.

Todo comando possui uma regra sintaxe que é preciso seguir para que o mesmo seja escrito corretamente sem erros. Para exibir uma mensagem na tela como por exemplo

“Seven game”, preciso colocar dentro da seção de comandos o comando escreva seguido da frase a ser impressa na tela. Veja como é feito a sua chamada na linha de código abaixo:

[inicio](#)

```
escreva("Seven Game")
```

[finalgoritmo](#)

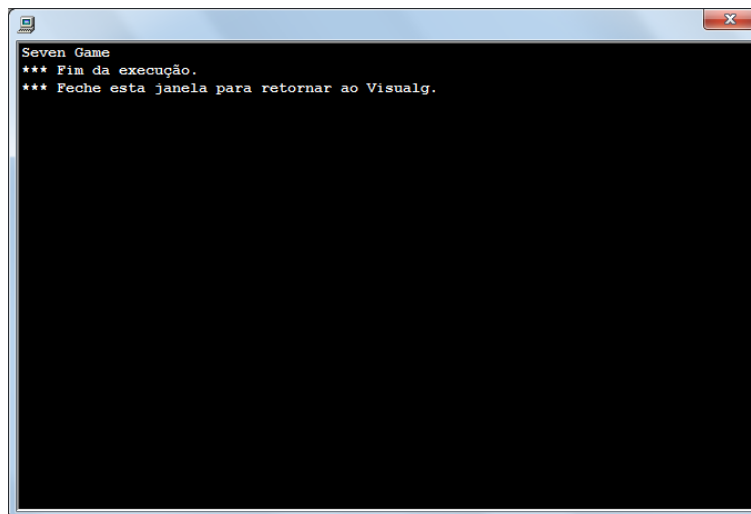
Observe que logo após a palavra “escreva” teve que se colocar um abre e fecha parênteses, e dentro dele, colocamos a frase que desejamos imprimir na tela. Essa é a regra sintática do comando escreva. Se eu quisesse imprimir na tela a frase “Estou aprendendo Flash”, bastava colocar isso:

[inicio](#)

```
escreva("Estou aprendendo Flash")
```

[finalgoritmo](#)

Vamos executar nosso programa ? Para isso basta pressionar a tecla F9 para ver o resultado. Confira na figura abaixo:



Programa em execução

Isso que nos realizamos acima, por mais simples que seja, é um algoritmo cujo objetivo é mostrar uma mensagem na tela.

3.1) Trabalhando com variáveis

Uma das coisas que os programas de computador sempre trabalham e com dados e informações.

Dado: é todo elemento a ser registrado no computador, ou seja, são os elementos de “entradas de dados”.

Informação: nada mais é do que o dado “processado”, ou seja, são os elementos de saída de dados.

Para entendermos melhor o que vem a ser dados e informações, considere o seguinte exemplo sobre algoritmos abaixo:

Algoritmo: Fazer um programa para ler as duas notas de um aluno e em seguida, calcular a média.

Entrada de dados : As duas notas

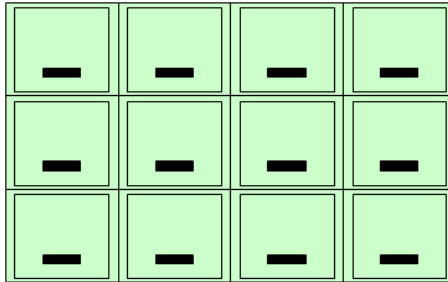
Processamento : Somar as duas notas e dividir por 2

Saída de dados: A Média.

Podemos concluir acima que o computador está sempre trabalhando em um ciclo de vida que envolve entrada, processamento e saída de dados. Essa é a atividade constante e comum dele.

Em um programa (como um jogo) os dados e as informações são representadas pelas variáveis. Mas afinal de contas, o que são variáveis ? Vamos fazer a seguinte analogia, imaginemos que a memória seja um armário com várias gavetas, e que cada gaveta possui um rótulo (nome) que em determinado momento guarda um conteúdo (informação).

Fazendo uma analogia sobre variáveis:



Armário com várias gavetas (simulando variáveis)

Podemos concluir que:

Variável é um endereço de memória que pode ter seu conteúdo alterado a qualquer momento durante a execução do programa.

Tipos de dados

Cada dado armazenado na memória possui um tipo. Os tipos de dados mais comuns são:

inteiro : Armazena valores numéricos inteiros, sem nenhuma fração .

real : Armazena valores numéricos de ponto flutuante , envolvendo valores quebrados.

caractere : Armazena valores que envolvem uma cadeia de caracteres, que podem ser números , letras e símbolos.

logico : Armazena somente valores lógicos como **verdadeiro** e **falso**.

3.2) Declarando variáveis

Na programação de modo geral, toda variável antes de ser usada, precisa ser antes declarada (essa é uma regra clássica da maioria das linguagens de programação). Na linguagem de programação que estamos aprendendo agora (Portugol ou também português estruturado), existe uma seção chamada **var** onde nela declaramos todas as variáveis que iremos utilizar dentro da linguagem. Vamos ver abaixo a sintaxe de como declaramos uma variável:

Sintaxe:

```
var  
    <variavel1>, ..., <variaveln> : <tipo>
```

Exemplo:

```
var  
    numero : inteiro
```

Quando temos mais de uma variável de mesmo tipo de dados, podemos separá-las por vírgula antes de informar seu tipo, como você confere abaixo:

```
var  
    nota1, nota2, media : real
```

Para demonstrar o uso das variáveis, vamos fazer um programa simples que vai ler “do teclado” um número e em seguida, esse programa vai mostrar o número lido na tela. Com o VisuAlg aberto, digite o código abaixo:

```
algoritmo "leia numero"  
  
var  
    numero : inteiro  
  
inicio  
  
    escreval("Digite um numero inteiro")  
  
    leia(numero)  
  
    escreval("O numero digitado foi : ", numero)  
  
fimalgoritmo
```

Vamos analisar o código acima. Observe que dentro da seção var foi declarada uma variável chamada “numero” para armazenar um número inteiro, logo, o tipo de dados que ela irá assumir será o tipo “inteiro”, confira abaixo:

var

numero : inteiro

A primeira linha de comando dentro da seção de bloco de comandos faz uso do comando **escreval** e não **escreva** ? Qual a diferença de um para o outro. O **escreva** logo após imprimir a frase, coloca o cursor “ao lado” dela, logo, qualquer coisa que fosse acontecer na tela irá partir do curso onde ele se encontra, se fosse para imprimir um novo texto ele sairia colado com o texto anteriormente exibido na tela. Já o comando **escreval** logo após imprimir o texto na tela, coloca o cursor na linha de baixo logo, qualquer coisa que fosse acontecer na tela, iria acontecer a linha abaixo do texto. Confira o comando na linha abaixo:

```
escreval("Digite um numero inteiro")
```

Na linha de baixo “leio” as informações digitadas pelo teclado através do comando **leia** , como parâmetro informo a variável que irá receber as informações digitadas do teclado (que no caso é a variável número). Veja o comando abaixo:

```
leia(numero)
```

Na linha seguinte exibo a frase seguido pelo nome da variável, que irá mostrar o conteúdo contido nela. Confira abaixo:

```
escreval("O numero digitado foi : ", numero)
```

Para exibirmos um conteúdo da variável, o nome da variável NÃO PODE estar entre “aspas”, conforme você pode conferir acima.

3.3) Comando de atribuição

Até agora aprendemos a informar um determinado valor para uma variável através da leitura de dados “via teclado”, mas, como fazemos para que uma variável receba um valor sem ser pelo teclado ? Para isso, usamos o “comando de atribuição” que serve atribuir um determinado valor ou expressão para uma variável. Veja a sua sintaxe abaixo:

Sintaxe:

```
<variavel1> <- <valor>
```

Exemplos

```
numero <- 12
```

Operadores aritméticos

É possível em uma atribuição trabalhar com cálculos e expressões utilizando os operadores aritméticos. Veja na tabela abaixo os operadores aritméticos básicos da linguagem:

Símbolo	Operador
+	Adição
-	Subtração
*	Multiplicação
/	Divisão

Exemplos:

```
Media <- (nota1 + nota2) / 2
```

```
Soma <- numero1 + numero2
```

```
w <- Y * K
```

Vamos agora fazer um programa para ler duas notas de um aluno e calcular a sua média. Com o VisuAlg aberto, digite o código abaixo:

```
algoritmo "calcular a media"  
var  
  nota1, nota2, media : real
```

inicio

```
escreval("Digite a primeira nota")
leia(nota1)
escreval("Digite a segunda nota")
leia(nota2)
media <- (nota1 + nota2) / 2
escreval("A media é : ", media)
```

fimalgoritmo

Se observarmos o código acima temos três variáveis : nota1, nota2 e media. Todas elas do tipo real (tipo que armazena números com fração) O programa acima realiza a média aritmética de duas notas de um aluno. O cálculo é efetuado na linha de comando abaixo:

```
media <- (nota1 + nota2) / 2
```

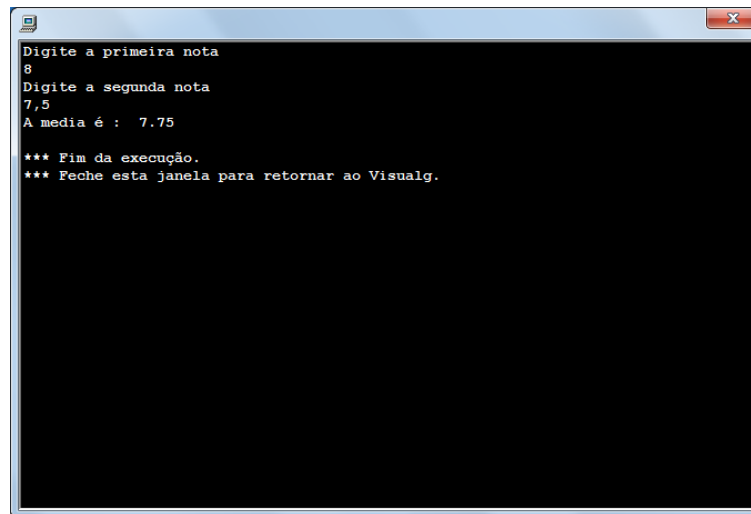
Primeiramente é resolvido a soma das notas dentro dos parênteses e em seguida é efetuada a divisão do valor por 2. Se a atribuição estivesse dessa forma:

```
media <- nota1 + nota2 / 2
```

Primeiramente seria efetuada a divisão da nota2 por 2 e depois a soma do resultado com nota1. Mas porque isso acontece ? Simples, como na matemática, os operadores tem precedência um sobre os outros. Veja na tabela abaixo a precedência deles.

Prioridade	Operador
Alta prioridade	()
	/
	*
Baixa prioridade	+ -

Veja a execução do programa na figura abaixo:



4) Estruturas condicionais

Até agora vimos que um algoritmo consiste na execução de instruções passo a passo, uma após a outra mas, em algumas situações certas instruções precisam ser executadas caso alguma condição seja satisfeita.

Na programação, alguns (ou melhor, a maioria) dos programas possuem comandos que precisam ser executados caso alguma condição seja satisfeita. Para esse tipo de situação utilizamos as “estruturas condicionais”. Vamos conhecer a estrutura condicional básica, a estrutura “se”:

4.1) A estrutura se

A estrutura condicional “se” executa um bloco de comandos se uma determinada condição for satisfeita.

Sintaxe:

se (<condição>) entao

<comando1>;

<comando2>;

:

<comandon>;

fimse

Para demonstrar o uso dessa estrutura vamos fazer um algoritmo para ler um número do teclado e só exibi-lo caso o número seja maior que 10. Veja o código abaixo:

algoritmo "estrutura se"

var

numero : inteiro

inicio

```
escreval("Digite um numero")
leia(numero)
se(numero > 10) entao
    escreval("O numero é maior que 10")
fimse
```

fimalgoritmo

Se executarmos o programa acima ele irá efetuar a leitura de dados via teclado para a variável “numero”. Em seguida, vai verificar se a variável é maior que 10, se for verdadeiro a situação , o programa vai mostrar a mensagem “O numero é maior que 10” , caso contrário, não exibirá nada.

4.2) A estrutura se/senao

Em algumas situações , precisamos que instruções sejam executadas caso uma condição seja verdadeira e outras caso uma condição seja falsa. Até agora aprendemos a executar um bloco de comandos caso a condição seja verdadeira na estrutura “se”. Agora, se eu quisesse executar um bloco de comandos caso uma condição seja falsa ? Para isso utilizamos a estrutura “senão” do “se”.

Sintaxe:

se (<condição>) entao

```
<executa um bloco de comandos se a condição acima
for verdadeira>
```

senao

```
<executa um bloco de comandos se a condição acima
for falsa>
```

fimse

Vamos utilizar o mesmo programa acima só que utilizando a estrutura se/senão.

algoritmo "estrutura se"

var

numero : inteiro

inicio

```

escreval("Digite um numero")
leia(numero)
se(numero > 10) entao
    escreval("O numero é maior que 10")
senao
    escreval("O numero é menor ou igual a 10")
fimse
    
```

fimalgoritmo

O programa acima efetua a leitura de dados, via teclado, para a variável número. Na estrutura condicional acima é avaliado se o número é maior que 10, se a condição for verdadeira será exibida a seguinte mensagem: “O número é maior que 10”. Se por acaso a condição acima for falsa, será exibida a mensagem “O número é menor ou igual que 10”.

4.3) Operadores relacionais

Um algoritmo em português estruturado, ou em qualquer linguagem de programação, possui seus operadores relacionais, que podem ser utilizados dentro de uma avaliação condicional. Veja na tabela abaixo os operadores relacionais pertencentes nessa linguagem.

Nome da operação	Operador
Maior que	>
Maior ou igual que	>=
Menor que	<
Menor ou igual que	<=
Igual	=
Diferente	<>

4.4) Operadores lógicos

É muito comum em estruturas condicionais efetuarmos mais de uma avaliação condicional. Para isso, precisamos conhecer os operadores lógicos para tal finalidade. Vejamos agora alguns dos operadores lógicos básicos existentes nessa e em outras linguagens

O operador lógico “E”

Retorna verdadeiro quando todas as condições forem verdadeiras.

Condicao1	Condicao 2	Operador E
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

O operador lógico “OU”

Retorna verdadeiro quando pelo menos uma das condições for verdadeira.

Condicao 1	Condicao 2	Operador OU
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

O operador lógico “NÃO”

Ele funciona como inversor de uma condição, retornando falso se uma condição for verdadeira e retorna verdadeiro se uma condição for falsa.

Condicao	Operador Não
Verdadeiro	Falso
Falso	Verdadeiro

Exemplos:

```
algoritmo "exemplo se1"

var

    numero : inteiro

inicio

    escreval("Digite um numero")

    leia(numero)

    se( (numero >=10) e (numero <= 90)) entao

        escreval("Dentro da faixa")

    senao

        escreval("Fora da faixa")

    fimse

finalgoritmo
```

```
algoritmo "exemplo se2"  
  
var  
  
    sexo : caractere  
  
inicio  
  
    escreval("Qual seu sexo ? [masculino/feminino]")  
  
    leia(sexo)  
  
    se( (sexo = "masculino") ou (sexo = "feminino")) entao  
        escreval("Sexo válido")  
  
    senao  
        escreval("Sexo inválido")  
  
fimse  
  
fimalgoritmo
```

```
algoritmo "exemplo se3"  
  
var  
  
    numero : inteiro  
  
inicio  
  
    escreval("Digite um numero")  
  
    leia(numero)  
  
    se( (nao(numero > 10)) ) entao  
        escreval("O número não é maior que 10")  
  
fimse  
  
fimalgoritmo
```

5) Estruturas de repetição

Estruturas de repetição são estruturas que executa um bloco de comandos repetidas vezes , caso a condição para a execução dessa repetição seja satisfatória. Todas as linguagens de programação

5.1) Estrutura “enquanto”

A estrutura “enquanto”, como o próprio nome diz,, repete um conjunto de instrução enquanto uma determinada condição for verdadeira.

Sintaxe:

```
enquanto(<condicao>) faca
    <comandos1>
    <comando2>
    :
    <comandon>
fimenquanto
```

Exemplo:

```
algoritmo "exemplo enquanto"
var
    numero : inteiro
inicio
    numero <- 1
    enquanto (numero <= 10) faca
        escreval(numero)
        numero <- numero + 1
    fimenquanto
```

finalgoritmo

Vamos analisar o código do programa acima e entender como funciona a estrutura de repetição acima. Antes de entrar dentro do loop, é executada a seguinte instrução :

```
numero <- 1
```

Que atribui o valor “1” para a variável “numero”. Logo após vem a seguinte linha de comando :

```
enquanto (numero <= 10) faca
```

Que avalia se a variável “numero” possui valor menor ou igual a “10”, a condição acima será verdadeira pois “numero” assume o valor “1”, logo, o bloco de comandos dentro da estrutura será executada. A primeira linha de comando:

```
escreval(numero)
```

Vai mostrar na tela o conteúdo da variável “numero” na tela (neste caso na tela só vai aparecer “1”). Na linha seguinte:

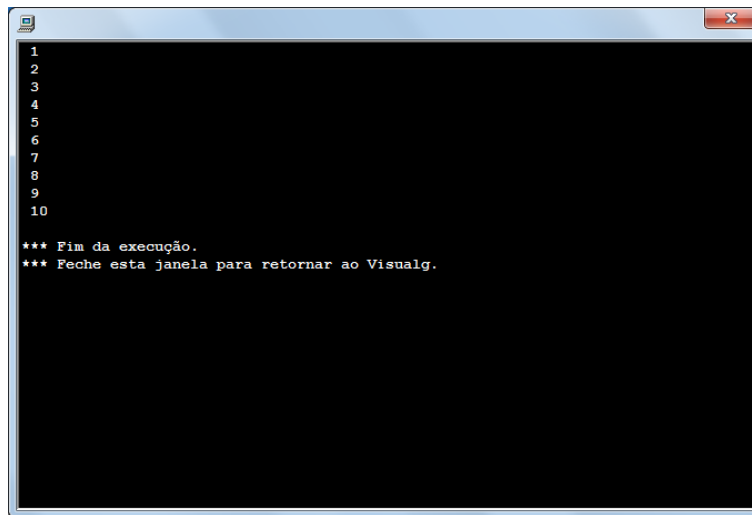
```
numero <- numero + 1
```

Ocorrerá o que nós chamados de “incremento” . Nesta linha estou somando o conteúdo da variável número (que contém “1”) mais “1”, resultando em “2”. O valor é novamente atribuído e atualizado na variável número, passando agora a ter o valor “2”. Como estou dentro de uma estrutura de repetição, após acabar a execução de todas as instruções dentro dela, novamente é feita a avaliação abaixo:

```
enquanto (numero <= 10) faca
```

Que avalia se a variável “numero” possui valor menor ou igual a “10”, a condição acima será verdadeira pois “numero” assume agora o valor “2”, logo, o bloco de comandos dentro da estrutura será executada novamente.

Esse ciclo irá se repetir enquanto a variável for menor ou igual a “10”. Veja o resultado na figura abaixo:



```
1
2
3
4
5
6
7
8
9
10

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```