



Departamento de Engenharia Rural

Centro de Ciências Agrárias

Programação I

```
public class Boneco2D
{
    private Rectangle quadro;
    private Texture2D textura;
    private bool paria;
    private int altura, largura;

    //construtor
    public Boneco2D(int posx, int posy, Texture2D t)
    {
        // inicializa o boneco com altura e largura proporcionais a suas coordenadas na tela
        if (posx == 0 || posy == 0)
        {
            posx = 10;
            posy = 10;
        }

        float propx = (float)posx / (float)t.height;
        altura = (int)Math.Round(t.height * propx);
    }
}
```

Prof. Bruno Vilela Oliveira

bruno@cca.ufes.br

<http://www.brunovilela.webnode.com.br>

- ❖ Elementos da programação Estruturada (Portugol e Diagrama de Blocos)
 - Estruturas sequenciais
 - Estruturas de seleção
 - Estruturas de iteração (repetições)

Instruções de controle

Aponte seu Lápis



Preciso de sua ajuda novamente.
Seu algoritmo para cálculo da
média funcionou muito bem. Mas
ele deve fazer algo mais...



Instruções de controle

Aponte seu Lápis



Precisamos que o algoritmo verifique se o aluno ficou de prova final (média < 7.0). Se o aluno tiver ficado de final, o algoritmo deve solicitar a nota da prova final, calcular e exibir a média final do aluno.

$$\text{Média final} = (\text{prova final} + \text{média}) / 2$$



A Estrutura Sequencial

A **estrutura sequencial** permite que instruções sejam executadas uma após a outra, ou seja, são executadas em sequência na ordem que foram escritas, a não ser que você especifique de outra forma.



❖ Sequências em Portugol:

- Para especificar uma estrutura sequencial em Portugol, basta escrever as instruções (de preferência uma por linha) uma após a outra, cada uma encerrando com um “ponto-e-vírgula”:

...

C1;

C2;

...

Cn;

(observe a indentação)

● Exemplo

...

Leia(n1,n2,n3,n4);

Soma ← (n1+n2+n3+n4);

Media ← Soma/4;

...

Um programa é mais que uma lista de instruções.

Você *poderia* criar um programa que seria simplesmente uma **lista** de comandos.

Mas quase nunca será tudo assim. Isto porque uma simples **lista** de comandos só pode levá-lo a uma única direção. É como dirigir em um trecho de estrada que é uma reta: existe apenas uma direção a seguir.



Os programas devem ser mais inteligentes que isto.

Relembrando seu algoritmo de cálculo da média:

```
1  {Calcular a média de um aluno.}
2  Algoritmo CALCULA_MEDIA;
3  var
4      P1, P2, MEDIA: real;
5  Início
6      leia (P1);
7      leia (P2);
8      MEDIA  $\leftarrow$  (P1 + P2) / 2;
9      escreva (MEDIA);
10 Fim.
```

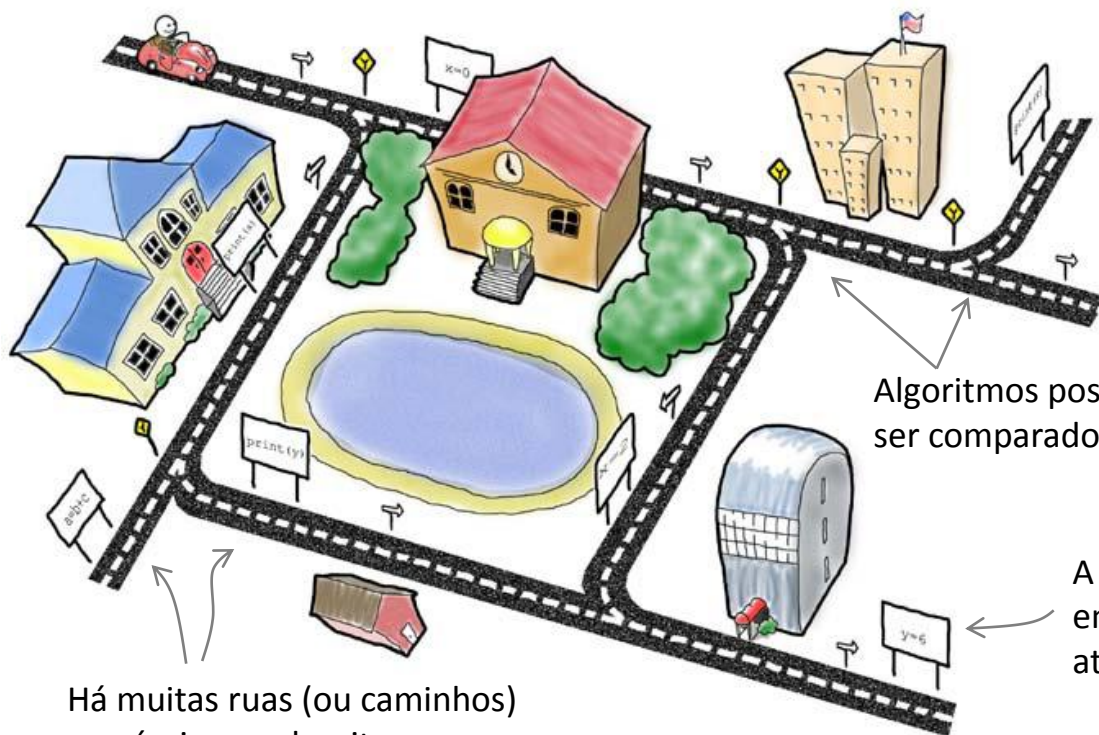
A primeira versão do seu algoritmo possui apenas **estruturas sequenciais**. Este algoritmo **sempre** realizará as **mesmas ações** quando for **executado**. No entanto, agora você precisa **tomar uma decisão**: **SE** a média do aluno for inferior a 7.0, o algoritmo deverá realizar um conjunto especial de ações: **obter a nota da prova final**, **calcular** e **exibir** a **média final**.

Instruções de controle

Codigolândia: Seus algoritmos são como uma rede de ruas.

Algoritmos precisam fazer coisas diferentes em diferentes situações.

Os algoritmos (programas), mesmo os mais simples, tipicamente possuem vários **caminhos** que podem ser escolhidos.



Algoritmos possuem pontos de decisão que podem ser comparados à bifurcações (desvios) em ruas..

A execução realiza os comandos que são encontrados no caminho que for seguido através do algoritmo.

Há muitas ruas (ou caminhos) possíveis nos algoritmos.

Instruções de controle



- ❖ Um caminho se refere ao conjunto de instruções que serão efetivamente seguidas (ou executadas) no algoritmo.
- ❖ Seu algoritmo é como uma rede de ruas, com várias seções de “**código**” interligadas tal como as ruas em uma cidade.
- ❖ Quando você dirige através de uma cidade, toma decisões como “**qual rua seguir?**” fazendo uma curva à esquerda ou à direita (ou mesmo evitando alguma curva) em diferentes interseções de ruas.



Instruções de controle



- ❖ Isto é muito parecido com o que acontece em um algoritmo (e conseqüentemente nos **programas** de computador).
- ❖ Algoritmos precisam fazer coisas diferentes em diferentes situações. Também precisam tomar decisões de **tempos em tempos** como qual **caminho** escolher, mas neste caso, não é como dirigir por uma estrada, trata-se da execução das instruções de um caminho particular.

Vejamos em mais detalhes como um algoritmo (programa) decide qual caminho seguir.



Instruções de desvio são bifurcações no código

Dirigir em uma rua é fácil. Você só precisa tomar uma decisão quando chega em uma bifurcação. O mesmo ocorre nos algoritmos.

Quando há uma lista de comandos, eles podem ser executados um após o outro.

Algumas vezes, seu algoritmo precisa tomar uma decisão: se executa “esse” ou “aquele” trecho de código.

Esses pontos de decisão são chamados de **instruções de desvio**.

Estruturas de seleção

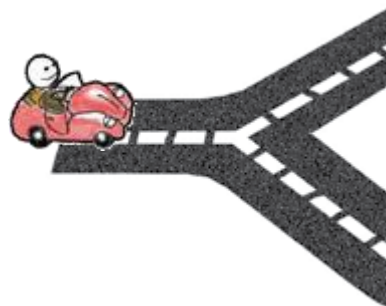


Instruções de desvio são bifurcações no código

Na programação estruturada as **estruturas de seleção** representam as **instruções de desvio** e são como **bifurcações** em seu algoritmo.

Uma instrução de desvio escolhe uma sequência comandos a ser executada.

A instrução de desvio mais simples é também chamada de **instrução condicional**.



Instruções de desvio são como **bifurcações** em estradas



Expressão condicional

Em **instruções de desvio**, seus algoritmos tomam decisões usando uma **expressão condicional**, **condição de desvio** ou simplesmente **condição**.

Uma **condição** tem o valor **verdadeiro** ou **falso**.

Tipos de instruções de desvio

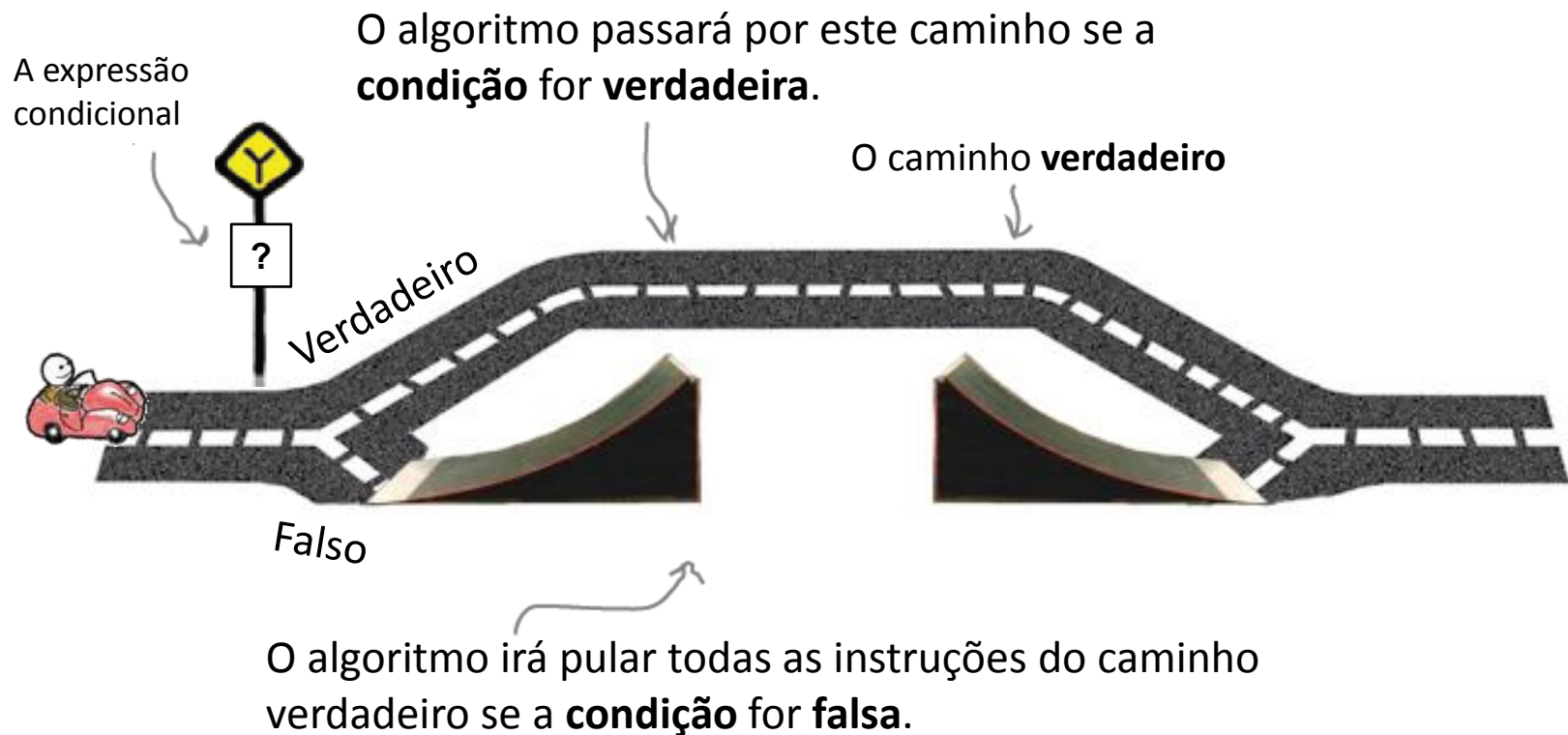
Na maioria das linguagens de programação estruturadas, existem **três** tipos de **estruturas** (ou instruções) que implementam **instruções de desvio**:

- **seleção simples**
- **seleção dupla e**
- **seleção múltipla.**

Estruturas de seleção

Seleção simples

A instrução de **seleção simples** permite que uma sequência de instruções seja **executada** caso a **condição** seja **verdadeira** ou que esta sequência **não seja executada** (ignora, ou pula as instruções) caso a **condição** seja **falsa**.



Estruturas de seleção



Seleção simples em Portugol

Em Portugol a instrução **SE..ENTÃO** é a instrução de **seleção simples** (ou condicional simples) pois ela **seleciona** ou **ignora** uma ação (ou um grupo de ações). A sintaxe da instrução **SE..ENTÃO** que usaremos é a seguinte:

```
...  
se <condição> entao  
    <sequência-de-comandos>;  
fim-se;  
...
```

{um conjunto de comandos é executado quando <condição> é avaliada como **VERDADEIRO**, caso contrário, o algoritmo prossegue para o próximo comando após **fim-se**.}

Condicional simples

Exemplo

```
...  
se ( media < 7.0 ) entao  
    leia( pFinal );  
    mediaFinal ← ( media + pFinal ) / 2;  
    escreva (mediaFinal);  
fim-se;  
...
```



Estruturas de seleção

```
se ( media < 7.0 ) entao
```

```
leia( pFinal );  
mediaFinal ← ( MEDIA + pFinal ) / 2;  
escreva (mediaFinal);
```

```
fim-se;
```

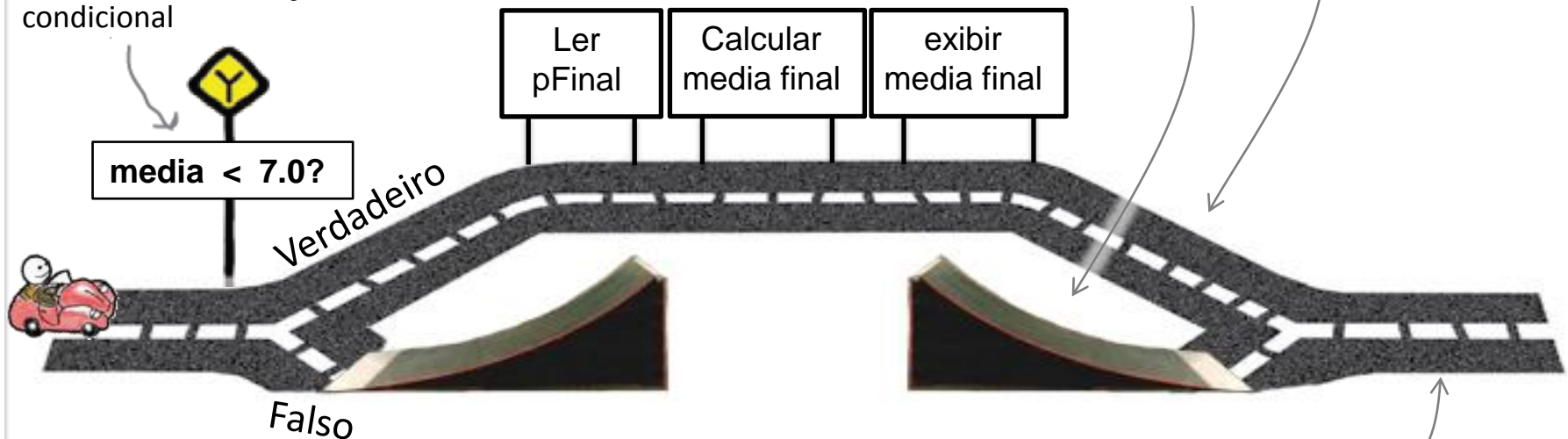
Expressão **condicional** (ou **condição**)

Caminho verdadeiro

A expressão condicional

Estes comandos só executam se o caminho **verdadeiro** for selecionado

Condição falsa: pular as instruções



Após a instrução de desvio condicional, o algoritmo continua a partir deste ponto

Estruturas de seleção



Assim, a nova versão do algoritmo poderia ser:

Temos mais dados para processar e, neste caso, Você precisará de mais variáveis.

```
1  {Calcular a média de um aluno.}
2  Algoritmo CALCULA_MEDIA;
3  var
4      P1, P2, MEDIA, mediaFinal, pFinal: real;
5  Inicio
6      leia (P1);
7      leia (P2);
8      MEDIA ← (P1 + P2) / 2;
9      escreva (MEDIA);
10     se ( media < 7.0 ) entao
11         leia ( pFinal );
12         mediaFinal ← ( MEDIA + pFinal ) / 2;
13         escreva (mediaFinal);
14     fim-se;
15 Fim.
```

O código no caminho verdadeiro é **identado** e escrito na linha logo abaixo da instrução condicional **se...entao**.

Estruturas de seleção



Versão do algoritmo para o Visualg:

```
algoritmo "media"  
// Função : Calculo da media de alunos  
// Autor : Bruno Vilela Oliveira  
// Data : 31/03/2012  
// Seção de Declarações  
var  
    p1, p2, pFinal, media, mediaFinal: real  
inicio  
// Seção de Comandos  
    leia( p1 )  
    leia( p2 )  
    media <- ( p1 + p2 ) / 2  
    escreval( media )  
  
    se ( media < 7.0 ) entao  
        leia( pFinal )  
        mediaFinal <- ( media + pFinal ) / 2  
        escreval( mediaFinal )  
    fimse  
fimalgoritmo
```

Seleção dupla

Em algumas situações, um algoritmo pode precisar escolher entre dois caminhos possíveis, **cada um com um conjunto de instruções específicas daquele caminho**.

A escolha de um caminho deve executar as instruções deste caminho e ignorar todas as instruções do outro caminho e vice-versa.

Isto não pode ser feito com a instrução de seleção simples.

Seleção dupla

Para introduzir uma nova instrução de seleção, vamos tentar resolver um problema: Ihe pediram para escrever um algoritmo simule um jogo de adivinhação.

Fixando um **valor** como a **resposta correta**, o jogo consiste de pedir que alguém tente adivinhá-lo. Em seguida o jogo deve responder se o jogador **venceu** (acertou) ou **perdeu** (errou).

O seguinte algoritmo é capaz de simular este jogo (Vamos considerar que o resultado correto seja 5):

Solicitar ao jogador que forneça um número
Se o número informado for igual a 5
exibir a mensagem: Você venceu!
Caso contrário,
exibir a mensagem: Você perdeu!

Selecionando um caminho entre dois possíveis

O algoritmo que simula esse jogo deve ser capaz de selecionar uma mensagem para exibir (“**Você venceu!**” ou “**Você perdeu!**”) de acordo com o palpite do jogador.

Dessa forma, existem **dois** caminhos possíveis a seguir no código:

- Exibir “**Você venceu!**” ou
- Exibir “**Você perdeu!**”

Para que seu algoritmo possa tomar esta decisão você precisa da instrução de **seleção dupla**, que também é uma **estrutura de seleção**, mas permite ao algoritmo escolher um entre dois caminhos possíveis em vez de passar por um caminho ou ignorá-lo (saltá-lo).

Estruturas de seleção

■ A Seleção dupla também é uma encruzilhada no código

Na instrução de seleção dupla seus algoritmos também escolhem o caminho a seguir usando uma **expressão condicional** (**verdadeira** ou **falsa**).

Se a **condição** é verdadeira, o algoritmo executa o código que está no caminho do **desvio verdadeiro**. Se a condição de desvio for falsa o código executado é o que está no caminho do **desvio falso**.



Estruturas de seleção



Notação da seleção dupla em Portugol

Em Portugol, a instrução **SE..ENTÃO...SENÃO** é uma instrução de seleção dupla (ou bicondicional). Ela **seleciona** um entre dois possíveis caminhos no código. Sua sintaxe é a seguinte:

```
...  
se <condição> entao  
    <sequência-de-comandos>;  
senao  
    < sequência-de-comandos>;  
fim-se;  
...
```

{um conjunto de comandos é executado apenas quando <condição> é avaliada como **VERDADEIRA**, e outro conjunto é executado somente quando <condição> assume valor **FALSO**. Após esta seleção, o algoritmo prossegue para o comando após **fim-se**.}

Exemplo

```
...  
se ( PALPITE = 5 ) entao  
    escreva ( 'Você venceu!' );  
senao  
    escreva ( 'Você perdeu!' );  
  
fim-se;  
...
```

Condicional dupla



Estruturas de seleção



Portanto, poderíamos escrever o algoritmo de adivinhação em Portugol da seguinte forma:

```
{simulação do jogo de adivinhação}
1  Algoritmo GUESS_GAME;
2  var
3      PALPITE: inteiro;
4  Inicio
5      escreva('Bem vindo!');
6      escreva('Adivinhe o número: ');
7      leia (PALPITE);
8      se ( PALPITE = 5 ) entao
9          escreva('Você venceu!');
10     senao
11         escreva('Você perdeu!');
12     fim-se;
13 Fim.
```

Estruturas de seleção

Neste exemplo, a expressão condicional é “**PALPITE = 5**”.

Este é um teste de igualdade e resultará em **verdadeiro** ou **falso**.

O código no caminho verdadeiro é **identado** e escrito na linha logo abaixo da instrução condicional **se...entao**. O código no caminho falso é **identado** e escrito na linha abaixo da palavra reservada **senao**.

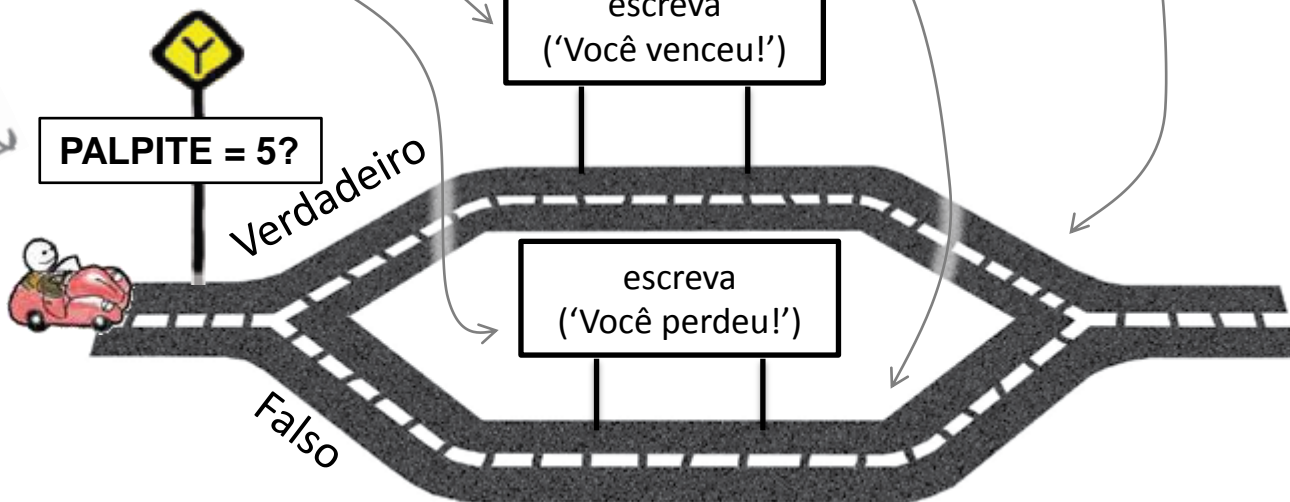
```
se ( PALPITE = 5 ) entao  
    escreva ( 'Você venceu!' ) ;
```

```
senao  
    escreva ( 'Você perdeu!' ) ;
```

```
fim-se ;
```

A expressão condicional

Estes comandos só executam se estiverem no caminho que o algoritmo selecionar.



Estruturas de seleção



Versão do algoritmo para o Visualg:

```
algoritmo "GuessGame"  
// Função : Simular o jogo de adivinhação de números  
// Autor : Bruno Vilela Oliveira  
// Data : 31/03/2012  
// Seção de Declarações  
var  
    palpite: inteiro  
inicio  
    // Seção de Comandos  
    escreval( "Bem vindo!" )  
    escreva( "Adivinhe o numero: " )  
    leia( palpite )  
    se ( palpite = 5 ) entao  
        escreva( "Você venceu!" )  
    senao  
        escreva( "Você perdeu!" )  
    fimse  
fimalgoritmo
```

Estruturas de seleção



Um dos usuários
do seu algoritmo.

Não entendi. Como posso
adivinhar o número correto?
Tudo que o algoritmo me diz
é que meu palpite está certo
ou errado. Vamos lá, me dê
uma ajuda!

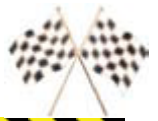
Você precisa melhorar o algoritmo do jogo para que ele dê mensagens mais informativas ao usuário.

Mas como ficarão os caminhos no algoritmo?

Aninhamento de instruções de controle

Como seria então, o algoritmo do jogo de adivinhação que é capaz de ajudar o usuário a chegar na resposta correta?

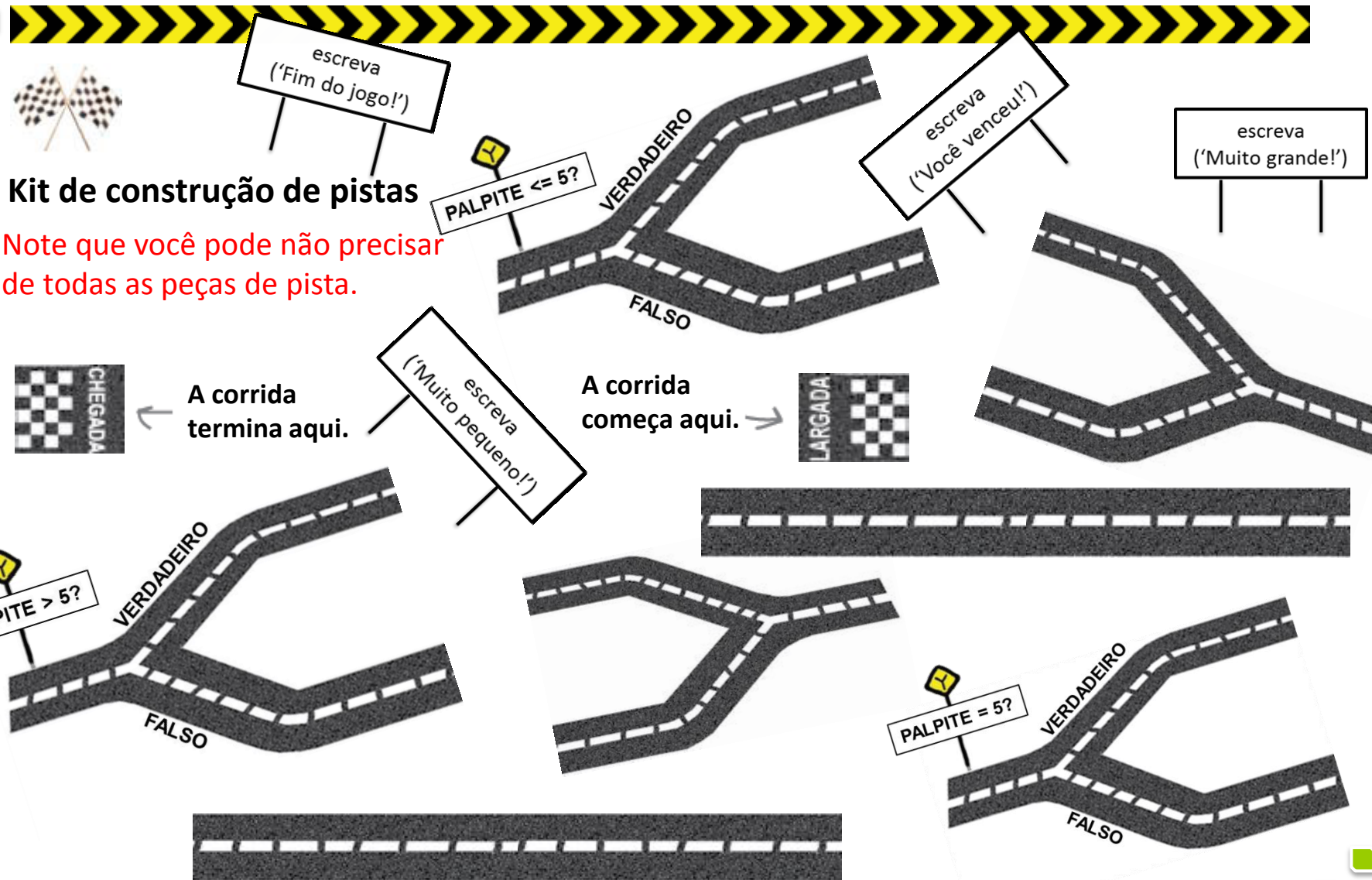
Você poderia escrever um algoritmo mais informativo que retorne mensagens como as dos exemplos na tabela abaixo:

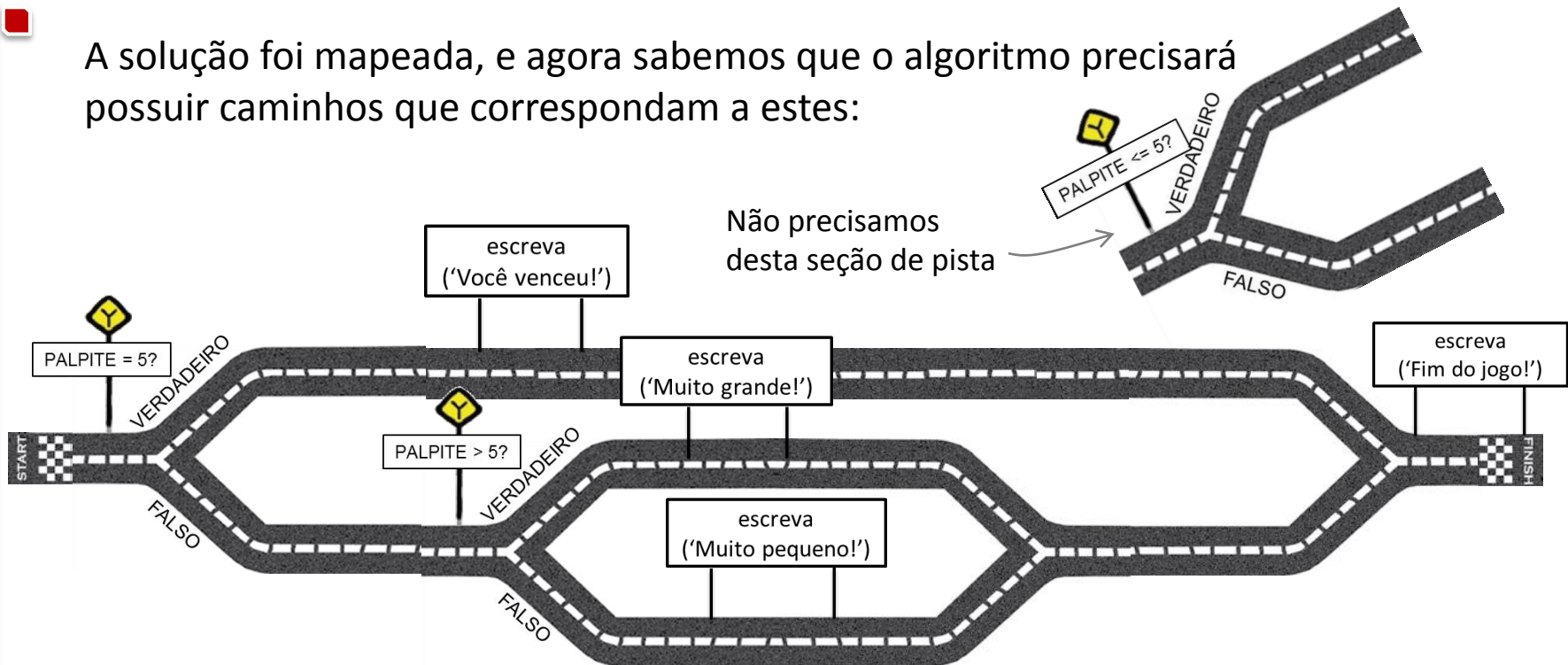


Número informado	Mensagem do algoritmo
3	Muito pequeno!
5	Você venceu!
7	Muito grande!
8	Muito grande!

A contagem regressiva começou no Grande Prêmio de Codigolândia. Os carros já chegaram e estão aquecendo os pneus no grid de largada. A corrida está prestes a começar. Você pode montar a pista de forma que ela dê um retorno adequado ao usuário quando ele fizer um palpite errado? Para isto, utilize as peças do nosso kit de montagem de pistas disponíveis no próximo slide.

Estruturas de seleção



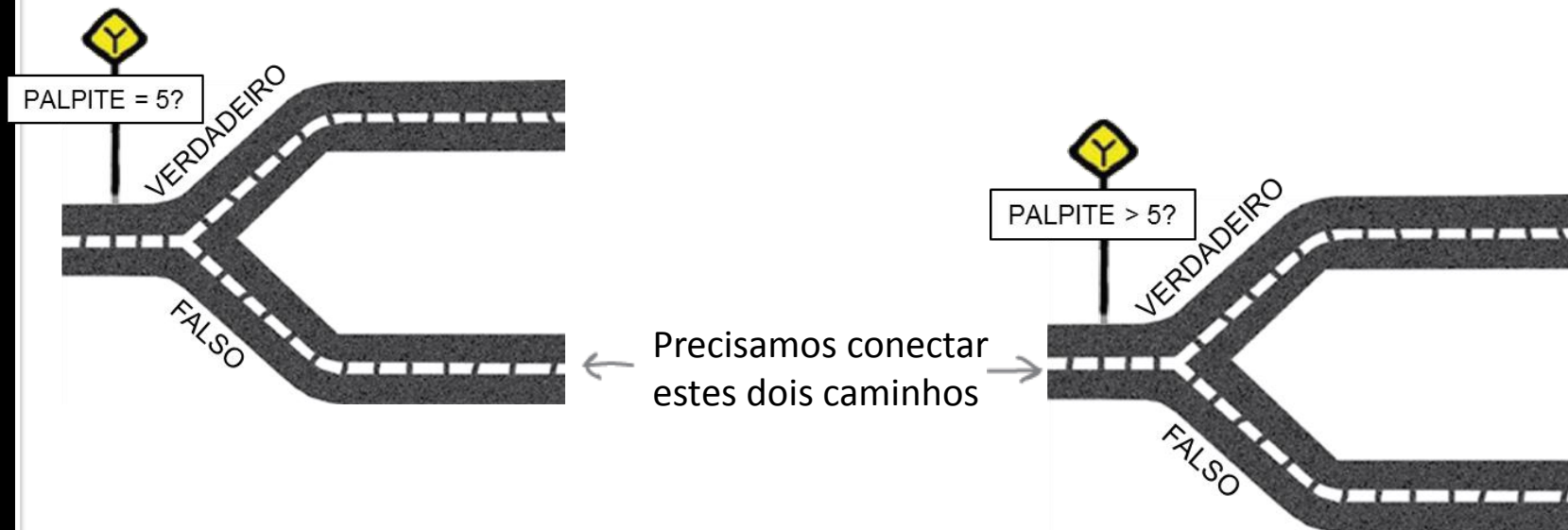


ACCEPTED MANUSCRIPT



Estruturas de seleção

No novo algoritmo, precisaremos conectar duas instruções de seleção.
Precisamos que a segunda seleção apareça no **caminho falso** da primeira.



Então como você pode conectar duas instruções de seleção em Portugal?

Aninhamento de instruções de controle

As instruções de controle ditam os caminhos que devem ser percorridos e a maneira que eles devem ser percorridos em um algoritmo.

As instruções em um caminho são executadas na sequência em que são alcançadas. Em alguns pontos pode ser necessário decidir entre tomar um caminho ou outro cada um com uma sequência de instruções (ou decidir entre pular um caminho ou passar por ele).

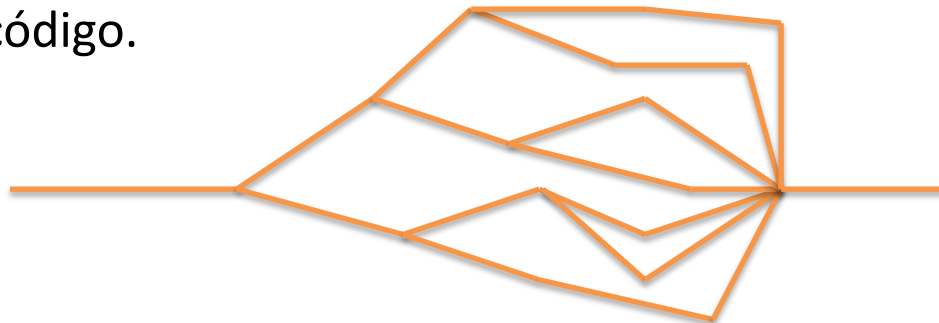
As estruturas de seleção permitem seus algoritmos fazerem isto.

Aninhamento de instruções de controle

Ao tomar um caminho através de uma instrução de desvio, nada impede que este caminho, além de instruções sequenciais possua outros pontos de desvio de código(uma bifurcação em uma rua pode te levar a uma rua que por sua vez também possua uma bifurcação em algum ponto).

Quando isto ocorre, diz-se que as estruturas estão aninhadas, isto é, uma instrução de seleção leva o algoritmo para uma sequência de código que também possui instruções de seleção.

E isso pode ocorrer de forma sucessiva criando uma grande quantidade de ramificações de caminhos de código.



Como criar seleções aninhadas?

Em uma instrução de seleção, as instruções que devem ser executadas no caminho verdadeiro devem estar entre as instruções **se...então** e **senao**. As instruções que devem ser executadas no caminho falso devem ficar entre a instrução **senao** e a cláusula **fim-se**.

Os elementos da programação estruturadas possuem palavras reservadas e/ou caracteres marcadores especiais para delimitar o início e o fim de um caminho dentro de determinada estrutura.

Como criar seleções aninhadas?

Vamos considerar o seguinte trecho de código de um algoritmo diferente : algo que decidirá se você pode dirigir até o centro da cidade (em uma cidade grande).

```
se ( COMBUSTIVEL > 3 ) entao  
    escreva('Tudo certo!');  
    escreva('Você pode dirigir até o centro.');
```

Este é o caminho verdadeiro.

```
    senao  
        escreva('Desculpe!');  
        escreva('Você não possui combustível suficiente!');
```

Este é o caminho falso.

```
    fim-se;  
    escreva('Qual é a próxima?');
```

Este comando não está no caminho falso, então, sempre será executado.

Como criar seleções aninhadas?

Para conectar as instruções de seleção, basta colocar uma delas como uma das instruções em um dos caminhos da outra seleção (caminho verdadeiro ou falso).

A primeira instrução de seleção.

```
se ( COMBUSTIVEL > 3 ) entao  
    escreva('Tudo certo!');  
    escreva('Você pode dirigir até o centro.');
```

Esta segunda instrução de seleção **se** está conectada ao caminho “falso” da primeira instrução de seleção

```
    senao  
        se ( DINHEIRO > 10 ) entao  
            escreva('Você precisa reabastecer!');  
            senao  
                escreva('Melhor você ficar em casa.');
```

```
    fim-se  
fim-se;  
    escreva('Qual é a próxima?');
```

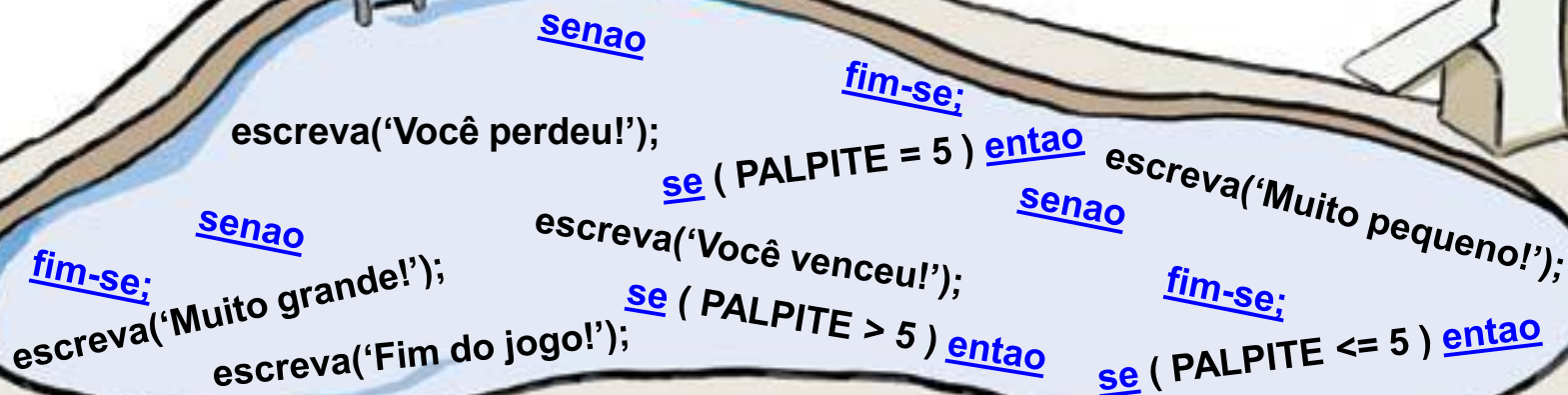
Estruturas de seleção

Quebra-cabeça da piscina.



Sua tarefa é pegar os fragmentos de código em Português que flutuam na piscina e organizá-los (um por linha) para criar um algoritmo. Você não pode usar o mesmo fragmento de código mais de uma vez e também não precisará usar todos eles. Seu objetivo é completar o algoritmo do jogo de adivinhação.

```
...  
escreva('Bem vindo!');  
escreva('Adivinhe o número: ');  
leia (PALPITE);  
...
```



senao
escreva('Você perdeu!');

fim-se;
se (PALPITE = 5) entao escreva('Muito pequeno!');

senao
escreva('Você venceu!');

fim-se;
se (PALPITE > 5) entao escreva('Muito grande!');

senao
escreva('Fim do jogo!');

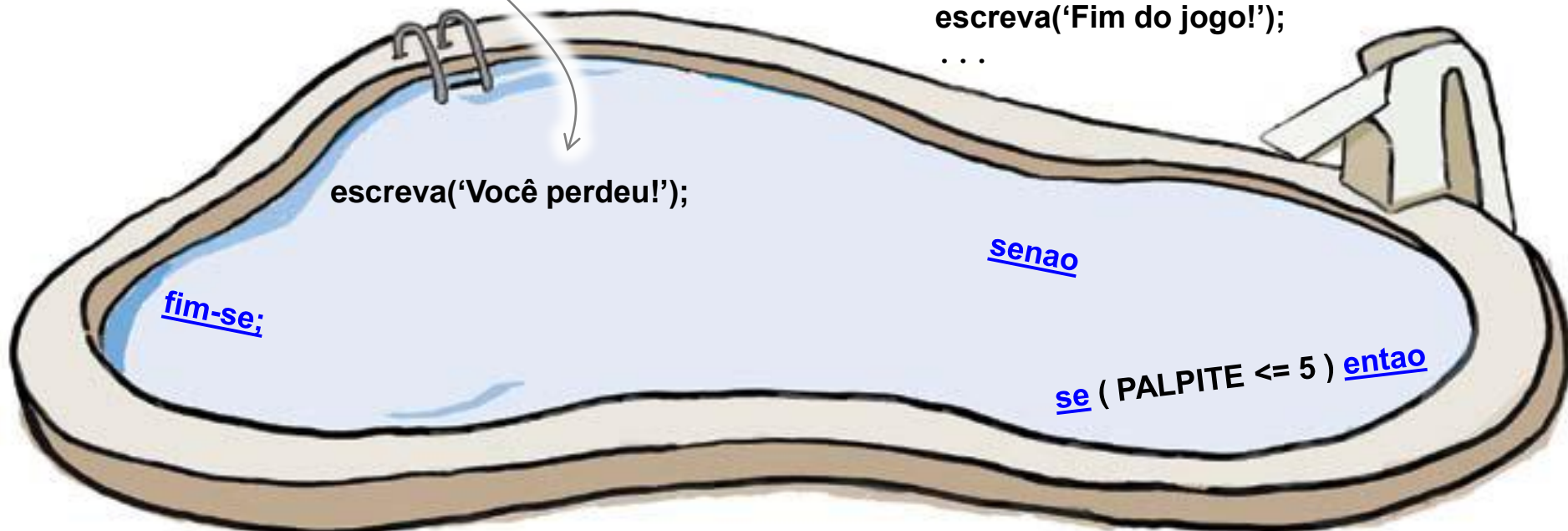
fim-se;
se (PALPITE <= 5) entao

Estruturas de seleção

Quebra-cabeça da piscina Solução.



Este fragmento de código da primeira versão do jogo não é mais necessário.



...

escreva('Bem vindo!');

escreva('Adivinhe o número: ');

leia (PALPITE);

se (PALPITE = 5) entao

escreva('Você venceu!');

senao

se (PALPITE > 5) entao

escreva('Muito grande!');

senao

escreva('Muito pequeno!');

fim-se;

fim-se;

escreva('Fim do jogo!');

...

Estruturas de seleção



Quebra-cabeça da piscina Solução.



Algoritmo 'Adivinhacao';
var

PALPITE: inteiro;

Início

escreva('Bem vindo!');

escreva('Adivinhe o número: ');

leia (PALPITE);

se (PALPITE = 5) entao
 escreva('Você venceu!');

senao

se (PALPITE > 5) entao
 escreva('Muito grande!');

senao

 escreva('Muito pequeno!');

fim-se;

fim-se;

escreva('Fim do jogo!');

Fim.



Seleção múltipla



- ❖ A instrução de **seleção dupla** representa adequadamente processos **binários** de **tomada de decisão** (um desvio com somente duas opções).
- ❖ Nas situações que uma **expressão condicional** pode assumir **mais que dois valores**, e para cada possível valor (ou faixa de valores) o algoritmo tem que assumir **um fluxo de ações específico**, o uso da estrutura condicional pode tornar o algoritmo confuso ou muito grande.



Seleção múltipla



- ❖ Seja um algoritmo que deve analisar se um caractere é ou não uma vogal, a especificação das instruções desse algoritmo poderia ser a seguinte:

...

```
se (simbolo = 'a') ou (simbolo = 'e') ou (simbolo = 'i') ou (simbolo = 'o') ou (simbolo = 'u') entao  
    escreva ( 'vogal' ) ;  
senao  
    escreva ( 'consoante' ) ;  
fim-se ; ...
```

- ❖ Para este exemplo, a leitura do algoritmo não é tão complicada.



Seleção múltipla



- ❖ Entretanto, imagine uma situação em que deve-se verificar se o valor de uma variável do tipo caractere é uma **letra**, um **dígito** numérico, um **símbolo de operador** ou um **caractere especial**?
- ❖ Seria bastante trabalhoso listar todas as condições a serem verificadas.
 - Por exemplo, o conjunto de caracteres do padrão ASCII possui 256 caracteres.
- ❖ Há quatro possíveis fluxos de ações a seguir no algoritmo:
 - Um para cada tipo de valor verificado:
 - letra, dígito, operador e caractere especial.

Seleção múltipla



- ❖ Uma estrutura de **seleção múltipla** permite especificar de forma mais natural as situações nas quais um algoritmo deve escolher um fluxo específico de ações entre vários possíveis, com base em no valor resultante de uma **expressão**, quando esta pode ter como resultado muito mais do que **dois valores**.
- ❖ **Cada fluxo** é associado a **um** dos vários possíveis valores resultantes da denominada **expressão de seleção**.

Seleção múltipla



❖ A estrutura tem a seguinte notação:

```
...  
escolha <expressão-de-seleção>  
    caso <valor11>, <valor12>, ... <valor1n>:<sequência-de-instruções>;  
    caso <valor21>, <valor22>, ... <valor2n>:<sequência-de-instruções>;  
    caso <valor31>, <valor32>, ... <valor3n>:<sequência-de-instruções>;  
    ...  
    outrocaso:<sequência-de-instruções-extra>;  
fim-escolha;  
...
```

Obs:

- **expressão-de-seleção** pode resultar em um valor do tipo inteiro, caractere ou booleano. (em algumas linguagens o tipo string é aceito, não é o caso do Pascal).
- A cláusula “**outrocaso**” é opcional.
- Cada cláusula “**caso**” é seguida de uma constante e/ou uma lista de constantes e/ou ou uma faixa de valores constantes do mesmo tipo da expressão de seleção.

❖ Seleção múltipla – Exemplo 1:

- considerando “**simbolo**” uma variável do tipo caractere:

```
...  
escolha simbolo  
    caso 'A' .. 'Z', 'a' .. 'z': escreva('letra');  
    caso '0' .. '9': escreva('numero');  
    caso '+', '-', '*', '/': escreva('Operador');  
    outrocaso: escreva('caractere especial');  
fim-escolha;  
...
```

→ Por que não usar
"caso 0..9: "
em vez de
"caso '0'..'9':" ???

❖ Seleção múltipla – Exemplo 2:

Algoritmo Times;
 var time: cadeia;

Início

escreva ('Entre com o nome de um time de futebol: ');
 leia (time);

escolha time

caso 'Flamengo', 'Fluminense', 'Vasco', 'Botafogo':

escreva ('É um time carioca.');

caso 'São Paulo', 'Palmeiras', 'Santos', 'Corinthians':

escreva ('É um time paulista.');

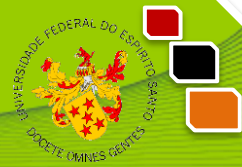
outrocaso

escreva ('É de outro estado.');

fim-escolha;

Fim.


Iterações (ou repetição)



Seu jogo de adivinhação funcionou perfeitamente!

Mas os usuário estão mais felizes?

Iterações (ou repetição)



Porque é que eu tenho que ficar executando novamente o algoritmo? Quer dizer que eu só tenho direito a um palpite?

Os usuários ainda não gostam do algoritmo.

O algoritmo funciona, e agora gera *feedback* extra, mas há um problema. Se o usuário quer dar outro palpite, ele tem que executar o algoritmo novamente (do início). Na realidade eles querem que o algoritmo continue lhes pedindo outros palpites até que eles finalmente deem a resposta correta. Você consegue ver qual é o problema?

Como podemos escrever um algoritmo que faz algo repetidamente? Devemos simplesmente copiar e colar o código no fim do algoritmo? Isto iria garantir que a pergunta do palpite seja feita ao usuário duas vezes. Mas e se ele precisar de três tentativas? Ou 4? ou 10000 tentativas? E quando ele der a resposta correta? (O jogo continuaria lhe perguntando?)

O jogo de adivinhação precisa ser capaz de executar alguma parte do código de repetidamente.

Iterações (ou repetição)

Não seria um sonho se houvesse um
jeito de fazer com que um trecho de
código seja executado várias vezes?
Mas acho que é só uma fantasia...



Iterações (ou repetição)



Os Loops (laços) permitem você executar o mesmo trecho de código repetidamente.

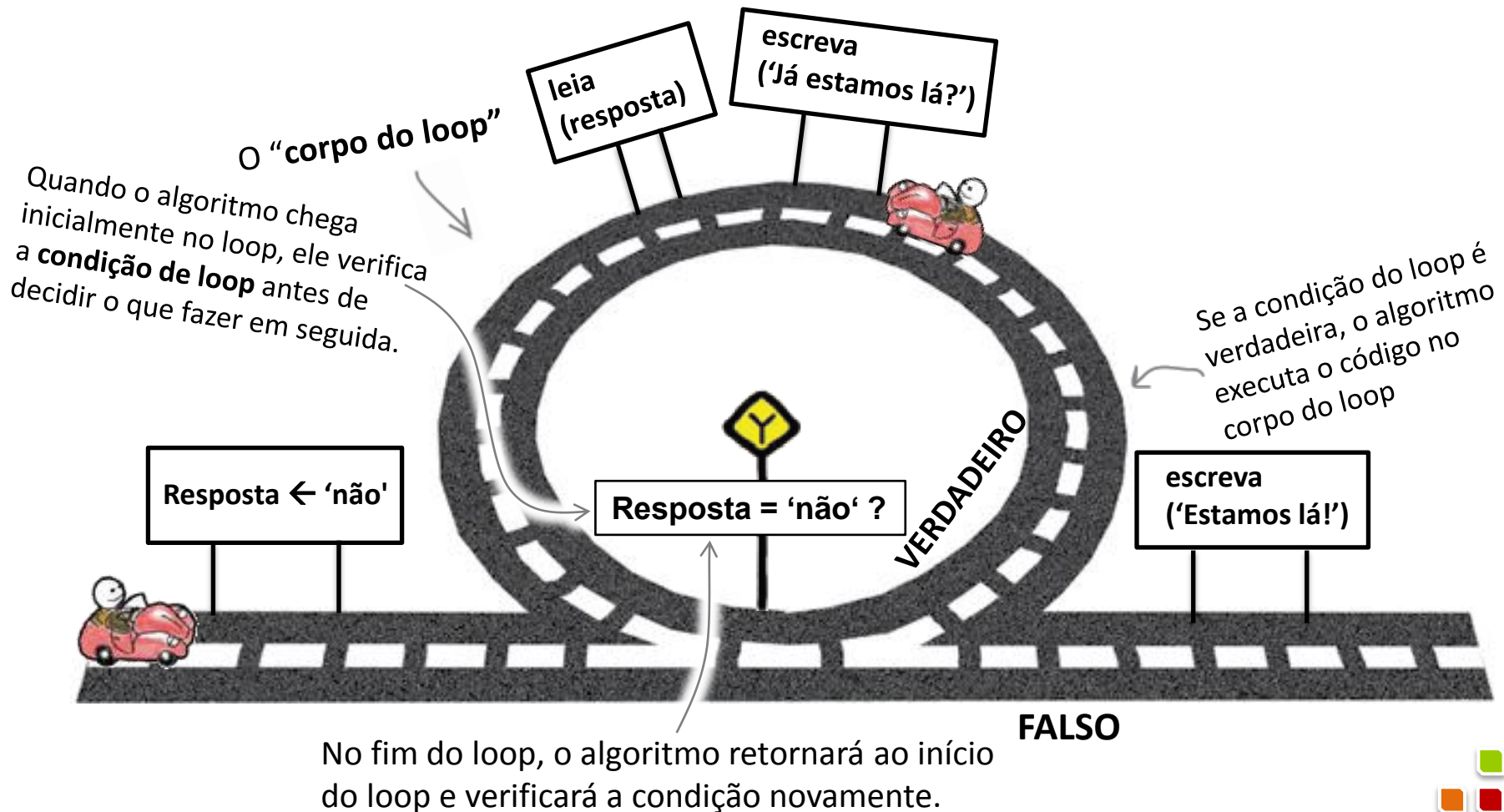
Programas frequentemente precisam continuar executando o mesmo trecho de código várias vezes. Além da estrutura de seleção, as linguagens de programação estruturadas também oferecem a **estrutura de iteração**, que é frequentemente chamada de **loop**, **laço**, **repetição** etc.

Os laços são como desvios (seleções). Tal como as instruções de seleção, os laços possuem uma expressão condicional (**condição**) que é **verdadeira** ou **falsa**. Assim como a parte **se..então** de uma seleção, se a **condição** do laço é verdadeira, então o laço irá executar um dado trecho de código. Nas instruções de seleção, este trecho de código é chamado de **corpo** (ou bloco em algumas linguagens de programação). Em um laço, é chamado de **corpo do laço** (ou bloco do laço ou de iteração).



Iterações (ou repetição)

Os Loops (laços) permitem você executar o mesmo trecho de código repetidamente.



Iterações (ou repetição)



Os Loops (laços) permitem você executar o mesmo trecho de código repetidamente.

A grande diferença entre um loop e uma instrução de seleção (desvio) é o número de vezes que eles executam o código associado a eles. Uma instrução de seleção executa o código somente uma vez. Mas um loop executará o corpo do loop, então irá verificar a condição de loop novamente, e se ela ainda for verdadeira, executará o corpo do loop de novo. E de novo. E de novo.

De fato, ele continuará executando o corpo do loop até a condição do loop se tornar **falsa**.

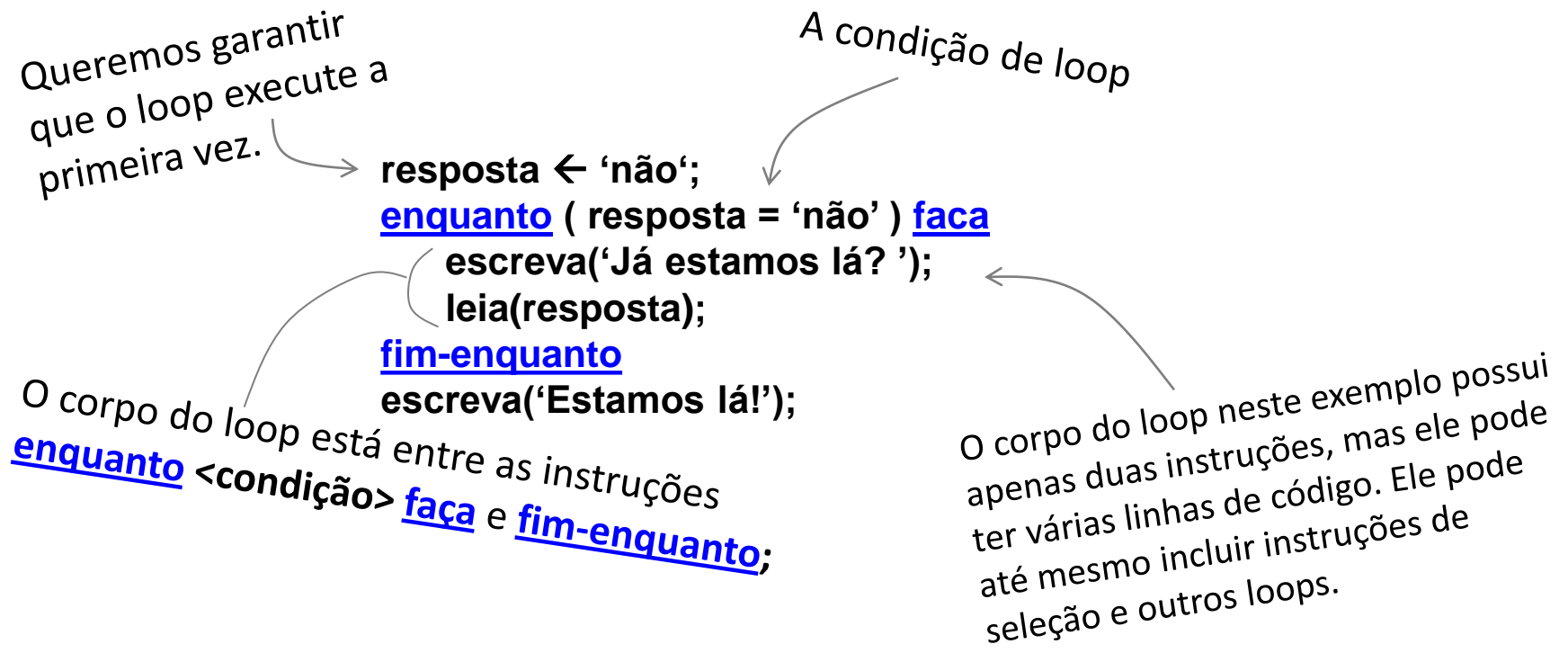


Iterações (ou repetição)



O loop “enquanto..faça”

As linguagens de programação permitem criar **loops** de várias maneiras, mas uma das mais simples é o loop **enquanto..faça**. Veja este exemplo:



Esta é a sintaxe do loop **enquanto..faça** em Portugol. O código continua perguntando “Já estamos lá?” até que o usuário digite algo diferente de “não”.

Iterações (ou repetição)



O loop “enquanto..faça”

```
resposta ← ‘não’;  
enquanto ( resposta = ‘não’ ) faca  
    escreva(‘Já estamos lá? ’);  
    leia(resposta);  
fim-enquanto  
    escreva(‘Estamos lá!’);
```

Quando executado, ocorre algo semelhante a isto:

```
Já estamos lá? não  
Já estamos lá? não  
Já estamos lá? não  
Já estamos lá? não  
Já estamos lá? sim, finalmente!  
Estamos lá!  
*** Fim da execução.  
*** Feche esta janela para retornar ao Visualg.
```

Assim que digitamos algo diferente de “não” o loop encerra.

Iterações (ou repetição)



O loop “enquanto..faça”

Versão completa do algoritmo para o visualg:

algoritmo "Loops"

// Função : Apresentar o conceito de instrucoes de iteracao

// Autor : Bruno Vilela Oliveira

// Data : 01/04/2012

// Seção de Declarações

var

resposta: caractere

inicio

// Seção de Comandos

resposta <- "não"

enquanto (resposta = "não ") **faca**

 escreva("Já estamos lá ")

 leia(resposta)

fimenquanto

 escreva("Estamos lá! ")

fimalgoritmo



Iterações (ou repetição)



O loop “enquanto..faça”

Você percebeu que tivemos que inicializar o valor da variável “**resposta**” para algo antes de iniciar o loop? Isto foi importante, pois se a variável **resposta** já não tivesse o valor inicial “**não**” a condição de loop já teria sido **falsa** e o código no corpo do loop nunca teria sido executado.

Tenha isto em mente. Pode ser útil no próximo exercício...



Iterações (ou repetição)



Um “grande” exercício...

Agora é a hora de aplicar sua experiência de programação. Fique atento: este exercício é um pouco manhoso. Você precisa reescrever seu programa do jogo de adivinhação de forma que ele continue executando até que o usuário adivinhe o número correto. Você precisará de tudo que aprendeu até agora. Precisarás elaborar as **condições** de cada **instrução de seleção** e dos **loops** que forem necessários. Lembre-se: o programa precisa continuar pedindo uma resposta ao usuário enquanto o palpite atual for errado.

Dica: quando precisar testar se duas coisas possuem valores diferentes, utilize o operador `<>`

← Este é o operador de desigualdade.

Escreva o código de seu algoritmo no caderno ou no Visualg.



Iterações (ou repetição)



Para você se lembrar da solução anterior

Algoritmo 'Adivinhacao';

var

PALPITE: inteiro;

Início

escreva('Bem vindo!');

escreva('Adivinhe o número: ');

leia (PALPITE);

se (PALPITE = 5) entao

escreva('Você venceu!');

senao

se (PALPITE > 5) entao

escreva('Muito grande!');

senao

escreva('Muito pequeno!');

fim-se;

fim-se;

escreva('Fim do jogo!');

Fim.

Solução



Visualg

```
algoritmo "JogoDeAdivinhacao"
// Função : O jogo de adivinhacao com iteracoes
// Autor : Bruno Vilela Oliveira
// Data : 01/04/2012
// Seção de Declarações
var
    palpite: inteiro
inicio
// Seção de Comandos
escreval( "Bem vindo!" )
palpite <- 0
enquanto ( palpite <> 5 ) faca
    escreva( "Adivinhe o número: " )
    leia( palpite )
    se ( palpite = 5 ) entao
        escreval( "Você venceu!" )
    senao
        se ( palpite > 5 ) entao
            escreval( "Muito grande" )
        senao
            escreval( "Muito pequeno" )
        fimse
    fimse
fimenquanto
escreva( "Fim do jogo!" )
finalgoritmo
```

Portugol

```
algoritmo 'JogoDeAdivinhacao';
{ Função : O jogo de adivinhação com iterações
  * Autor : Bruno Vilela Oliveira
  * Data : 01/04/2012
  * Seção de Declarações }
var
    palpite: inteiro;
inicio
{ Seção de Comandos }
escreva( 'Bem vindo!' );
palpite ← 0;
enquanto ( palpite <> 5 ) faca
    escreva( 'Adivinhe o número: ' );
    leia( palpite );
    se ( palpite = 5 ) entao
        escreva( 'Você venceu!' );
    senao
        se ( palpite > 5 ) entao
            escreva( 'Muito grande' );
        senao
            escreva( 'Muito pequeno' );
        fim-se;
    fim-se;
fim-enquanto;
escreva( 'Fim do jogo!' );
Fim.
```

Iterações (ou repetição)



Melhorando seu jogo...

Seu jogo ainda tem uma limitação: depois que o usuário descobrir a resposta correta, ele não vai mais achar divertido jogá-lo outra vez.

A maioria das linguagens de programação oferecem recursos para geração de valores aleatórios.

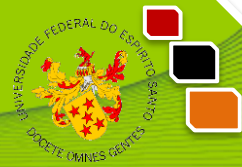
Se em vez de considerar que a resposta correta sempre será 5 você iniciar resposta correta aleatoriamente toda vez que o algoritmo iniciar, seu jogo certamente será mais desafiador e interessante.

Para isto, vamos considerar a seguinte função : **aleatorio(x, y)**

Sendo **x** e **y** constantes do tipo inteiro ou real, com **x < y** a instrução **aleatorio(x, y)** gera e retorna um valor aleatório entre x e y (incluindo x e y).



Iterações (ou repetição)



Código pré-cozido.

Melhorando seu jogo...

Como usar o comando **aleatorio(x, y)**? Exemplo:

```
numero_aleatorio ← aleatorio( 1, 10 );
```

O comando **aleatorio**(1, 10) retornará um valor entre 1 e 10.
O resultado será armazenado na variável **numero_aleatorio**
que está à esquerda do operador de atribuição.
O único jeito de saber qual foi o valor retornado é verificar o
valor armazenado na variável **numero_aleatorio**.



algoritmo 'JogoDeAdivinhacao';

{ Função : O jogo de adivinhação com iterações e geração de valores aleatórios

* Autor : Bruno Vilela Oliveira

* Data : 01/04/2012

* Seção de Declarações }

var

palpite, secreto: inteiro;

inicio

{ Seção de Comandos }

secreto ← aleatorio (1, 10);

escreva('Bem vindo!');

palpite ← 0;

enquanto (palpite <> secreto) **faca**

escreva('Adivinhe o número: ');

leia(palpite);

se (palpite = secreto) **entao**

escreva('Você venceu!');

senao

se (palpite > secreto) **entao**

escreva('Muito grande');

senao

escreva('Muito pequeno');

fim-se;

fim-se;

fim-enquanto;

escreva('Fim do jogo!');

Fim.

Esta é linha que cria o número aleatório.

Agora, em vez de verificar se a resposta do usuário é 5, comparamos ela com o valor aleatório, que está guardado na variável "secreto".

algoritmo "JogoDeAdivinhacao"

// Função : O jogo de adivinhação com iterações e geração de valores aleatórios

// Autor : Bruno Vilela Oliveira

// Data : 01/04/2012

// Seção de Declarações

var

palpite, secreto: inteiro

inicio

// Seção de Comandos

aleatorio 1, 10

leia (secreto)

aleatorio off

limpatela

escreval("Bem vindo!")

palpite <- 0

enquanto (palpite <> secreto) faca

escreva("Adivinhe o número: ")

leia(palpite)

se (palpite = secreto) entao

escreval("Você venceu!")

senaose (palpite > secreto) entao

escreval("Muito grande")

senao

escreval("Muito pequeno")

fimsefimsefimenquanto

escreval("Fim do jogo!")

Fimalgoritmo

Esta é linha que inicia a geração de números aleatórios. (entre 1 e 10 inclusive)

Agora, em vez de verificar se a resposta do usuário é 5, comparamos ela com o valor aleatório, que está guardado na variável "secreto".

No visualg, a geração de valores aleatórios é um pouco diferente. O comando **aleatorio** faz com que todos os comandos de entrada (leia()) atribuam automaticamente um valor aleatório às variáveis que estão sendo lidas. Para limitar o trecho de código em que isto é feito, o comando **aleatorio off** termina este mecanismo no a partir de onde for colocado no algoritmo. Além disso todo comando leia() executado entre aleatorio e aleatorio off exibe automaticamente na tela os valores aleatórios gerados.

Iterações (ou repetição)



Então, o que acontece quando você executa o novo algoritmo?

```
Bem vindo!  
Adivinhe o número: 3  
Muito pequeno  
Adivinhe o número: 9  
Muito grande  
Adivinhe o número: 7  
Muito grande  
Adivinhe o número: 4  
Você venceu!  
Fim do jogo!  
*** Fim da execução.  
*** Feche esta janela para retornar ao Visualg.
```

O algoritmo continua pedindo um novo palpite enquanto você estiver dando respostas erradas.

A resposta correta agora é um número aleatório e poderá ser diferente cada vez que você jogar o jogo.

O algoritmo para quando você dá a resposta correta.

Iterações (ou repetição)

Este jogo é muito maneiro.
Não importa quantas vezes eu
jogo, ainda tenho que pensar para
descobrir a resposta correta!



Seus usuários adoram o algoritmo.

E você o criou por si só. **Analisando** cuidadosamente o problema, decidindo qual deveria ser o *feedback*, e elaborando a lógica complexa dos **loops** e **seleções** você criou algo que realmente “detona”!

Parabéns. Você está no caminho de se tornar um verdadeiro mestre do código.

Iterações (ou repetição)



Outras formas de repetir o código

Conforme foi comentado, diferentes estratégias podem ser usadas para se repetir um determinado código. Além da repetição **enquanto..faça** é comum o uso de instruções de repetição com algumas características diferentes.

No loop (ou laço) **enquanto..faça** o teste da condição de loop é feito antes de entrar no corpo do loop. Consequentemente, se ao chegar em um loop **enquanto..faça** a condição de loop já estiver **falsa**, o corpo do loop não será executado nem uma vez.

Esse tipo de loop, que testa a condição antes de executar o corpo do loop também é conhecido como iteração com **teste de condição no início**.



Iterações (ou repetição)



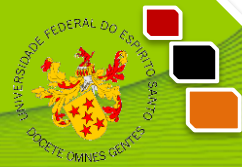
Outras formas de repetir o código

Existem instruções de repetição que permitem executar o corpo do loop antes de se testar a condição. Dessa forma, independente da condição de loop ser verdadeira, ao chegar em uma instrução desse tipo, o corpo do loop é executado para somente depois se verificar a condição de repetição, que caso (verdadeira, ou falsa) fará com que o corpo do loop seja executado outras vezes ou não.

Esse tipo de loop, que testa a condição após executar o corpo do loop também é conhecido como iteração com **teste de condição no fim**.



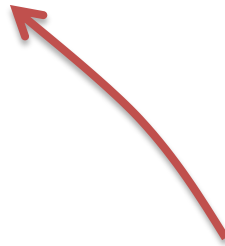
Iterações (ou repetição)



Outras formas de repetir o código

No Portugal (semelhando ao Pascal) o **loop** (laço ou repetição) com teste no fim é feita com o uso da instrução **repita..até**.

Ela possui uma diferença em relação ao **loop enquanto..faça**: o corpo do loop **continuará sendo executado até que a condição de loop se torne verdadeira** (em outras palavras, **repetirá o corpo do loop** enquanto a condição de loop estiver valendo **falso**).



Atenção



Iterações (ou repetição)



❖ Com teste de condição no início:

Enquanto-faça (controle a priori):

```
enquanto <condição> faca  
    <sequência-de-instruções>;  
fim-enquanto;
```

❖ Com teste de condição no fim:

Repita-até (controle a posteriori):

```
repita  
    <sequência-de-instruções>;  
ate <condição>;
```

Iterações (ou repetição)

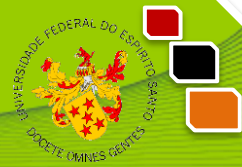


“Repita-Até” – Exemplo:

```
...
secreto ← aleatorio(1, 10);
repita
    escreva("Adivinhe o número: ");
    leia(palpite);
    se( palpite = secreto ) entao
        escreva( "Você venceu!" );
    senao
        se( palpite > secreto ) entao
            escreva( "Muito grande!" );
        senao
            escreva( "Muito pequeno!" );
        fim-se;
    fim-se;
ate palpite = secreto;
escreva( "Fim de jogo!" );
...
```

Além da instrução de repetição, o que mais mudou neste algoritmo?

Iterações (ou repetição)

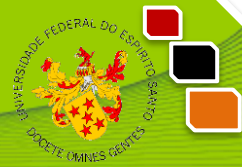


❖ “Enquanto-Faça” – Exemplo 1:

```
...  
leia(dia) ;  
enquanto (dia < 1) OU (dia > 31) faca  
    leia(dia) ;  
fim-enquanto ;  
...
```



Iterações (ou repetição)



❖ “Repita-Até” – Exemplo 1:

```
...  
repita  
  leia(dia) ;  
ate(dia >= 1) E (dia <= 31) ;  
...
```



Iterações (ou repetição)



❖ “Enquanto-Faça” – Exemplo 2:

```
...  
leia(numeroalunos) ;  
contador ← 1;  
enquanto contador <= numeroalunos faca  
    leia(nota1,nota2) ;  
    media ← (nota1 + nota2) / 2;  
    escreva(media) ;  
    contador ← contador + 1;  
fim-enquanto ;  
...
```



Iterações (ou repetição)



❖ “Repita-Até” – Exemplo 2:

```
...  
leia(numeroalunos) ;  
contador ← 1;  
se numeroalunos >= contador entao  
  repita  
    leia(nota1,nota2) ;  
    media ← (nota1 + nota2) / 2;  
    escreva(media) ;  
    contador ← contador + 1;  
  ate contador > numeroalunos;  
fim-se;  
...
```



❖ “Enquanto-Faça” – Exemplo 3:

- Escrever os número de 1 a 100:

Algoritmo EXEMPLO_PRATICO; *{escrever os números inteiros de 1 a 100}*

var

N: inteiro;

Inicio

N \leftarrow 1;

enquanto N \leq 100 faca

escreva (N);

N \leftarrow N + 1;

fim-enquanto;

Fim.

❖ “Repita-Até” – Exemplo 3:

- Escrever os número de 1 a 100:

Algoritmo EXEMPLO_PRATICO; *{escrever os números inteiros de 1 a 100}*

var

N: inteiro;

Inicio

N \leftarrow 1;

repita

escreva (N) ;

N \leftarrow N + 1;

ate N > 100;

Fim.

❖ “Enquanto-Faça” – Exemplo 4:

Algoritmo F_ou_M;

{permite o fluxo seguir apenas se forem pressionadas as teclas f ou m}

var

c: caractere;

Inicio

leia(c);

enquanto (c <> 'F') e (c <> 'M') faca

leia(c);

fim-enquanto;

se c = 'F' entao

escreva('Feminino');

senao

escreva('Masculino');

fim-se;

Fim.

❖ “Repita-Até” – Exemplo 4:

Algoritmo F_ou_M;

*{permite a execução prosseguir apenas se forem pressionadas as teclas f ou m}
{Note que, neste caso, utilizando a estrutura “repita-até” o código não necessita do comando leia(c) antes do início da iteração – o teste é no fim}*

var

c: caractere;

Início

repita

leia(c);

ate (c = 'F') **ou** (c = 'M');

se c = 'F' entao

escreva ('Feminino');

senao

escreva ('Masculino');

fim-se;

Fim.

O uso da estrutura **enquanto-faça** e da **repita-até** dependerá do tipo de iteração que será especificada. Quando se deve analisar uma condição antes de executar qualquer instrução da iteração é mais conveniente, mas não obrigatório, usar **enquanto-faça**. Quando as instruções que serão repetidas devem executar ao menos uma vez antes de se verificar a condição de continuidade da iteração, adotar a estrutura **repita-até** permite especificar a lógica da repetição de maneira mais natural.

Notação de diagrama de blocos



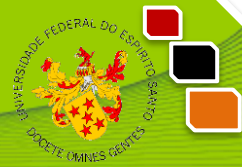
Outras formas de se especificar algoritmos

Você já sabe que existem notações gráficas para a especificação de algoritmos. Uma delas utiliza fluxogramas para especificar as sequências de ações e (entre outros nomes) é conhecida como diagrama de blocos.

A seguir serão apresentados os principais elementos da programação estruturada numa notação diagramática.



Diagrama de blocos



❖ Os principais elementos representados são:

- Sequências
- Decisões
- Iterações
- Entradas e saídas
- Processamento



Diagrama de blocos



❖ Elementos básicos de um fluxograma (notação)

Terminal



Indica o INÍCIO ou FIM de um algoritmo.

Entrada



Indica entrada de dados. [Exemplo: Digite a nota da prova 1.](#)

Processamento



Indica que uma ação (processamento) deve ser executada.
[Exemplo: Calculo da soma de dois números.](#)



Diagrama de blocos



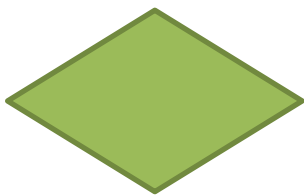
❖ Elementos básicos de um fluxograma (notação)

Saída



Mostra informações ou resultados. **Exemplo:**
Mostre o resultado do calculo.

Condição



Símbolo de decisão ou condição - indica que uma decisão deve ser tomada, verificando-se a validade de uma condição. **Exemplo:** **Se a média for inferior a 7,0 então o aluno está reprovado, senão ele está aprovado.**

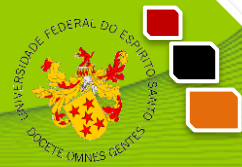
Sequencia



Indica o sentido do fluxo.



Diagrama de blocos



❖ Exemplo - Entrada

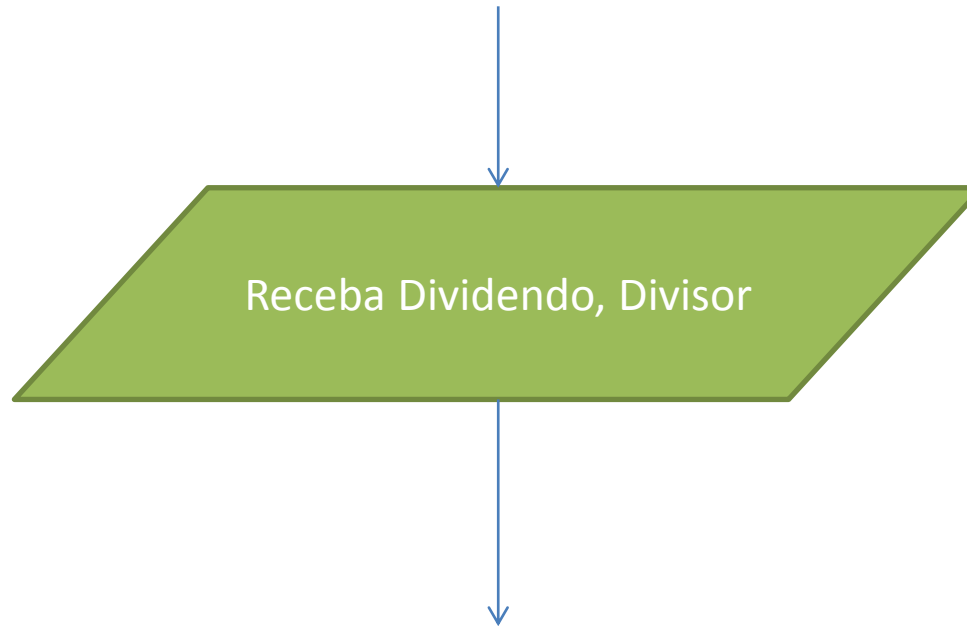


Diagrama de blocos



❖ Exemplo - Saída

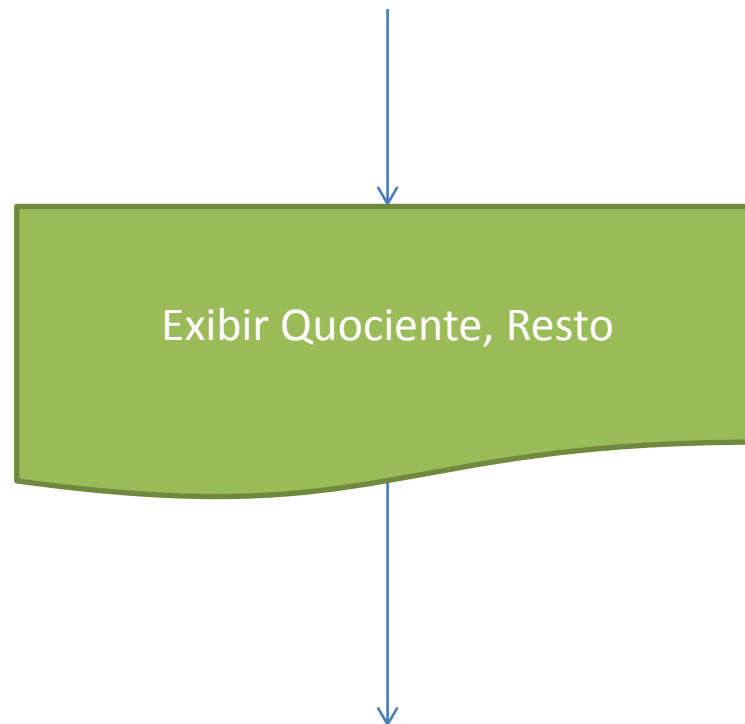


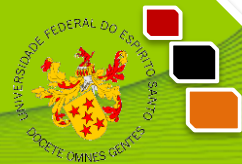
Diagrama de blocos



❖ Exemplo - Processamento

Quociente = Quociente + 1

Diagrama de blocos



❖ Exemplo - Sequência



Diagrama de blocos

❖ Exemplo - Seleção (decisão)

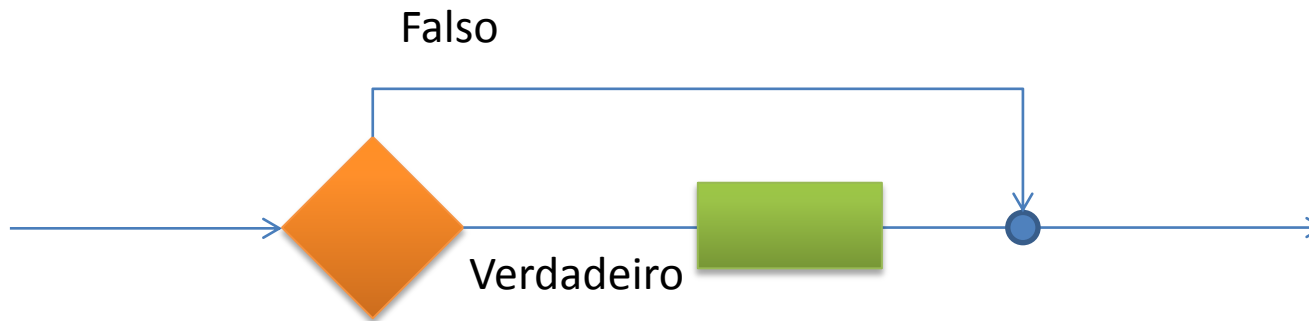


Diagrama de blocos

❖ Exemplo - Seleção (decisão)

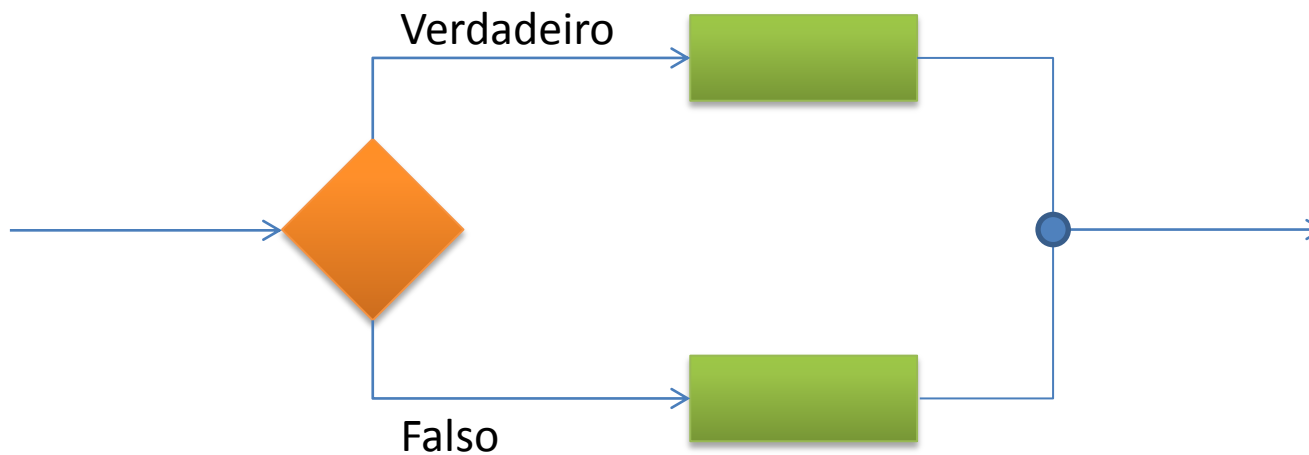


Diagrama de blocos

❖ Exemplo - Seleção múltipla

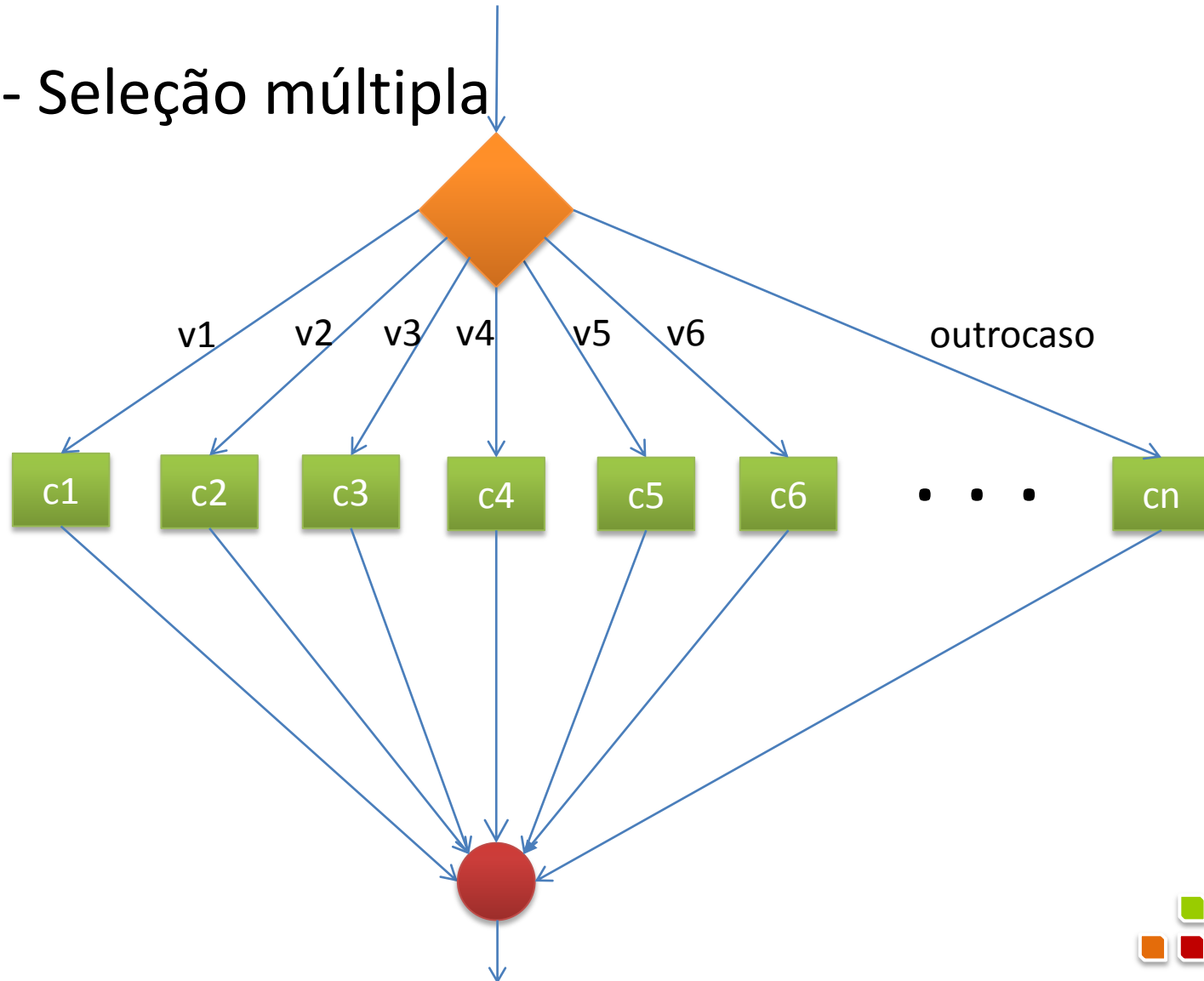
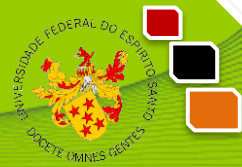


Diagrama de blocos



❖ Exemplo - Estrutura Repetição (Iteração)

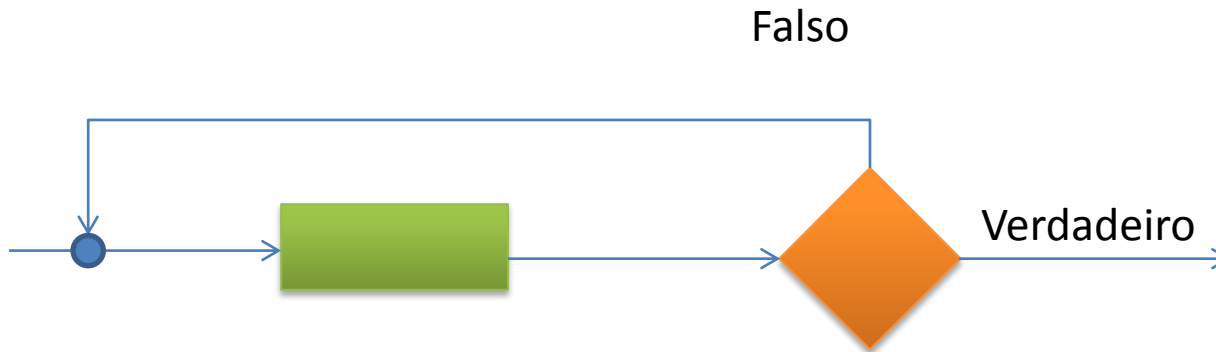
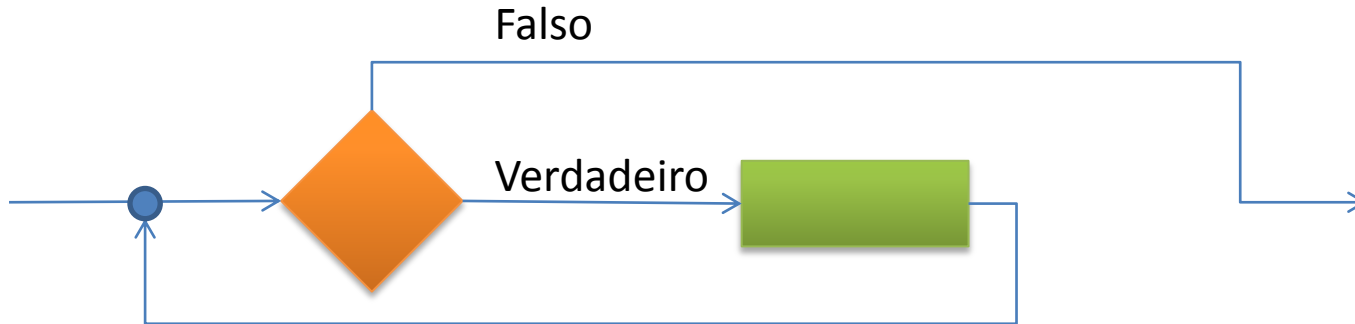


Diagrama de blocos

❖ Exemplo - Estrutura Repetição (Iteração)



Exercícios



- 1) Defina algoritmo com suas palavras.
- 2) Cite 3 exemplos de algoritmos que você costuma utilizar no seu dia-a-dia.
- 3) Um algoritmo não pode conter um comando como “Escreva todos os números inteiros positivos”. Por quê?
- 4) Escreva um algoritmo usando descrição narrativa que receba como entrada um número e exiba seu sucessor.
- 5) Faça um algoritmo que aceite como entrada dois valores numéricos, calcule e forneça como saída:
 - a) A soma destes valores
 - b) O produto deles
 - c) O quociente entre eles



Exercícios



- 6) Elaborar um algoritmo que calcule o quociente inteiro e o resto de uma divisão, dados o dividendo e o divisor, com a restrição de que as únicas operações aritméticas disponíveis são: adição, subtração e multiplicação.

$$\begin{array}{r} \text{Dividendo} \rightarrow 112 \\ \underline{90} \\ 22 \\ \text{Resto} \end{array} \quad \begin{array}{r} \text{Divisor} \\ 30 \\ \hline 3 \\ \text{Quociente} \end{array}$$



Referências



- ❖ DEITEL, P. J.; DEITEL, H.M.; Java: How to program, 9th ed, Ed. Prentice-Hall, 2011. ISBN: 978-0-13-257566-9.
- ❖ FARRER, H.; BECKER, C. G.; FARIA, E. C.; MATOS, H. F.; et al. Algoritmos estruturados. 3ed, Ed. LTC, 1999. ISBN: 9788521611806.
- ❖ GUIMARÃES, A. M.; LAGES, N. A. C.; Algoritmos e estruturas de dados. 1ed, Ed. LTC, 1994. ISBN: 9788521603788.
- ❖ GRIFFITHS, D., BARRY, P., Head First Programming – A learner's guide to programming using the Python language, O'Reilly, 2009, 406p.
- ❖ FARRER, H.; BECKER, C. G.; FARIA, E. C.; MATOS, H. F.; et al. Pascal estruturado. 3ed, Ed. LTC, 1999. ISBN: 9788521611745.
- ❖ Velloso, F. C.; Informática: Conceitos Básicos. 7ed, Ed. Campus, 2004. ISBN: 9788535215366.

