2013

Portugol

Equivalências de estruturas entre Portugol e Perl

Decode Team
INSTITUTO POLITÉCNICO DE TOMAR

Índice

Nota Geral:

Devido à especificação da linguagem, a tradução só é possível depois de ser executado o fluxograma.

Algumas notas sobre Perl:

- Software livre, lançado em 1987 por Larry Wall;
- Tem como lema: “There's More Than One Way To Do It"
- É uma das linguagens mais populares de programação web, devido às suas capacidades de manipulação de textos;
- Influenciada pelas linguagens *shell script*;
- Suporta vários tipos de base de dados (Oracle, Sybase, PostgreSQL, MySQL, etc);
- Unicode;
- Multiplataforma;
- Usa o ponto e virgula (;) para terminar uma linha de código;
- Os comentários do código, são iniciados pelo caracter cardinal (#);
- As funções podem ser definidas antes ou depois do código a executar;
- O código deve ser guardado num ficheiro com extensão .pl .

Estrutura Início

Início:

Apenas em ambiente Linux é necessário iniciar o programa com a instrução `#!/usr/bin/perl` por forma a identificar a bash/shell (interpretador de comandos) que vai ser usado.
Noutras plataformas não é necessário nem usada nenhuma instrução específica para indicar o começo do programa.

Estrutura Fim

Fim:

Não é necessário nem usada nenhuma instrução específica para indicar o final do programa.

Variáveis

Tipos de variáveis

Linguagem *Perl* é considerada como não-tipada.
Em *Perl*, as variáveis não ficam restringidas a ter um único tipo de dados, o interpretador irá automaticamente definir o seu tipo baseado no contexto em que a variável se associa.
Não existe necessidade de pré-declarar o tipo da variável.
Existem três estruturas básicas de dados:

- **Escalares:** assumem um valor único (número – inteiro ou real, string e referência);
- **Listas/Arrays:** conjunto ordenado de escalares;
- **Hashes:** conjunto indexado de escalares, não ordenados, que podem ser acedidos através de chaves.

Definição e atribuição de variáveis

variavel <- valor

Todos os nomes podem conter letras, o caracter *underscore* e números, desde que não se inicie com número.

- **Escalares**

  Todas as variáveis são declaradas/iniciadas pelo caracter **$**

  Ex:

  $sum

  $sumTotal

  $sum_1

  Para atribuir dados à variável:

  $nome_variavel = valor;

  O valor pode ser um número inteiro ou real, uma string ou uma referência.

  Ex: $var = 1; $var = 1.5; $var = “Texto”;

- **Arrays**

  Todas as variáveis são declaradas/iniciadas pelo caracter **@**

  Ex:

  @animais; @escolaSec; @escola_1;

  Para atribuir dados ao array:

  @variavel =(valor1, valor2);

  Ex:

  @animais = (“gato”, “cao”, “porco”);

  @numeros = (1, 2, 3, 4);

- **Hashes**

  Todas as variáveis são declaradas/iniciadas pelo caracter **%**

  Ex:

  %dados = (“Um”, 1, “Dois”, 2, “Tres”, 3);

  Para adicionar novos elementos:

  %dados{Quatro} = 4;

Para remover elementos:

delete($dados{Tres});

Estruturas input/output

Input – Ler

Para ler do teclado, teremos que atribuir a instrução <STDIN> à variável que vai estar associado o valor introduzido:

$num = **<STDIN>**;

Output – Escrever

Para escrever no ecrã, é utilizada a instrução ***print.***
Após escrever a mensagem, permanece na mesma linha, exceto se no final da instrução encontrar **\n**

Ex:

```
print “Nome”;
print “Apelido;
```

Teremos no ecrã:
NomeApelido

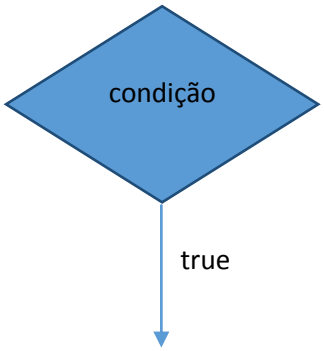
Mas se colocar-mos:

```
print “Nome\n”;
print “Apelido”;
```

Teremos:
Nome
Apelido

Estruturas de Decisão

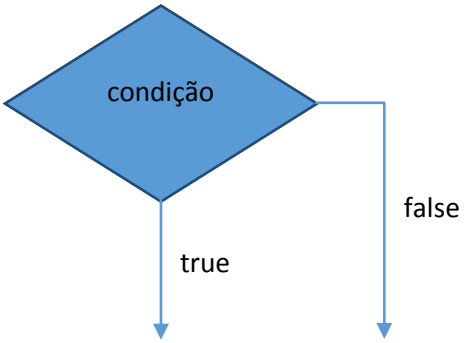
Condição *"if"*




```
if (condição)
{
Instruções;
}
```

Nota: A condição deve estar dentro de parêntesis

Condição *"if/else"*




```
if (condição)
{
Instruções;
}
else
{
Instruções;
}
```

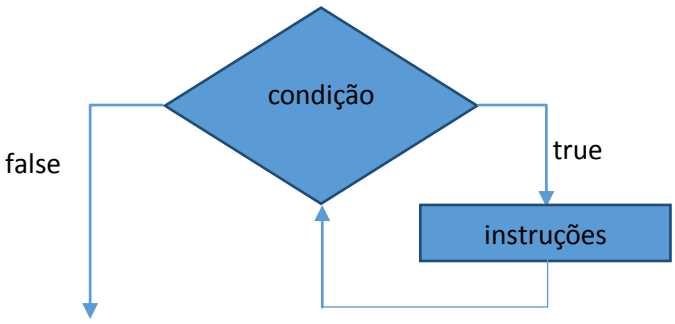
Nota: A condição deve estar dentro de parêntesis

Condição *"else if"*

```
if (condição1)
 {
Instruções;
}
elsif(condição2)
 {
Instruções;
}
else
{
Instruções;
}
```

Nota: A condição deve estar dentro de parêntesis

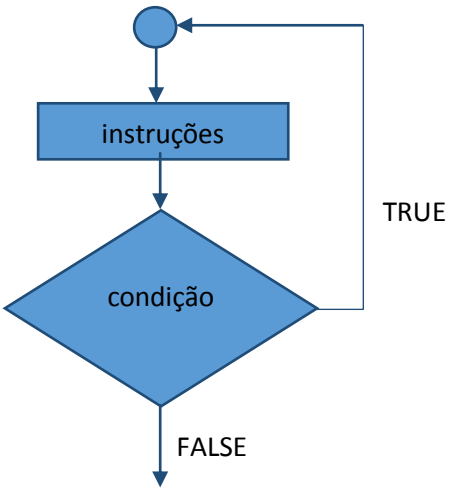
Condição *"while"*




```
while (condição){
        Instruções
    }
```

Nota: A condição deve estar dentro de parêntesis

Condição *"do while"*




```
do{
        Instruções
 }while(condição);
```

Nota: A condição deve estar dentro de parêntesis

Funções

Definir funções

Não existe nenhum procedimento especial quanto ao local onde devem ser definidas as funções, estas podem estar antes ou após o código a executar. São habitualmente definidas por:

```
sub nomeFuncao{

}
```

Sempre que exista necessidade de se utilizar parâmetros, estes são definidos/lidos através da expressão **@_** que é atribuído a uma variável. Exemplo:

```
sub funcaoExemplo{
        $numParametros = @_ ;

        print "O numero de parâmetros é $numParametros \n";
}
```

Chamada de funções

Aproveitando o exemplo da função acima declarada:

funcaoExemplo(1); #Resultado: **O numero de parâmetros é 1**
funcaoExemplo(1,2); #Resultado: **O numero de parâmetros é 2**
funcaoExemplo(1..3); #Resultado: **O numero de parâmetros é 3**
funcaoExemplo("A"..."Z"); #Resultado: **O numero de parâmetros é 26**

Estrutura de retorno

Return

É definido dentro da função/sub-rotina implementada:

return expressao;

Operadores

Aritméticos

Nome	Portugol	Perl
Adição	a + b	a + b
Subtração	a – b	a - b
Divisão	a / b	a / b
Multiplicação	a * b	a * b
Resto da divisão inteira		a % b
Potenciação		**
Concatenação de texto		+

Tabela 1 - Equivalência de operadores aritméticos

Lógicos

Nome	Portugol	Perl
Disjunção	a && b	a && b
Conjunção	a || b	a || b
Conjunção Exclusiva	a ^ b	a ^ b
Negação		a !b

Tabela 2- Equivalência de operadores lógicos

Relacionais Numéricos

Nome	Portugol	Perl
Igual	a == b	a == b
Diferente	a != b	a != b
Maior	a > b	a > b
Maior ou igual	a >= b	a >= b
Menor	a < b	a < b
Menor ou igual	a <= b	a <= b

Tabela 3- Equivalência de operadores relacionais

Relacionais Strings

Nome	Portugol	Perl
Igual	a == b	a eq b
Diferente	a != b	a ne b
Maior	a > b	a gt b
Maior ou igual	a >= b	a ge b
Menor	a < b	a lt b
Menor ou igual	a <= b	a le b

Tabela 3- Equivalência de operadores relacionais

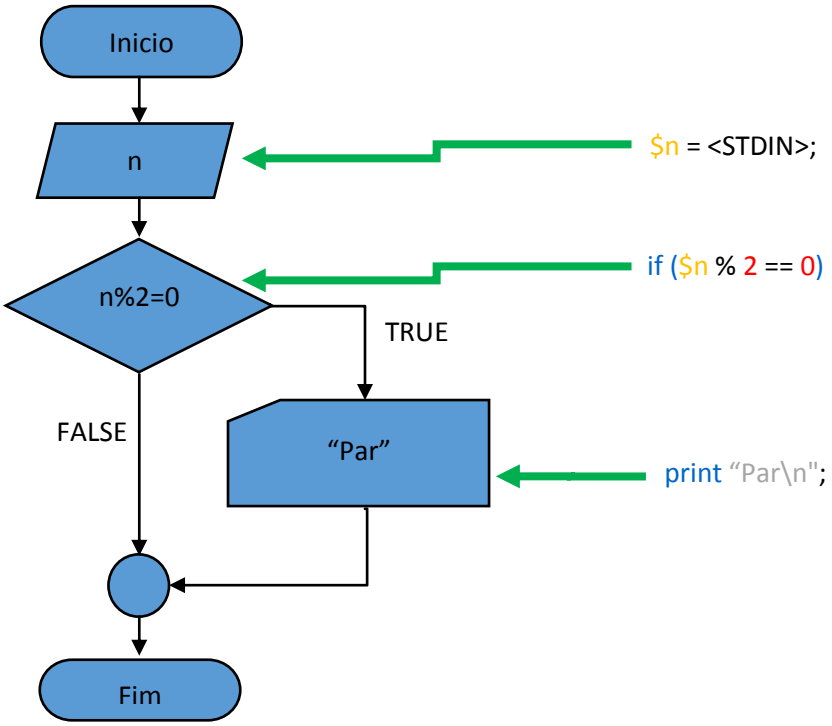
ANEXO

Para uma compreensão mais abrangente do uso das estruturas, ficam alguns exemplos mais extensivos, com o uso de várias estruturas em algoritmos completos.

Algoritmo com o uso da condição "if"

Problema: Verificar se um número introduzido pelo utilizador é par.

Fluxograma



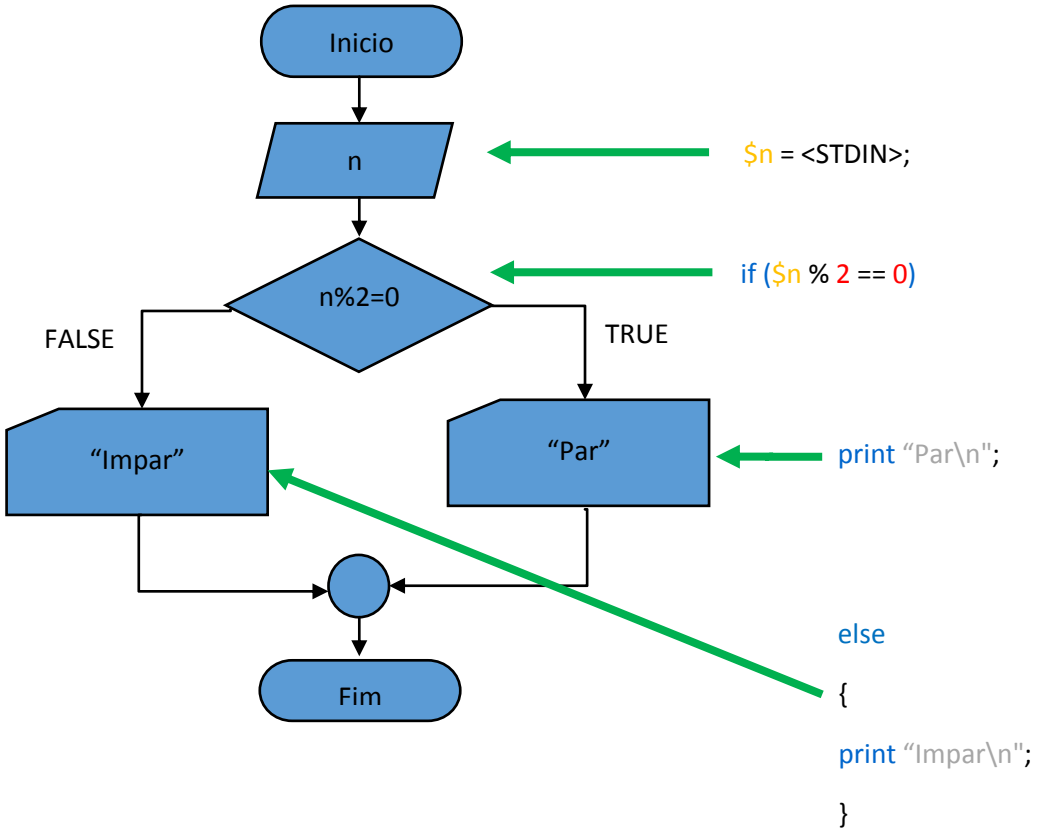
Código

```
$n = <STDIN>;
if ($n % 2 == 0)
{
        print "Par\n";
}
```

Algoritmo com o uso da condição "if else"

Problema: Verificar se um número introduzido pelo utilizador é par ou ímpar.

Fluxograma



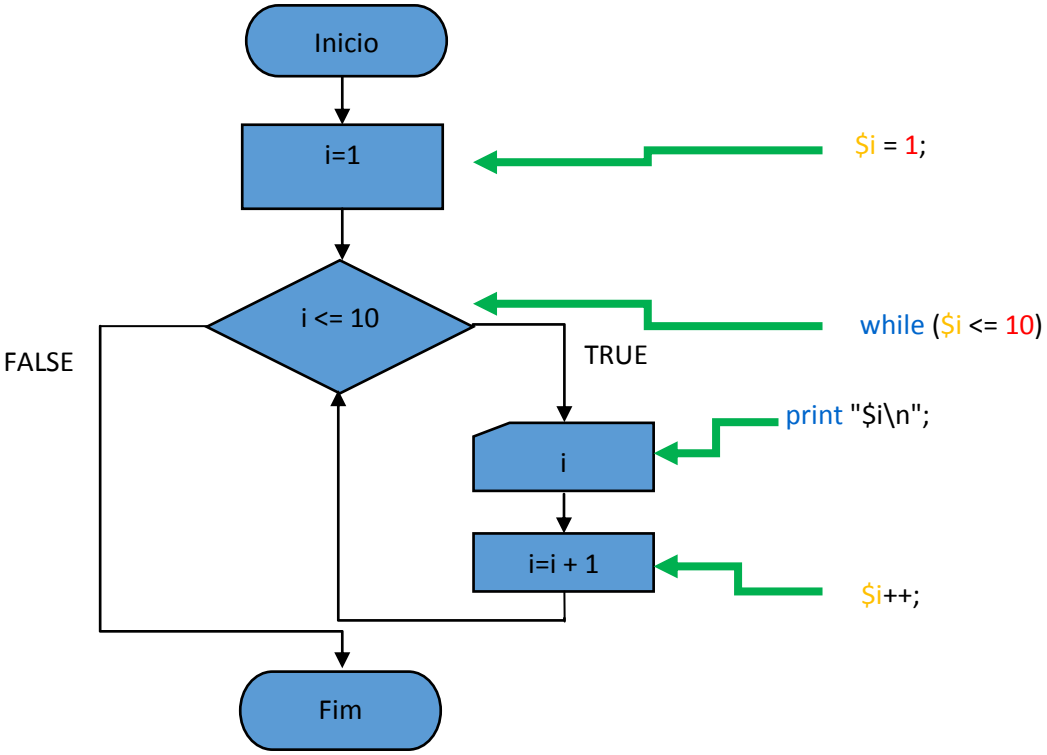
Código:

```
$n = <STDIN>;
if ($n % 2 == 0)
{
        print "Par\n";
}
else
{
        print "Impar\n";
}
```

Algoritmo com o uso da condição “while”

Problema: Escrever um número de 1 a 10.

Fluxograma



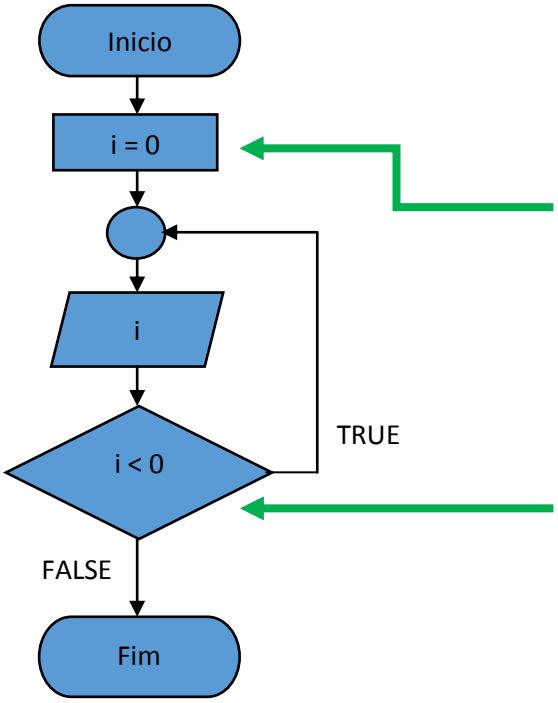
Código

```
$i = 1;
while ($i <= 10)
{
        print "$i\n";
        $i++;
}
```

Algoritmo com o uso da condição " do while"

Problema: Pedir um número positivo.

Fluxograma




```
$i = 0;
do
{
print "Introduza um numero positivo:\n";
}
while ($i < 0)
```

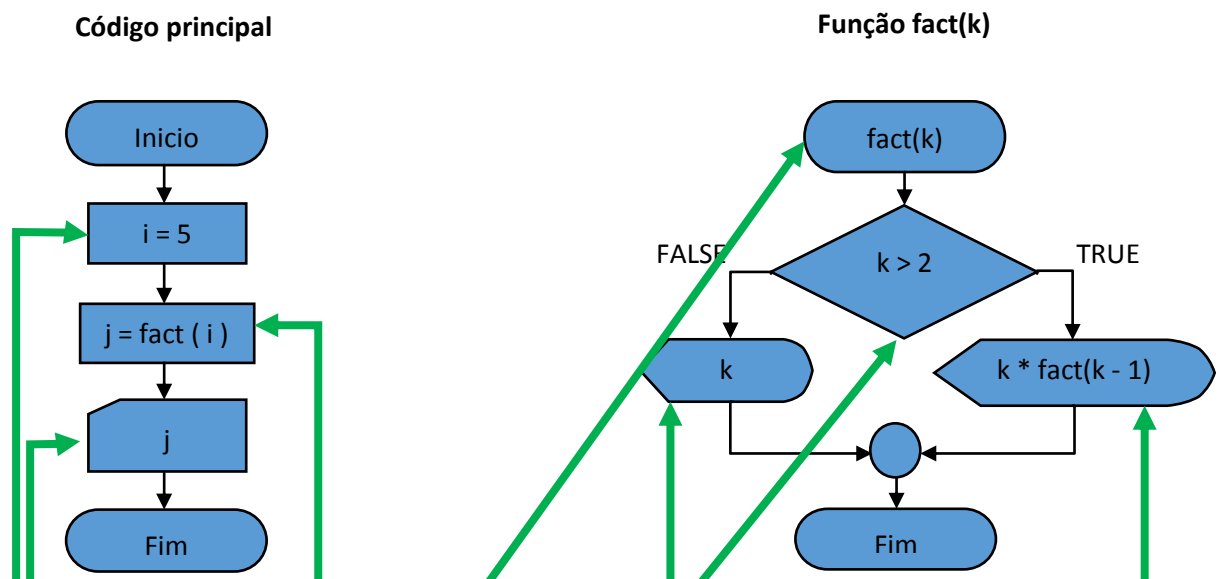
Código

```
do
{
        print "Introduza um numero positivo:\n";
        $i = <STDIN>;
}
while (i<0)
```

Algoritmo com o uso de uma função

Problema: Factorial de um número.

Fluxogramas



Código

```
$i = 5;
$j = factorial($i);

print "$j\n";

sub factorial
{
        my $k = shift;
        if ($k > 2)
        {
                $j = $k * factorial($k-1);
        }
        else
        {
                $j = $k;
        }
}
```