Lógica de Programação

Sérgio Queiroz de Medeiros
Escola de Ciências e Tecnologia
UFRN

Agosto de 2016

Capítulo 1

Introdução

Em um curso como Lógica de Programação, onde o aluno vai começar o aprendizado de algoritmos e de programação de computadores, é comum a utilização de uma linguagem de programação que possa facilitar o entendimento do assunto, ao invés de usar linguagens de programação mais convencionais.

Geralmente uma linguagem de programação utiliza termos em inglês, de modo que ela possa ser usada por pessoas de todo o mundo. Contudo, o uso de termos em inglês pode dificultar o aprendizado de alguns alunos. Por conta disso, algumas linguagens, voltadas principalmente para o aprendizado de programação, usam termos em português. Essas linguagens são usualmente chamadas de *Portugol*.

Neste curso utilizaremos a linguagem de programação *QuoGol* , cuja sintaxe foi inspirada na sintaxe da linguagem Quorum (`https://www.quorumlanguage.com/`).

Com QuoGol será possível implementarmos algoritmos que envolvam estruturas de decisão, estruturas de repetição, coleções de dados e funções.

Para executarmos os programas escritos em QuoGol é necessário instalar o ambiente de programação da linguagem, que está disponível em `https://github.com/alex7alves/Setup-QuoGol-IDE`.

1.1 Programa *Olá, mundo*

Vamos começar o nosso aprendizado com um programa QuoGol simples, que imprime um texto:

```
escreva("Olá, mundo")
```

No programa acima usamos a função `escreva`, que recebe um valor entre parênteses e o imprime para a saída padrão do computador (o monitor, geralmente). Nesse programa o valor recebido por `escreva` foi um texto, delimitado por aspas duplas. Quando executado, esse programa imprime *Olá, mundo*.

Escreva o programa acima no ambiente de programação de QuoGol , salve o arquivo como `alo.gol` e tente executá-lo. Use sempre a extensão `.gol` para os arquivos com programas QuoGol .

A função `escreva` não imprime só texto, ela também imprime valores inteiros e outros tipos de dados que veremos mais adiante. Por exemplo, no programa a seguir a função `escreva` recebe dois valores, um texto e o valor inteiro 42:

```
escreva("A resposta é ", 42)
```

Note que usamos uma vírgula para separar os valores passados para a função `escreva`.

1.2 Fazendo contas

Podemos usar QuoGol como uma calculadora, onde a função `escreva` irá imprimir o resultado de uma determinada expressão, como no exemplo abaixo:

```
escreva("A resposta é ", 1 + 1)
```

Nesse exemplo, a expressão `1 + 1` terá seu valor calculado e então esse valor será exibido. Podemos calcular também o resultado de expressões mais complexas:

```
escreva("A resposta é ", 3 + 4 * 5)
```

Nesse caso, como o operador * possui maior precedência a multiplicação `4 * 5` será realizada primeiro e o seu resultado será somado ao valor 3. Para mudar a ordem em que as operações devem ser feitas podemos usar parênteses, como no exemplo a seguir:

```
escreva("Usando parênteses ", (3 + 4) * 5, (5 + 7)  / (3 - 1))
```

Esse programa deve imprimir os valores 35 e 6 como resultado das operações de multiplicação e divisão, respectivamente.

1.3 Variáveis

Nos programas anteriores fizemos operações com literais, isto é, valores que já estavam definidos no próprio programa (os números 3, 7, etc). Agora vamos escrever um programa que vai pedir para o usuário digitar valores e então vai fazer operações sobre esses valores.

Esse programa requer que as informações digitadas pelo usuário sejam armazenadas e para isso vamos usar variáveis. Uma variável é um nome associado a uma região da memória do computador. Podemos usar esse nome para acessar e alterar o valor armazenado nessa região de memória.

Abaixo, temos o programa `somaInteiro.gol`, que calcula a soma de valores inteiros digitados pelo usuário:

```
01  inteiro valor1, valor2, soma
02
03  escreva("Digite um valor inteiro: ")
04  leia(valor1)
05
06  escreva("Digite outro valor inteiro: ")
07  leia(valor2)
08
09  soma = valor1 + valor2
10  escreva("A soma deu ", soma)
```

Na linha 1 declaramos as variáveis `valor1`, `valor2` e `soma`, do tipo **inteiro**. Em QuoGol toda variável possui um tipo, que indica a natureza dos valores que iremos armazenar naquela variável. No exemplo acima, o tipo **inteiro** indica que iremos guardar valores inteiros, tais como 42, -255, 0 e 1000000, nas variáveis `valor1`, `valor2` e `soma`.

Toda variável deve ser declarada antes de ser usada. Para declarar uma variável devemos informar o tipo e o nome da variável.

Na linha 3, usamos a função `escreva`, que vimos anteriormente, para pedir que o usuário forneça um valor inteiro.

Usamos a função `leia` na linha 4 para ler a informação digitada pelo usuário e armazená-la na região de memória associada à variável `valor1`. De modo similar, na linha 7 usamos `leia` para guardar na variável`valor2` o segundo valor inteiro digitado pelo usuário.

Na linha 9, usamos o operador de atribuição = para guardar na variável `soma` o resultado da operação `valor1 + valor2`, que é um valor inteiro. Mais adiante iremos estudar a precedência dos operadores, que determina que operações serão realizadas primeiro. No caso da linha 9, primeiro será realizada a operação + (soma) e depois a operação = (atribuição).

Em seguida, na linha 10, imprimimos um texto seguido do valor armazenado em `soma`.

Digite o programa QuoGol acima no seu ambiente de programação e depois o execute. Em seguida, modifique o programa de modo que ele imprima também o valor da subtração `valor1 - valor2`.

1.4 Operadores Aritméticos

Abaixo listamos os operadores aritméticos disponíveis em QuoGol , os quais são todos operadores binários:

```
+   -   *   /   mod
```

O operador / calcula o resultado da divisão entre dois valores x e y. Se x e y são valores inteiros, então o resultado será um valor inteiro. Por exemplo, a expressão abaixo:

```
12 / 5
```

produzirá o valor 2.

Se um dos valores for um número real, como por exemplo 3.14[1], então o resultado será um número real. Por exemplo, a expressão:

```
12 / 5.0
```

produzirá o valor 2.4.

Para armazenar um valor real, devemos usar uma variável do tipo **numero**, como mostrado no exemplo abaixo:

```
01  numero x = 12 / 5
02  escreva("x = ", x)
03
04  x = 12 / 5.0
05  escreva("x = ", x)
```

Na linha 1 declaramos a variável `x` e a inicializamos com o valor da expressão `12 / 5`. Na linha 2 imprimimos o valor de `x`, no caso 2. Note que uma variável do tipo **numero** também pode guardar valores inteiros, porém uma variável do tipo **inteiro** não pode guardar valores reais.

Na linha 4, atribuímos a `x` o valor da expressão `12 / 5.0` e na linha 5 imprimimos o novo valor de `x`.

Enquanto o operador `/` nos dá o resultado do quociente de uma divisão, o opererador **mod** nos dá o resto de uma divisão. Por exemplo, o resultado da expressão

```
14 mod 5
```

é 4, que representa o resto da divisão de 14 por 5.

Ao contrário dos outros operadores aritméticos, que aceitam valores reais e inteiros, o operador **mod** somente pode ser aplicado a operandos inteiros.

O operador **mod** é bastante útil se desejamos saber se um número é ou não divisível por outro. Quando um número é divisível por outro o resultado do operador **mod** é zero.

Abaixo, temos um programa que lê dois números inteiros `x` e `y` e imprime `verdadeiro` se `x` é divisível por `y`, e **falso** caso contrário:

```
01  inteiro x, y
02
03  escreva("Digite dois valores inteiros: ")
04  leia(x, y)
05
06  inteiro resto = x mod y
07  escreva(x,  " divisível por ",  y, "? ", resto == 0)
```

[1]Note que devemos usar o ponto ao invés da vírgula para separar a parte inteira da parte fracionária de um número.

Na linha 1 declaramos as variáveis inteiras `x`, `y`, e na linha 4 fazemos a leitura dos valores de `x` e `y`.

Em seguida, na linha 6, declaramos e inicializamos a variável `resto` como o resto da divisão de `x` por `y`. Na linha 7 usamos o operador de igualdade ==. A expressão `resto == 0` produzirá um valor booleano, isto é, um valor verdadeiro (quando `resto` for igual a zero) ou falso (quando `resto` for diferente de zero).

Note que no programa anterior não guardamos o resultado da expressão `resto == 0` em uma variável, ao invés disso imprimimos diretamente o valor dessa expressão. Claro que também poderíamos guardar o valor dessa expressão em uma variável, e então imprimir o valor dessa variável. Nesse caso, precisaríamos usar uma variável do tipo **booleano**. A seguir listamos os tipos disponíveis em QuoGol e qual informação podemos armazenar em uma variável de cada tipo:

- **booleano**: representa os valores **verdadeiro** e **falso**
- **inteiro**: representa valores inteiros, como 101, -93 e 0
- **numero**: representa valores reais, como 3.14, -10.4, e 42.0
- **texto**: representa sequências de caracteres, que devem estar entre aspas duplas, tais como "bola"e "ciclo básico"

1.5 Conversão de Temperaturas

Dado que sabemos como realizar operações aritméticas em QuoGol , podemos escrever um programa que converte temperaturas. Em todo o mundo as temperaturas são medidas na escala Celsius, porém os Estados Unidos ainda usam uma escala diferente, e medem a temperatura em graus Fahrenheit.

A nossa tarefa é fazer um programa que converte uma temperatura em graus Celsius (ou centígrados) para Fahrenheit.

Não adianta tentar escrever um programa para isso se não sabemos como converter uma temperatura entre essas duas escalas.

Na escala Celsius a água congela a 0°e ferve a 100°C, enquanto que na escala Fahrenheit ela congela a 32°e ferve a 212°F.

Com isso, podemos definir uma regra de conversão entre essas escalas da seguinte maneira, onde `C` representa uma temperatura qualquer em graus Celsius e `F` representa uma temperatura qualquer em graus Fahrenheit:

$$\frac{F-32}{212-32} = \frac{C-0}{100-0} \quad \Rightarrow \quad \frac{F-32}{180} = \frac{C}{100}$$

Podemos dividir ambos os lados da igualdade por 20 e então obter a equação:

$$\frac{F-32}{9} = \frac{C}{5} \quad \Rightarrow \quad F = \frac{9 * C}{5} + 32$$

Agora que sabemos como converter uma temperatura na escala Celsius para a correspondente na escala Fahrenheit podemos escrever o nosso programa QuoGol , que está listado na figura 1.1:

```
01  //Programa que converte de Celsius para Fahrenheit
02
03  numero tempC, tempF
04
05  escreva("Digite uma temperatura em graus Celsius: ")
06  leia(tempC)
07
08  tempF = (tempC * 9) / 5  +  32
09
10  escreva(tempC, "C  equivalem a ", tempF, "F")
```

Figura 1.1: Programa que converte uma temperatura em graus Celsius para Fahrenheit

Na linha 1 do programa temos um comentário de uma linha. Um comentário é indicado por `//` e não faz parte da lógica do programa, é apenas uma informação para o programador, muitas vezes usado para explicar o que faz um certo trecho de código. O texto de um comentário é ignorado pelo compilador ao gerar um arquivo executável.

Na linha 3 declaramos as variáveis `tempC` e `tempF` para armazenar os valores das temperaturas em graus Celsius e Fahrenheit, respectivamente. Note que elas são do tipo `numero`, não do tipo `inteiro`, pois podemos representar uma temperatura como um valor real.

Na linha 6 inicializamos a variável `tempC` com o valor fornecido pelo usuário. Em seguida, na linha 8, convertemos a temperatura para Fahrenheit de acordo com a equação mostrada anteriormente.

Por fim, na linha 10, imprimimos o valor da temperatura nas duas escalas.

Se fornecermos como entrada do programa a temperatura prevista para hoje em Natal, que é de 29°C, ele mostrará o valor correspondente de 84.2°F.

1.6 Exercícios

1. Escreva um programa que converte um valor em dólares para o valor correspondente em reais. O seu programa deve ler a quantidade de dólares que se deseja converter, a taxa de conversão, e então imprimir o valor equivalente em reais.

   - Exemplo de entrada:

     ```
     100 3.5
     ```

   - Exemplo de saída:

     ```
     U$$ 100 equivalem a R$ 350
     ```

2. Escreva um programa que dada uma quantidade S de segundos calcula e imprime a quantidade correspondente de horas, minutos e segundos.

- Exemplos de entrada:

```
201
3670
```

- Exemplos de saída:

```
0 hora(s), 3 minuto(s) e 21 segundo(s)
1 hora(s), 1 minuto(s) e 10 segundo(s)
```

3. Dado um número inteiro n, imprima os três últimos dígitos de n separados por espaços.

   - Exemplos de entrada:

```
1234
56
```

   - Exemplos de saída:

```
2  3  4
0  5  6
```

1.7 Operadores Relacionais e Lógicos

Além dos operadores aritméticos, em QuoGol também temos operadores relacionais e lógicos.

Já vimos um exemplo de operador relacional, o operador de igualdade ==. Todos os operadores de igualdade possuem dois operandos e produzem os valores **verdadeiro** ou **falso**. Abaixo listamos todos os operadores relacionais, onde **nao=** é o operador de negação da igualdade:

```
==   nao=   <   >   <=   >=
```

Podemos aplicar os operadores == e **nao=** a qualquer tipo de valor, isto é podemos aplicá-lo a valores inteiros, reais, booleanos e textuais. Os operadores de ordem <, >, <= e >= podem ser aplicados a valores inteiros, reais e textuais, mas não podem ser aplicados a valores booleanos.

Para manipular valores booleanos devemos usar os operadores lógicos, que também produzem como resultado um valor booleano. Abaixo listamos os operadores lógicos de QuoGol , onde o operador de conjunção (**e**) e o operador de disjunção (**ou**) são operadores binários, enquanto que o operador de negação (**nao**) é unário:

```
e   ou   nao
```

Sejam `exp1` e `exp2` duas expressões booleanas (isto é, duas expressões cujo valor é verdadeiro ou falso), o operador **e** somente produzirá um valor verdadeiro quando ambas as expressões forem verdadeiras. Podemos ver a tabela verdade para esse operador na figura 1.2.

exp1	exp2	exp1 **e** exp2
falso	falso	falso
falso	verdadeiro	falso
verdadeiro	falso	falso
verdadeiro	verdadeiro	verdadeiro

Figura 1.2: Tabela verdade para o operador lógico **e** (conjunção)

exp1	exp2	exp1 **ou** exp2
falso	falso	falso
falso	verdadeiro	verdadeiro
verdadeiro	falso	verdadeiro
verdadeiro	verdadeiro	verdadeiro

Figura 1.3: Tabela verdade para o operador lógico **ou** (disjunção)

Já a aplicação do operador **ou** resultará em um valor verdadeiro quando uma das expressões for verdadeira, como podemos ver na figura 1.3, que apresenta a tabela verdade correspondente a esse operador.

Os operadores **e** e **ou** fazem avaliação de curto-circuito, isto é, eles só avaliam o seu segundo operando quando necessário. No caso do **e**, o segundo operando só é avaliado se o primeiro for verdadeiro, caso contrário já é possível determinar que o resultado da conjunção será falso. No caso do **ou**, o segundo operando só é avaliado se o primeiro for falso, caso contrário a disjunção será verdadeira.

O operador de negação **nao** inverte o valor da expressão booleana sobre o qual é aplicado. Podemos ver a sua tabela verdade na figura 1.7, onde `exp` representa uma expressão booleana.

O operador **nao** possui maior precedência do que o **e**, que por sua vez possui maior precedência do que o **ou**.

Vamos ver o uso desses operadores em um programa que imprime verdadeiro ou falso dependendo se um aluno foi aprovado ou não em uma disciplina. Assuma que o aluno recebeu a nota de duas avaliações e para ser aprovado deve ter pelo menos uma nota maior ou igual a 7 e não pode tirar nenhuma nota abaixo de 3.

Considerando que `nota1` e `nota2` são variáveis com os valores da notas do aluno, a seguinte expressão booleana verifica as condições de aprovação:

```
(nota1 >= 7.0 ou nota2 >= 7.0) e nao (nota1 < 3.0) e nao (nota2 < 3.0)
```

exp	**nao** exp
falso	verdaeiro
verdadeiro	falso

Figura 1.4: Tabela verdade para o operador lógico **nao** (negação)

```
01  numero nota1, nota2
02
03  escreva("Digite a nota da primeira avaliação: ")
04  leia(nota1)
05
06  escreva("Digite a nota da segunda avaliação: ")
07  leia(nota2)
08
09  booleano resultado
10  resultado = (nota1 >= 7.0 ou nota2 >= 7.0) e nao (nota1 < 3.0) e nao (nota2 < 3.0)
11
12  escreva("Aprovado? ", resultado)
```

Figura 1.5: Programa que verifica se um aluno foi aprovado

Note o uso de parênteses para forçar a ordem de avaliação desejada. Uma outra maneira de expressar a condição acima seria como abaixo:

```
(nota1 >= 7.0 ou nota2 >= 7.0) e nota1 >= 3.0 e nota2 >= 3.0
```

A figura 1.5 apresenta um programa que imprime uma mensagem informando se um aluno foi aprovado ou não de acordo com a condição acima.

Na linha 1 declaramos duas variáveis do tipo **numero**, pois as notas de um aluno podem ser valores reias como 5.5 e 8.9. Em seguida, nas linhas 3–7, fazemos a leitura das notas. Nas linhas 9 e 10 declaramos uma variável do tipo booleano e atribuímos a ela o resultado da expressão que verifica a condição de aprovação. Por fim, na linha 12 imprimimos o resultado.

1.8 Exercícios

1. Implemente um programa que recebe um valor inteiro correspondente a um ano e imprime **verdadeiro** se esse ano é bissexto e falso caso contrário. Um ano é bissexto se ele é múltiplo de 4 mas não de 100, ou se ele é múltiplo de 400.

   - Exemplos de entrada:

     ```
     2016
     1900
     2000
     ```

   - Exemplos de saída:

     ```
     verdadeiro
     falso
     verdadeiro
     ```