Lógica de programação

## A disciplina “Lógica de programação orientada a objetos” visa apresentar ao aluno o conhecimento básico para a programação de computadores.

## Na primeira parte será estudado o conceito de algoritmo, bem como sua representação por meio da linguagem chamada “pseudocódigo”.

Na segunda parte será apresentada a linguagem de programação C#, muito utilizada na programação de jogos na plataforma *Unity*. O básico da linguagem será apresentado. Conhecimento necessário para a confecção e interpretação de programas com complexidade simples a moderada.

Por fim, na terceira etapa serão apresentados os conceitos fundamentais da “orientação a objetos”, que é um dos paradigmas de programação mais utilizados no mundo.

Bons estudos!!

Primeira parte – Algoritmos

## Sobre lógica de programação

* O que é lógica?

Iniciamos o nosso estudo com esta pergunta...

- O que é lógica??

A palavra **lógica** é originária do grego *logos*, que significa linguagem racional.

*Segundo o* Dicionário Michaelis*:*

*“Sequência coerente de ideias, modo pelo qual se encadeiam naturalmente as coisas ou os acontecimentos.”*

A lógica se preocupa com a **forma**, como um pensamento ou ideia são **organizados** e **apresentados**, possibilitando uma **conclusão** por meio do **encadeamento dos argumentos**.

O campo da lógica estuda a *“****correção do raciocínio****”,* tem em vista a *“****ordem da razão****”,* ensina a colocar *“****ordem no pensamento****”.*

Podemos exercitar o conceito com os seguintes exemplos:

* + Ana é mais velha que Beto;
  + Beto é mais velho que Clara;

Com base nas afirmações postadas (chamadas de premissas), podemos concluir, logicamente, que Ana é mais velha que Clara.

Cavalos são animais.

Animais possuem patas

Logo, com base nas premissas, podemos concluir que os cavalos possuem patas.

Fica a reflexão sobre possíveis outras conclusões:

Podemos concluir que todos os animais são cavalos?

Podemos concluir que os cavalos possuem quatro patas?

Aproveitando os conceitos apresentados, quando entramos no campo da programação de computadores, temos que a **lógica de programação** consiste em encadear ações, ou instruções, a serem seguidas pelo computador para a resolução de problemas. Busca-se aprender a decidir como será a **sequência de passos que serão executados para atingir um objetivo.** Esta sequencia de passos é o algoritmo, conceito que será apresentado a seguir.

Algoritmo

Para resolver um problema em um computador, de modo semelhante ao que fazemos para resolver um problema do nosso cotidiano, devemos encontrar uma maneira de descrever esse problema de uma forma clara e precisa, através de uma sequência de passos a serem seguidos até que se atinja a solução procurada.

Essa estruturação deve seguir uma determinada lógica. A lógica aplicada à programação de computadores, deve buscar a melhor sequência de ações para se resolver um problema. Essa sequência de ações damos o nome de algoritmo.

Ou seja, podemos dizer que um algoritmo é um roteiro que mostra passo a passo os procedimentos necessários para a resolução de uma tarefa. O algoritmo não responde a pergunta “o que fazer?”, mas sim “como fazer”.

No dia a dia, vários algoritmos são utilizados de forma intuitiva e automática na execução tarefas comuns. Como estas atividades são simples e dispensam ficar pensando nas instruções necessárias para fazê-las, o algoritmo presente nelas acaba passando despercebido. Por exemplo, quando é necessário trocar uma lâmpada:  
  
 Início  
   Verificar se o interruptor está desligado;  
   Procurar uma lâmpada nova;  
   Pegar uma escada;  
   Levar a escada até o local;  
   Posicionar a escada;  
   Subir os degraus;  
   Parar na altura apropriada;  
   Retirar a lâmpada queimada;  
   Colocar a lâmpada nova;  
   Descer da escada;  
   Acionar o interruptor;  
     Se a lâmpada não acender, então:  
         Retirar a lâmpada queimada;  
         Colocar outra lâmpada nova  
     Senão  
         Tarefa terminada;  
   Jogar a lâmpada queimada no lixo;   
  Guardar a escada;  
 Fim

Algumas definições disponíveis de algoritmo:

*“Algoritmo é um* ***conjunto*** *finito de* ***regras*** *que provê uma* ***sequência*** *de operações para* ***resolver*** *um tipo de* ***problema*** *específico”.*

*“Algoritmo é uma s****equência ordenada****, e não ambígua, de passos que levam à* ***solução*** *de um dado* ***problema****”.*

*“Algoritmo é um processo de cálculo, ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições, as regras formais para a obtenção do resultado ou da solução do problema” .*

Destas três definições apresentadas, podem-se extrair as características mais relevantes que sintetizam o conceito de algoritmo que será trabalhado durante o curso:

* Sequência ordenada de operações;
* Sequência não ambígua;
* Levam a solução de um problema;
* Conjunto finito de regras que levam a solução de um problema.

Cabe destacar que um algoritmo **não é a solução do problema**, pois, se assim fosse, cada problema teria um único algoritmo; um **algoritmo é um caminho para a solução de um problema**.

É o **conceito central** da programação e da ciência da computação.

A figura a seguir expressa o ciclo de resolução de um problema via programação de computadores.

A partir de um problema busca-se uma solução. A partir da solução é montado como a solução será executada, em qual sequencia.

Com a definição do algoritmo, as instruções são traduzidas em alguma linguagem de programação que se converterá em uma linguagem de máquina que pode ser entendida pelo computador.

Utilidades de um algoritmo:

Serve para registrar o processo de solução de um problema

Serve para que outra pessoa possa resolver o mesmo problema, sem ter que reinventar a roda

* 3 tipos
  + Descrição narrativa
  + Representação gráfica (fluxograma)
  + Pseudocódigo
* **Descrição narrativa**

Os algoritmos são expressos diretamente em linguagem natural (o português, por exemplo).

Qual a vantagem?

* + *Não há a necessidade de aprender nenhum novo conceito.*

Qual a desvantagem?

* + *Em virtude da ambiguidade presente na linguagem natural a descrição narrativa é passível de mais de uma interpretação.*
* Algoritmo de domingo:
  + - Acordar;
    - Tomar o café;
    - Se estiver sol vou ao clube senão leio o jornal;
    - Ir ao cinema;
    - Fazer uma refeição;
    - Ir dormir;
    - Fim do domingo.
* **Representação gráfica (fluxogramas)**

Emprega formas geométricas padronizadas para indicar as diversas ações e decisões que devem ser executadas para resolver o problema.

Qual a vantagem?

* + *a representação gráfica é mais concisa que a representação textual.*

Qual a desvantagem?

* + *é necessário aprender a simbologia dos fluxogramas.*
* **Pseudocódigo**

Emprega uma linguagem intermediária entre a linguagem natural e uma linguagem de programação para descrever algoritmos.

Essa linguagem intermediária é chamada por "Portugol" ou "Português Estruturado".

Qual a vantagem?

* + *A passagem do algoritmo para qualquer linguagem de programação é quase imediata, bastando conhecer as palavras-reservadas dessa linguagem que serão utilizadas.*

Qual a desvantagem?

* + *É necessário aprender as regras de construção de um pseudocódigo*.
* É também utilizado para documentar rotinas de um sistema.
* Inicialmente, os algoritmos serão implementados em pseudocódigo, mas depois serão codificados em C#\*.
* *\*leia-se “C Sharp”*
* Antes de nos aprofundarmos nos métodos de representação de algoritmos, devemos ter de forma clara a compreensão de alguns conceitos básicos como:
  + Constantes;
  + Variáveis;
  + Identificadores;
  + Palavra-reservada;
  + Entrada;
  + Saída;
  + Operadores.
* **Constantes**
  + São valores que permanecem os mesmos do início ao fim.
  + Não são alterados pelas instruções do algoritmo.
  + Tipos:
    - Numéricos (positivos ou negativos)
      * 3 45 -8 5,92
    - Literais (alfanuméricos)
      * “TURMA” “nome” “09/08/2017”
    - Lógica

*verdadeiro* *falso*

* **Variáveis**
  + É um espaço de memória que recebeu um nome (identificador)
  + Armazena um valor que pode ser modificado durante a execução do algoritmo.
  + Devem ser declaradas no início do algoritmo indicando o tipo (numérica, literal, lógica).
* **Identificadores**

São os nomes utilizados para referenciar variáveis, constantes, funções ou vários outros objetos definidos em um algoritmo.

* Regras para formação de identificadores
  + o primeiro caractere deve ser, obrigatoriamente, uma letra;
  + do segundo caractere em diante são permitidos números e letras;
  + o símbolo de sublinhado (\_) pode ser usado para separar nomes compostos;
* **Identificadores** 
  + Regras para formação de identificadores (cont.)
    - não são permitidos espaços, caracteres acentuados e símbolos especiais na composição do nome de um identificador;
    - palavras reservadas não podem ser usadas;
    - há distinção entre maiúsculo e minúsculo (case sensitive).
  + Identificadores válidos:
    - DataNascimento
    - DATA\_DE\_NASCIMENTO
    - IDADE
    - Nota1
  + Identificadores inválidos:
    - 3CD
    - Meu Nome
    - Idade&
    - DA\*TA
    - Sílaba
* **Palavras-reservadas (palavras-chave)**

São identificadores predefinidos que possuem significados especiais para o interpretador do algoritmo.

* ***inicio, leia, se, então, senão, senão-se, fimse, escreva*** são exemplos de palavras-reservadas que representam estruturas presentes em todas as linguagens de programação.
* **algoritmo** “PegarUmOnibus”
* **início**
* ir até a parada
* **enquanto** ônibus não chega **faça**
* esperar ônibus
* **fimenquanto**
* subir no ônibus
* pegar passagem
* **se** não há passagem **então**
* pegar dinheiro
* **fimse**
* **Tipos de dados (primitivos)**
  + **Inteiro**

Define variáveis numéricas do tipo inteiro, ou seja, sem casas decimais.

* + **Real**

Define variáveis numéricas do tipo real, ou seja, com casas decimais.

* + **Literal** ou **caracter**

Define variáveis que podem receber letras, números e símbolos.

*Os números armazenados em uma variável cujo o tipo de dado é literal não poderão ser utilizados para cálculos.*

* + **Lógico** ou **booleano**

Define variáveis do tipo booleano, ou seja, com valor VERDADEIRO ou FALSO

* **Operadores**
  + **Operador de Atribuição ←**
    - Atribui um valor à variável.
    - Sintaxe
      * <identificador> **←** <valor a ser armazenado>
        + Exemplos

a ← 5 //variável ‘a’ recebeu o valor 5

valor1 ← 3.14

valor2 ← valor1

* + - Unários: +, -

Exemplo:

+1

-5.9

...

var a: inteiro

...

a **←** -a

* + - **Binários: +, -, \*, /, \, %, ^**
  + **Precedência (Hierarquia nas operações)**

Operadores do mesmo nível de precedência são avaliados pelo compilador da esquerda para a direita.

* **Expressões aritméticas**

Exemplos:

3/4+5 = 5.75

3/(4+5) = 0.33333333

3\2\*9 = 9

11%3^2 = 2

11%(3^2) = 2

(11%3)^2 = 4

3\2+(65-40)^(1/2) = 6

* **Operadores Relacionais**
* **Operadores Lógicos**

Tabela verdade

Durante a execução de determinadas tarefas ou durante a execução de um processo para obtenção da solução de um determinado problema **são necessários alguns dados** e ao final deve-se **retornar uma saída**.

Da mesma forma, em alguns algoritmos tornam-se necessários mecanismos que possibilitem uma **interface com o ambiente externo**, ou seja, são necessários **comandos que possibilitem a entrada e saída de dados**.

* **Entrada de dados**

Palavra-reservada: **leia**

Utilizada para receber dados externos ao algoritmo e armazená-los em variáveis.

Exemplos:

**var** a: inteiro

b: real

...

leia(a) *//variável “a” recebeu o valor inteiro recebido pelo sistema.*

leia(b, a) *// as variáveis b e a recebem os valores recebidos nessa ordem.*

* **Saída de dados**

Palavra-reservada: **escreva**

Utilizada para externar (enviar para o monitor) dados gerados pelo algoritmo.

Exemplo:

...

var x: inteiro

y: caracter

...

escreva (y) //caractere ‘y’ é apresentado na tela

escreva (x,y) //caractere ‘2y’ é apresentado na tela, no caso de x ter recebido o número 2

escreva (“Inteiro: ”, x-2) // ‘Inteiro: 1’ é apresentado na tela, no caso de x ter recebido o número 3

escreva(“Estudou e se dedicou tirou ”, 10) // ‘Estudou e se dedicou tirou 10’ é apresentado na tela

As expressões dentro desta lista devem estar separadas por vírgulas; depois de serem avaliadas, seus resultados são impressos na ordem indicada.

Os passos necessários para a construção de um algoritmo são:

* + ler atentamente o enunciado do problema, compreendendo-o e destacando os pontos mais importantes;
  + definir os dados de entrada, ou seja, quais dados serão fornecidos;
  + definir os dados de saída, ou seja, quais dados serão gerados depois do processamento;
  + definir o processamento, ou seja, quais cálculos serão efetuados e quais as restrições para esses cálculos. O processamento é responsável pela obtenção dos dados de saída com base nos dados de entrada;
  + definir as variáveis necessárias para armazenar as entradas e efetuar o processamento;
  + elaborar o algoritmo;
  + testar o algoritmo realizando simulações.
* A estrutura de um algoritmo em pseudocódigo pode variar um pouco de acordo com o autor ou com base na linguagem de programação que será utilizada posteriormente, mas essas variações ocorrem apenas na sintaxe, pois a semântica deve ser exatamente a mesma.
* A estrutura que empregaremos para a construção de nossos pseudocódigos será a seguinte:

***algoritmo*** *<nome do algoritmo>*

***var*** *<declaração das variáveis>*

***inicio*** *//marca o inicio das instruções*

*<comandos>*

***fimalgoritmo*** *//marca o final do algoritmo*

* Exemplo 1

Pseudocódigo que recebe um valor inteiro, fornecido pelo usuário, e o retorna no monitor.

*algoritmo "exemplo1"*

*var x: inteiro*

*inicio*

*leia (x)*

*escreva (x)*

*fimalgoritmo*

* Exemplo 2

Pseudocódigo de um algoritmo que recebe um valor inteiro, acresce duas unidades a este, e exibe o resultado desta manipulação.

*algoritmo "exemplo2"*

*var numero, resposta: inteiro*

*inicio*

*escreva (“Digite um número inteiro: ”)*

*leia (numero)*

*resposta* **←** *numero+2*

*escreva (“Resultado (número + 2): ”, resposta)*

*fimalgoritmo*

* Exemplo 3

Variação do algoritmo anterior.

*algoritmo "exemplo2\_1"*

*var n: inteiro*

*inicio*

*escreva (“Digite um número inteiro: ”)*

*leia (n)*

*escreva (“Resultado (número + 2): ”, n+2)*

*fimalgoritmo*

* Exercícios
  1. Construa um pseudocódigo para representar um algoritmo que efetue a multiplicação de dois inteiros quaisquer.
* *algoritmo “exer\_1"*
* *var x, y, resposta: inteiro*
* *inicio*
* *escreva (“Digite um primeiro multiplicador: ”)*
* *leia (x)*
* *escreva (“Digite um segundo multiplicador: ”)*
* *leia(y)*
* *resposta* **←** *x\*y*
* *escreva (“Resultado da multiplicação: “, resposta)*
* *fimalgoritmo*
* *algoritmo “exer\_1\_variacao"*
* *var x, y: inteiro*
* *inicio*
* *escreva (“Digite um multiplicando e em seguida ”)*
* *escreva (“um multiplicador (ambos inteiros): ”)*
* *leia(x, y)*
* *escreva (“Resultado da multiplicação: “, x\*y)*
* *fimalgoritmo*
* *algoritmo “exer\_1\_outra\_variacao"*
* *var x, y: inteiro*
* *inicio*
* *escreva (“Digite um multiplicando e em seguida ”)*
* *escreva (“um multiplicador (ambos inteiros): ”)*
* *leia(x, y)*
* *escreva (x,”\*”,y, “ = “, x\*y)*
* *fimalgoritmo*
  1. Gere um pseudocódigo que aplique um desconto de 30% sobre o valor de um produto, recebido como entrada, e retorne o resultado da manipulação na saída padrão.
* *algoritmo "exer\_2 "*
* *var valor: real*
* *inicio*
* *escreva (“Digite o valor do produto: ”)*
* *leia (valor)*
* *valor* **←** *valor\*0.7*
* *escreva (“Valor com desconto de 30%: “)*
* *escreva (valor)*
* *fimalgoritmo*

Faça um pseudocódigo de um algoritmo que receba 4 notas, calcule e mostre a média aritmética entre elas.

*algoritmo "exer\_3 "*

*var N1, N2, N3, N4: real*

*inicio*

*escreva (“Digite a primeira nota: ”)*

*leia (N1)*

*escreva (“Digite a segunda nota: ”)*

*leia (N2)*

*escreva (“Digite a terceira nota: ”)*

*leia (N3)*

*escreva (“Digite a quarta nota: ”)*

*leia (N4)*

*escreva (“A média das notas é “, (N1+N2+N3+N4)/4)*

*fimalgoritmo*

*algoritmo "exer\_3\_1 "*

*var N1, N2, N3, N4, MEDIA: real*

*inicio*

*escreva (“Digite as quatro notas obtidas: ”)*

*leia (N1, N2, N3, N4)*

*MEDIA* **←** *(N1+N2+N3+N4)/4*

*escreva (“A média das notas é “, MEDIA)*

*fimalgoritmo*

* + 1. Elabore um algoritmo que calcule a área de um círculo.

1. *algoritmo "exer\_4 "*
2. *var RAIO, PI: real*
3. *inicio*
4. *escreva (“Digite o raio do círculo: ”)*
5. *leia (RAIO)*
6. *PI* **←** *3.14*
7. *escreva (“A área do círculo é “, PI\*RAIO^2)*
8. *fimalgoritmo*

* É uma representação gráfica de algoritmos onde formas geométricas diferentes implicam ações (instruções, comandos) distintos.
* Preocupam-se com detalhes de nível físico da implementação do algoritmo.
  + Ex: Operações de saída de vídeo
* De modo geral, o fluxograma se resume a um único símbolo inicial, por onde a execução do algoritmo começa, e um ou mais símbolos finais, que são pontos onde a execução do algoritmo se encerra.

Links úteis:

<https://www.devmedia.com.br/como-nao-escrever-codigo/38679>

<https://www.devmedia.com.br/programacao-o-que-e-algoritmo/38677>

**algoritmo** exercicio

var res, cont, x, n: inteiro

inicio

cont <- 1

res <- 0

x <- 2

n <- 4

enquanto (cont <= n) faca

res <- res\*x

cont <- cont + 1

escreva(res)

res <- res + 1

fimenquanto

escreva(res)

fim