INTRODUÇÃO Á LÓGICA DE PROGRAMAÇÃO

Autor

Tânia Martins Preto

Revisão

Diretoria de Educação Profissional e Tecnologia

CURITIBA
JULHO/2007

APRESENTAÇÃO

A lógica de programação é de fundamental importância para qualquer pessoa que deseja se dedicar à aprendizagem e estudo de linguagens de programação. A construção de programação visa instruir o computador para que ele realize determinadas tarefas de acordo com as nossas necessidades.

Os compiladores são ferramentas que auxiliam a construção de programas e aceitam instruções, desde que estas obedeçam a determinas regras e estejam dentro do universo conhecido pelo compilador. Algoritmos são programas em uma forma mais simples e próxima da linguagem humana. O objetivo da lógica é ajudar o estudante a organizar as idéias e escrevê-las de uma forma que seja compreensível para o computador, ou seja, construir algoritmos que posterior mente possam facilmente ser transformados em programas.

Este material tem como objetivo servir de apoio a um curso introdutório de lógica de programação. Apresentam-se aqui os principais conceitos relacionados ao desenvolvimento de algoritmos e programas, bem como uma introdução a uma linguagem estruturada de descrição de algoritmos. O aluno vai usar os conceitos da lógica de programação, tendo assim contato com as atividades do ato de programar através da construção de algoritmos simples.

Espera-se que ao final deste curso o aluno esteja parcialmente preparado para iniciar efetivamente seus estudos em linguagens de programação de propósito diversos.

SUMÁRIO

INTRODUÇÃO

Um computador em funcionamento pode ser visto como a união de duas partes que funcionam juntas: o hardware e o software. O hardware refere-se como a parte física composta de peças eletrônicas tais como dispositivos de entrada de dados, memória, placas, processadores e etc. O software corresponde à um conjunto de tarefas que vai utilizar toda essa parte física para seu funcionamento. Quanto melhor for o hardware, mais eficiente é o software, por outro lado, o software pode ser elaborado de forma a melhor utilizar as componentes de hardware. As duas partes trabalham em conjunto, uma facilitando o trabalho da outra.

O estudo da lógica de programação e algoritmos é de fundamental importância para o desenvolvimento de software, também chamado de programas. Programas fazem parte de nosso dia a dia e estão presentes em diversas situações como, por exemplo, no caixa eletrônico, em cadastro de clientes em uma loja, em *sites* de compras pela internet, no auxilio ao diagnostico de tomografias e exames médicos, dentre outros.

Esta apostila aborda os principais conceitos relacionados com lógica de programação e busca dar uma base inicial para quem queira seguir estudos nessa desafiante área da Computação.

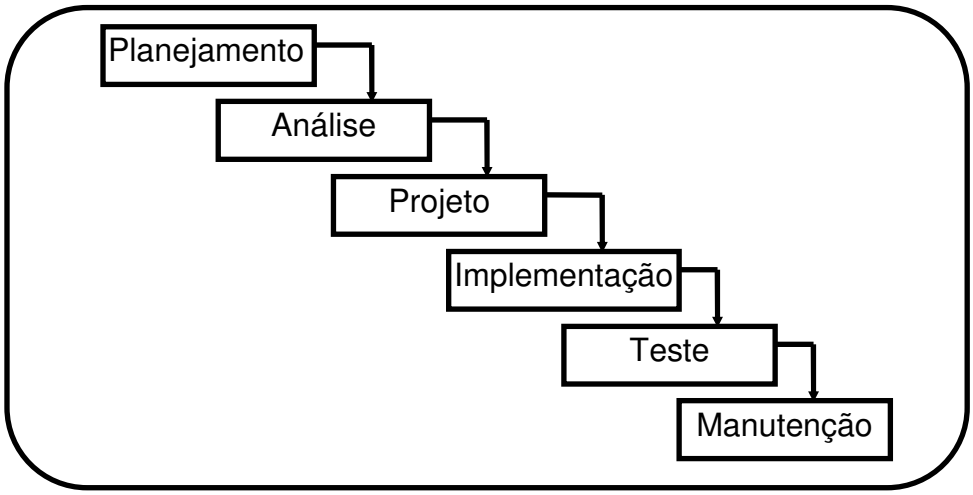
O capitulo 1 aborda os princípios básicos relacionados a lógica de programação e desenvolvimento de algoritmos, a saber: conceitos relacionados com as etapas de desenvolvimento de programas, lógica de programação e algoritmos.

1. CONCEITOS BÁSICOS

1.1 Desenvolvimento de Programas

A área de ciência da computação engloba duas subáreas de estudo que estão relacionadas ao desenvolvimento de programas: a engenharia de software e as linguagens de programação.

O desenvolvimento de software (programas) pode ser feito de várias formas. Na literatura existem diversas abordagens para o desenvolvimento de programas. Vamos considerar a seguinte representação, onde o processo de desenvolvimento é dividido em 6 etapas:



Etapas do processo de desenvolvimento de software.

Essas etapas são descritas a seguir:

- Planejamento: Define-se um plano inicial, considerando a abrangência do sistema, missão e objetivos, cronogramas, análise de custo x benefício e levantamento inicial de informações, dentre outros.

- Análise: Corresponde à análise de requisitos (necessidades) e definição de modelos, servindo de base para o processo de implementação do software.

- Projeto: Detalhes do projeto são especificados de forma a atender aos requisitos do sistema identificado na etapa de análise.Os aspectos computacionais são considerados e os algoritmos dos programas a serem implementados são construídos nesta fase.

- Implementação: É feita a transição dos algoritmos para a linguagem de programação, ou ainda, dizemos que ocorre a atividade de codificação.

- Teste: Necessários para verificar se o sistema está funcionando da forma correta. Todas as partes do sistema devem ser testadas por uma equipe de usuários.

- Manutenção: Essa fase pode durar vários anos. São ajustes e melhorias feitos de acordo com as necessidades. Os ajustes podem ser ocasionados por vários motivos: erros de projeto identificados após a implementação e o teste do software, inovações tecnológicas, novas necessidades e evolução do sistema, dentre outros.

1.2 Lógica de Programação

O estudo de lógica de programação é fundamental para o desenvolvimento de programas.

De forma geral, a **lógica** é uma ciência de índole matemática e fortemente ligada à filosofia. Assim, a lógica é o ramo da filosofia que cuida das regras do bem pensar, ou do pensar correto, sendo, portanto, um instrumento do pensar.

Pode-se dizer também que a lógica é a arte de pensar corretamente ou ainda que a lógica ensina a colocar ordem no pensamento.
Exemplo:
- Todo peixe nada.
- Nemo é um peixe
- Logo, Nemo nada.

Pela lógica deduzimos que Nemo nada.

Raciocínio é uma forma mais complexa de pensamento. Um sistema lógico é um conjunto de regras que visam representar formalmente o raciocínio válido.

O uso do raciocínio lógico adequado (e atitudes decorrentes deste) vai determinar o sucesso na execução de uma tarefa. Exemplos:

- Bons resultados numa prova escolar só serão obtidos se o assunto a ser tratado na prova for bem estudado.
- A construção de uma casa será bem sucedida de todos os devidos cuidados forem tomados ao longo de todo o processo, por exemplo: um bom projeto (estrutural, hidráulico e elétrico); equipe adequada (mestre de obras, pedreiros e ajudantes); material de construção de boa qualidade e etc.
- Para obter bons programas de computadores estes devem ser programados corretamente, ou ainda, de forma lógica. Só assim atenderá aos requisitos de forma correta e eficiente.

A **lógica de programação** corresponde ao raciocínio lógico empregado no desenvolvimento de programas, englobando um conjunto de elementos e regras de acordo com o tipo de programação.

Esses elementos e regras nos levam a encadear pensamentos para atingir os objetivos. Os pensamentos correspondem a seqüências de instruções.

Seqüência lógica são passos executados até atingir um determinado objetivo ou a solução de um problema.

O conceito de **instrução** corresponde a um conjunto de regras ou normas definidas para a realização ou emprego de algo.

Uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em **ordem seqüencial lógica**.
Mesmo ações simples devem executadas segundo uma ordem seqüencial lógica, ou ainda, seqüências lógicas. Por exemplo: “Tomar um sorvete”.

- .Pegar o sorvete;
- .Tirar o papel;
- .Tomar o sorvete;
- Jogar o papel (e o palito) no lixo.

Não daria certo, por exemplo, tomar o sorvete antes de tirar o papel.

1.3 Algoritmo

A seqüência lógica descreve como coisas ou processos devem ocorrer. São formadas por um conjunto básico de ações, também chamadas de passos ou instruções primitivas.

O conceito de algoritmo está fortemente associado a área de computação, no entanto pode ser aplicado a diversas áreas sendo sinônimo de processo, rotina ou procedimento.

Algumas definições para algoritmo, encontradas na literatura são as seguintes:

- Uma receita, uma seqüência de instruções que servem para realizar uma meta específica;
- Uma seqüência finita de passos que levam a execução de uma tarefa;
- Descrição de um conjunto padronizado de ações primitivas, bem definidas e executáveis, que encadeiam a realização de uma tarefa;
- Uma seqüência ordenada, finita e não ambígua de etapas que conduzem a solução de um problema;
- 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 obtenção do resultado ou da solução do problema.

O exemplo anterior de tomar um sorvete corresponde a um algoritmo simples. A seguir serão dados exemplos de algoritmos em diversos contextos.

Exemplo 1 - Contexto do dia a dia:Trocar uma lâmpada.

- Pegue uma escada;
- Coloque-a embaixo da lâmpada;
- Busque uma lâmpada nova;
- Suba na escada com a lâmpada nova;
- Retire a lâmpada velha;
- Coloque a lâmpada nova;
- Desça da escada.

Observa-se uma seqüência de comandos que devem ser executados na ordem sugerida, por exemplo, seria impossível retirar a lâmpada sem subir na escada.

Exemplo 2 - Contexto do dia a dia: Uma receita de bolo.

- Misture os ingredientes;
- Unte o tabuleiro com manteiga;
- Despeje a mistura no tabuleiro;
- **Se** (ha´queijo parmezão) **então**
  Espalhe sobre a mistura.
- Leve o tabuleiro ao forno.
- **Enquanto** (não dourar)
  Deixe o tabuleiro no forno.
- Deixe esfriar.

Observa-se que a ordem das instruções é de fundamental importância, por exemplo, não seria adequado despejar a mistura antes de untar o tabuleiro, isso acarretaria em erros no resultado esperado.

Nessa receita existe uma condição a ser testada (se existe queijo parmezão) que precisa ser satisfeita para a execução de uma ação (então espalhe sobre a mistura), as palavras **se** e **então** determinam o teste e a execução, respectivamente. Tem-se que a ação de espalhar sobre a mistura está subordinada ao fato de existir queijo parmezão, se não existir queijo, nada é feito.

Observa-se também uma condição de repetição (**enquanto** a massa não corar) que determina a execução repetida de uma ação (deixe o tabuleiro no forno).

Exemplo 3 - Contexto administrativo: Atendimento ao cliente.

- Verifique o preenchimento de um formulário
- **Se** (preenchimento correto)
  então
  Arquivar documento;
  Fornecer o protocolo;
  senão
  Adquirir outro formulário
  Fazer novo preenchimento.
- Despeça-se educadamente do cliente.

Verifica-se novamente uma condição de teste (se preenchimento correto) que determina duas ações, uma para o caso em que o teste fornecer resultado positivo (então) e outra para o caso em que o teste fornecer resultado negativo (senão)

Exemplo 4 - Contexto computacional: Somar dois números.

- Obtenha dois números
- Escreva o primeiro número no retângulo A
- Escreva o segundo número no retângulo B
- Some o número do retângulo A com número do retângulo B e coloque o resultado no retângulo C.

Neste exemplo existem apenas seqüências de comandos, não são feitas verificações de testes.

Exemplo 5 - Contexto computacional: Verificação de media de alunos.

- Obtenha as notas de duas provas de um aluno.
- Calcule a média aritmética
- **Se** (média maior ou igual a 5.0)
  então
    escreva a mensagem “APROVADO”;
  senão
    escreva a mensagem “REPROVADO”;
- Deseje boas férias ao aluno.

Verifica-se também a condição de teste (se média maior ou igual a 7.0) que determina duas ações, uma para o caso em que o teste fornecer resultado positivo (então) e outra para o caso em que o teste fornecer resultado negativo (senão).

1.4 Programas

Algoritmos nada mais são do que programas escritos de uma forma mais simples. Para que estes sejam realmente interpretados e compilados pelo computador a fim de ser tornarem programas independentes e utilizáveis, é necessário que os algoritmos sejam transformados em programas, isto é, sejam “traduzidos” para

uma linguagem de programação. Sendo assim, a linguagem de programação traduz o algoritmo para uma forma de linguagem entendida pela máquina, o algoritmo passa a ser então um programa.

Existem diversas linguagens de programação, por exemplo: Pascal, C, Cobol, Fortran, Visual Basic, Java, dentre outras. Algumas são muito parecidas entre si, outras nem tanto. No geral, as linguagens são bem mais cheias de detalhes e regras que os algoritmos, qualquer erro por menor que seja, compromete o funcionamento. Por esse motivo, a etapa de testes no desenvolvimento de programas é de fundamental importância.

Para auxiliar o correto desenvolvimento de programas executa-se também testes com os elementos presentes. O esquema chamado de “chinês” é uma maneira simples e eficiente de testar se os resultados estão corretos e será utilizado ao longo deste curso.

A seguir veremos o exemplo do algoritmo que faz a soma de dois números traduzido para linguagens de programação Pascal e C. Observa-se que essas duas linguagens em especial, apresentam estrutura semelhante, sendo também muito parecidas com o algoritmo original. No entanto, essa semelhança nem sempre ocorre com todas as linguagens.

Exemplo 1 – Soma de dois números:

Em algoritmo	Em Linguagem Pascal	Em linguagem C

algoritmo "soma"
var a, b, c: inteiro
inicio
  a <- 2
  b <- 3
  c <- a + b
  escreva (c)
fimalgoritmo
	program soma;
	uses Crt;
	var a, b, c: integer;
	begin
	  a := 2;
	  b := 3;
	  c := a + b;
	  write (c);
	end.
		#include <stdio.h>
		int a, b, c;
		void main()
		{
		  a= 2;
		  b= 3;
		  c= a+b;
		  printf(“%d”, c);
		}

Exemplo 2 – Cálculo da média de 2 notas

Em algoritmo	Em Linguagem Pascal	Em linguagem C

algoritmo "media2n"
 	
var nota1, nota2 : real
media: real
inicio
leia(nota1)
leia(nota2)
media <- (nota1 + nota2)/2
se media >=5 entao
escreva ("aprovado")
senao
escreva ("reprovado")
fimse
fimalgoritmo
	program media2n;
	uses Crt;
	var nota1, nota2 : real;
	media: real;
	begin
	readln(nota1);
	readln(nota2);
	media= (nota1 + nota2)/2;
	if media >=5 then
	begin
	write ('aprovado');
	end
	else
	begin
	write (' reprovado');
	end;
	end.
		#include <stdio.h>
		
		float nota1, nota2;
		float media;
		void main( )
		
		{
		scanf("%f", &nota1);
		scanf("%f", &nota2);
		media = (nota1+nota2)/2;
		if (media >=5)
		printf(" aprovado"
		else
		printf ("reprovado");
		}

1.5 Estruturas Básicas de Controle

Existem algumas estruturas que são usadas nos algoritmos e programas, constituindo formas de agregar e organizar as instruções primitivas. Essas estruturas controlam a maneira de executar as tarefas, facilitando a organização das idéias.

Existem três tipos de estruturas básicas: seqüenciação, seleção e repetição. Estes já foram citados no item 1.3, colocaremos aqui brevemente sua descrição e estes conceitos serão explorados com mais detalhes ao longo da apostila.

- **Seqüenciação**: Forma de agregar as instruções uma após a outra, de acordo com a ordem que devem ser executadas.

  - Exemplo 1
    Misture os ingredientes
    Unte o tabuleiro com manteiga
    Despeje a mistura no tabuleiro

- Exemplo 2:
  Obtenha dois números
  Escreva o primeiro número no retângulo A
  Escreva o segundo número no retângulo B

- **Seleção**: A partir do teste de uma condição, executa-se ou não uma tarefa, ou em algumas situações, executa-se uma tarefa ou outra tarefa. O teste da condição sempre resulta em verdadeiro ou falso, determinando o que fazer.

  - Exemplo 1:
    Se (ha´queijo parmezão) então
      Espalhe sobre a mistura

  - Exemplo 2:
    Se (média maior ou igual a 5.0)
      então
        escreva a mensagem “APROVADO”;
      senão
        escreva a mensagem “REPROVADO”;

- Repetição: semelhante ao anterior, no entanto, enquanto o teste condição estiver com resultado verdadeiro, as ações subordinadas são repetidas, diversas vezes. A partir do momento que a condição se torna falsa, o processo é interrompido.

  - Exemplo:
    Enquanto (não dourar)
      Deixe o tabuleiro no forno.

Todas essas estruturas são muito usadas e possuem diversas maneiras para serem representadas, tanto na forma de algoritmos como nas diversas linguagens de programação.

1.6 Fases do Processamento

Ao se observar a execução dos algoritmos e programas diversos, verifica-se que os mesmos são compostos de 3 fases fundamentais:

- **Entrada:** São as informações (dados) fornecidas ao algoritmo;
- **Processamento:** São os cálculos e procedimentos necessários para a atingir o resultado;
- **Saída:** São os dados já processados.

A seguir serão mostrados exemplos de cada uma das fases de processamento, em situações diversas.

Exemplo1 – Bolo

- Entrada: Ingredientes, tabuleiro, manteiga;
- Processamento: Misturar os ingredientes, verificar se há parmesão, verificar se o bolo dourou;
- Saída: Bolo pronto.

Exemplo 2 – Somar 2 números

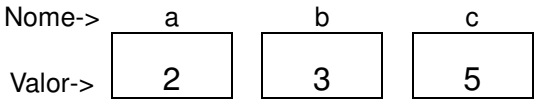
- Entrada: números valores de a e b;
- Processamento: soma de a com b, armazenamento do resultado em c;
- Saída: Mostrar o valor de c (resultado).

1.7 Memória

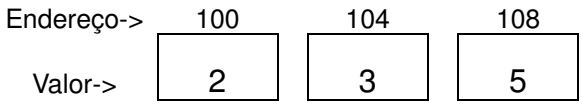
Durante a execução de um programa, tantos os dados quanto as instruções ficam armazenadas na memória do computador. A memória é dividida em partes ou elementos que são acessadas através de seu endereço.

Os dados dos programas e algoritmos ficam armazenados na memória e geralmente recebem um nome, por exemplo, a e b que podem receber os valores 2 e 3 respectivamente (a vale 2 e b vale 3). Esses nomes servem para acessar o valor que fica armazenado na memória.

Valores armazenados na memória podem ser visualizados como uma seqüência de “caixinhas”, uma ao lado da outra, onde cada caixinha possui um nome e também um endereço. A figura abaixo mostra alguns elementos armazenados na memória, seus respectivos nomes e valores.



A seguir, verifica-se na figura, os valores de alguns elementos e endereço de memória onde os mesmos se encontram:



Pode-se dizer que o dado de nome **a**, contém o valor **2** e ocupa a posição **100** da memória. Da mesma forma, o dado de nome **b**, contém o valor **3** e ocupa a posição **104** da memória e também o dado de nome **c**, contém o valor **5,** que foi resultado da soma de **a** com **b** e ocupa a posição **100** da memória.

2 REPRESENTAÇÃO DE ALGORITMOS

Existem algumas formas de representar um algoritmo, independente do contexto. Essas formas visam padronizar a representação e ajudar o entendimento de como ações devem interagir entre si.

Temos as seguintes representações: Linguagem Estruturada; fluxograma e diagrama estruturado. A primeira representação será adotada ao longo do curso, pois é que mais se aproxima das linguagens de programação, no entanto as outras também serão abordadas pois seu entendimento colabora para a prática da lógica de programação.

2.1 Linguagem Estruturada

A linguagem estruturada, também chamada de português estruturado, PDL (*Program Design Language*) ou ainda pseudo-código e pseudo-linguagem, vem sendo amplamente utilizado por projetistas de software e programadores, pois obriga o uso de estruturas que facilitam o entendimento do algoritmo, e também facilitam a transformação do mesmo em programas.

A partir do próximo capítulo este tema será abordado em detalhes, sendo aqui feita apenas uma pequena introdução a fim de compara com as outras formas de representação.

A linguagem estruturada é composta de comandos escritos segundo regras, utilizando apenas palavras para as estruturas básicas de controle e demais comandos.

Algumas regras simples são as seguintes:

- Escrever a palavra algoritmo e o nome do algoritmo (escolhido pelo programador);
- Colocar as palavras **inicio** antes do algoritmo e **fimalgoritmo** no final do mesmo;
- Dizer quais os valores que vão fazer parte do algoritmo e que tipo de números que eles vão manipular, por exemplo, a, b e c são números inteiros, logo escreve-se: **var a, b, c: inteiro**
- Usar o símbolo "<-" como atribuição, por exemplo, a recebe 2 é escrito como **a <- 2**;
- Usar a palavra **escreva(c)** para mostrar o resultado da soma na tela de saída.

A partir dessas regras, o exemplo da soma de dois números pode então ser representado em linguagem estruturada ou pseudocódigo.

Exemplo 1 - soma de dois números:

```
algoritmo "soma"
var a, b, c: inteiro

inicio

a <- 2
b <- 3
c <- a + b
escreva (c)

fimalgoritmo
```

Exemplo 2- média de 2 notas:

```
algoritmo "media2n"

var nota1, nota2 : real
    media: real
inicio
leia(nota1)
leia(nota2)
media <- (nota1 + nota2)/2

se media >=5  entao
                    escreva (" aluno aprovado")
                senao
                    escreva (" aluno reprovado")
fimse
fimalgoritmo
```

2.2 Diagrama Estruturado

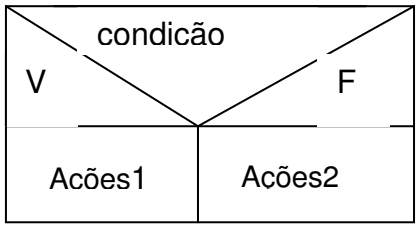
Também conhecido como diagrama Chapin e diagrama de Nassi-Shneiderman, utiliza um retângulo com subdivisões para representar as ações de um algoritmo. A utilização de diagramas para algoritmos maiores deve ser feita com cuidado a fim de não tornar a representação confusa.

As estruturas de sequenciação, seleção e repetição possuem uma notação especifica para sua representação, sendo mostradas a seguir:

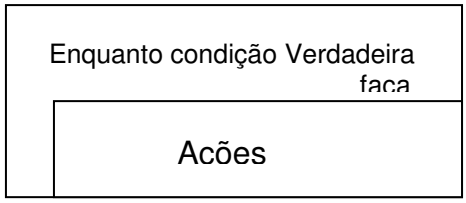
Seqüênciação

ação-1
ação-2
ação-n

Seleção



Repetição

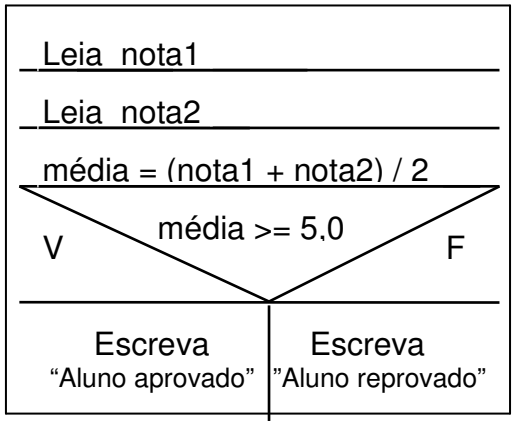


A seguir veremos os exemplos de soma de dois números e calculo de média através dessa representação.

Exemplo 1 – soma de dois números:

a ← 2
b ← 3
c ← a+ b
Escreva(c)

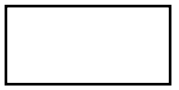
Exemplo 2 - media de notas:

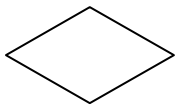


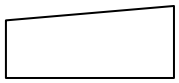
2.3 Fluxograma

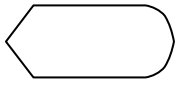
O fluxograma foi utilizado por muito tempo, porém não é adequado para algoritmos estruturados, como a maior parte das novas linguagens são estruturadas, então não é mais tão utilizado. No entanto para algoritmos simples representa uma boa ferramenta, pois facilita seu entendimento.

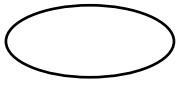
O fluxograma é composto por objetos gráficos para a representação de algoritmos. Retângulos losangos e outros objetos gráficos são usados, podendo variar alguns detalhes dessa notação. Os símbolos são unidos por setas que indicam para onde vai o fluxo, sendo que os objetos gráficos mais usados são os seguintes:


Ação


Decisão

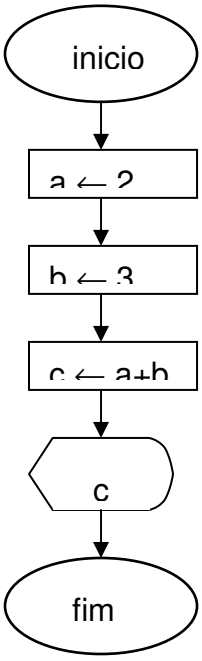

Leitura


Escrita

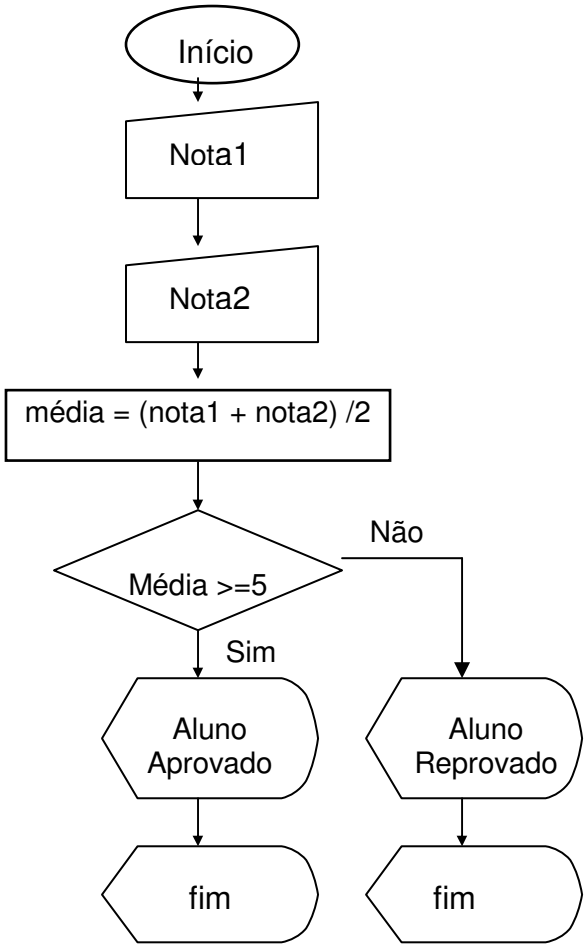

Inicio e Fim

A seguir veremos os exemplos de soma de dois números e calculo de média através dessa representação.

Exemplo 1 – soma de dois números:



Exemplo 2 - - media de notas:



3. ELEMENTOS BÁSICOS

A seguir serão vistos os principais elementos básicos para a construção de algoritmos.

3.1 Estrutura básica de um algoritmo

Os algoritmos possuem uma estrutura básica para a sua organização. Existem as chamadas palavras reservadas que são palavras da linguagem que possuem um significado próprio pré-definido e não devem ser usadas para outras finalidades.

A estrutura básica define uma ordem que deve sempre ser seguida Os principais componentes são:

- Nome do algoritmo: palavra reservada **algoritmo** seguindo do nome entre aspas;
- Bloco de definições e declaração de variáveis globais: palavra reservada **var**, seguida dos nomes das variáveis e de seu tipo (ex.: inteiro);
- Palavra reservada **inicio** que define de fato o inicio do algoritmo;
- Bloco de comandos que é composto por operações matemática, lógicas e as estruturas de controle;
- Fim do algoritmo definido pela palavra reservada **fimalgoritmo**. A partir desse ponto, qualquer outra informação é ignorada.

A figura abaixo ilustra essa idéia:

```
nome do algoritmo ---------------->   algoritmo "soma"
declaração de variáveis -------->     var a, b, c: inteiro
início do algoritmo---------------->  inicio
                                        a <- 2
bloco de comandos ---------------->     b <- 3
                                        c <- a + b
                                        escreva (c)
fim do algoritmo ------------------>  fimalgoritmo
```

Essa estrutura pode apresentar algumas variações e ter mais elementos acrescentados.

3.2 Dados

Dados são os objetos básicos da linguagem, destacando-se as constantes e as variáveis. Os dados possuem um nome, chamado de identificador e pertencem a um determinado tipo de acordo com os valores a serem manipulados.

Identificadores

Corresponde aos nomes utilizados para variáveis, constantes e funções. Essa escolha de identificadores deve seguir algumas regras, sendo que não pode haver repetição de nome. Existem algumas regras para escolher os identificadores:

- Não usar palavras reservadas, como por exemplo: inicio, se, então, senão, escreva;
- Devem iniciar com letras, e depois conter letras, números ou *underline (_)*, até um limite de 30 caracteres, no entanto, identificadores longos demais podem dificultar a digitação;
- Os identificadores devem esclareçam a finalidade das variáveis. Exemplos.: nota1, nota2, media, total, soma, nome, matricula, CPF, etc.

Tipos de dados

Determinam o conteúdo de uma variável. Os tipos básicos (simples) são:

- inteiro: para variáveis numéricas contendo números inteiros, ou seja, sem casas decimais;
- real: define variáveis numéricas contendo números reais, ou seja, com casas decimais;
- caractere: para variáveis composta por um ou mais caracteres (letras e alguns símbolos);
- logico: define variáveis que podem conter apenas dois valores: VERDADEIRO ou FALSO.

Os dados podem ser simples, contendo apenas um único valor, ou podem ser estruturados, contendo vários valores. Como exemplo de dados estruturados, tem-se as matrizes que são muito usadas em matemática. Uma matriz de inteiros de tamanho 3X4 (3 linhas e 4 colunas) contém 12 valores armazenados.

Variáveis

Podem conter diversos valores ao longo de sua existência. Devem ser declaradas antes de seu uso no bloco de declarações. Após a declaração, a variável passa a existir na memória. A declaração é iniciada através da palavra reservada **var**, seguida do nome de uma ou mais variáveis, dois pontos e o tipo dessas variáveis. Cada tipo deve estar em uma linha.

Exemplo 1 – três variáveis inteiras:

```
var a, b, c : inteiro;
```

Exemplo2 - n é inteiro, nota1 e nota2 é real, código é caractere e sinalizador é do tipo lógico.

```
var   n:  inteiro
      nota1,  nota2:  real
      codigo:  caractere
      sinalizador:  logico
```

Entrada e Saída de Dados

Os dados podem ser fornecidos pelo usuário, ou seja, são digitados no teclado e são “lidos” pelo algoritmo. Existem um comando responsável pela leitura de dados, é o comando **leia**. Os valores lidos são armazenados na variável especificada dentro de parênteses, ou seja, a forma do comando é leia (<lista-de-variáveis>).

Exemplo 1:

```
leia (x);
```

Exemplo 2:

```
leia(nota1, nota2)
```

é a mesma coisa que

```
leia(nota1)
leia(nota2)
```

Após o processamento, os dados de saída devem ser exibidos (escritos) na tela do computador para que o usuário veja os resultados. Para fazer a exibição dos

dados, utiliza-se o comando **escreva**, seguido do nome da variável a ser escrita entre parênteses, ou seja, escreva(<variáveis>).

Exemplo 3:

escreva(media) -> escreve o valor da media

escreva("aprovado") -> escreve a palavra "aprovado"

escreva("Resultado = ", media) -> escreve a expressão "Resultado =" e o valor da media;

Exemplo 4:

escreva(nota1, nota2)

é a mesma coisa que

escreva(nota1)

escreva(nota2)

3.3 Operadores

Os operadores auxiliam na execução de operações matemáticas e comparações entre valores, sendo que os mais utilizados são os operadores de atribuição, os matemáticos, relacionais e lógicos. Os operadores relacionais e lógicos são muito usados na formação das estruturas de seleção e repetição nos algoritmos. A seguir serão descritos os principais operadores.

Atribuição

São responsáveis por atribuir valores às variáveis, através do símbolo "<-" que significa "recebe". As variáveis podem ser lidas ou podem receber valores ao longo do processamento. Vale ressaltar que os valores a serem atribuídos devem estar de acordo com o tipo estabelecido na declaração.

Exemplos:

n <- 3

nota1 <- 7.5

código -< " X"

sinalizador <- FALSO

Matemáticos

Os operadores matemáticos fazem operações entre dois números, produzindo como resultado um terceiro número.

Operadores Matemáticos		
Símbolo	**Ação**	**Exemplo**
+	soma	2 + 3 = 5
-	subtração	5 –2 = 3
*	multiplicação	5*2 = 10
/	divisão	7/2 = 3.5
\	divisão inteira	7\2 = 3
MOD	resto da divisão inteira	7 MOD 2 = 1
^	potenciação	5^2 = 25

Relacionais

Os operadores relacionais fazem comparações entre dois valores, produzindo como resultado um valor que pode ser verdadeiro ou falso. Por exemplo, a expressão (a > b) é entendida como uma pergunta (a é maior que b?) e pode ter resposta VERDADEIRO ou FALSO.

Operadores Relacionais		
Símbolo	**Ação**	**Exemplo**
=	igual	2 = 3 (FALSO)
>	maior	5 > 2 (VERDADEIRO)
<	menor	5<2 (FALSO)
>=	maior ou igual	3>=3 (VERDADEIRO)
<=	menor ou igual	2<=3 (VERDADEIRO)
<>	diferente	5<>7 (VERDADEIRO)

Lógicos

Os operadores lógicos unem as respostas de expressões relacionais, também produzindo como resultado um valor lógico que pode ser verdadeiro ou falso. Por exemplo, a expressão (x > a **e** x < b) é entendida como uma pergunta (x é maior que a **e** x é menor que b ?) e pode ter resultado VERDADEIRO ou FALSO.

Operadores Lógicos		
Símbolo	**Ação**	**Exemplo**
ou	escolhe	7 > 5 e 2 >5 (FALSO)
e	une	5 > 7 ou 5 >2 (VERDADEIRO)
nao	nega	nao (5 > 2) (FALSO)

4. ESTRUTURAS DE CONTROLE

As estruturas ou comandos de controle podem ser divididos em seqüenciação, seleção e repetição, sendo que os dois últimos servem para ações mais elaboradas. A seguir serão vistas as estruturas de seleção e repetição.

4.1 Estruturas de Seleção

As estruturas de seleção, também chamadas de estruturas de decisão, executam testes através dos operados relacionais e lógicos a fim de decidir qual ação (ou conjunto de ações) deve ser tomada. Essas estruturas são obtidas através dos chamados comandos de desvio condicional para tomar decisões. São eles o se-entao , o se-entao-senao e o escolha.

Se-entao

Também chamado de comando de decisão simples. Se o resultado da expressão for verdadeiro, os comandos serão executados, caso contrário, nada é feito. O comando é formado pelas palavras reservadas se, então e fimse, sendo escrito da seguinte forma:

```
se   <expressão-lógica-verdadeira> entao
        <comando(s)>
fimse
```

Exemplo 1:

```
se media >=5  entao
                    escreva (" aluno aprovado")
fimse
```

Exemplo 2 – Algoritmo para ler dois números e calcular sua divisão, mas so´se o denominador for diferente de zero:

```
algoritmo " divisao"
var
  res, num, den :real
inicio
  escreva(" Digite dois numeros para divisão: ")
  leia(num)
  leia(den)
  se den <> 0 entao
       res <- num/den
       escreva("Resultado = ", res)
  fimse
fimalgoritmo
```

Se-entao-senao

Também chamado de comando de decisão dupla, determinando dois tipos de ações, uma para quando o resultado da expressão for verdadeiro (após entao) e outra para quando o resultado da expressão for falso (após senao). O comando é formado pelas palavras reservadas se, então, senão e fimse, sendo escrito da seguinte forma:

```
se   <expressão-lógica-verdadeira> entao
                                  <comando(s)1>
                              senao
                                  <comando(s)2>
fimse
```

Exemplo 1:

```
se media >=5  entao
                      escreva (" aluno aprovado")
                  senao
                       escreva (" aluno reprovado")
fimse
```

Exemplo 2 – Algoritmo que lê o tipo de um objeto (triangulo ou retângulo), lê os valores de base e altura e calcula a área de acordo com o tipo do objeto, ou seja: área de triangulo é base*altura/ 2 e área de retângulo é base*altura:

```
algoritmo "area_triangulo_retangulo"

var
  base, altura, area :real
  tipo: caractere
inicio
    escreva(" Digite se o objeto é um triangulo ou um retangulo: ")
    leia(tipo)
    escreva(" Digite valor da base: ")
    leia(base)
    escreva(" Digite valor da altura: ")
    leia(altura)
    se tipo = "triangulo" entao
        area <- (base * altura)/2
    senao
        area <-base* altura
    fimse
    escreva("Area calculada = ", area)
fimalgoritmo
```

Escolha

Também chamado de comando de decisão múltipla, determinando diversas ações, de acordo com o valor de uma variável. O comando é formado pelas palavras reservadas escolha, caso, outrocaso e fimescolha, sendo escrito da seguinte forma:

```
    escolha  <expressão-de-seleção>
        caso <exp1a>, <exp2a>, ..., <exp na>
                <comandos-a>
        caso <exp1b>, <exp2b>, ..., <expmb>
                <comandos-b>
        caso <exp1c>, <exp2c>, ..., <exppb>
                <comandos-c>
    ...
    outrocaso
              < comandos-x>
    fimescolha
```

Exemplo – Algoritmo que lê o nome de um time e imprime se é time de um algoritmo completo usando o comando escolha:

```
algoritmo "Times"
   var time:   caractere
   inicio
   escreva ("Entre com o nome de um time de futebol: ")
   leia (time)
   escolha time
            caso "Flamengo", "Fluminense", "Vasco", "Botafogo"
                     escreval ("É time carioca.")
             caso  "São Paulo", "Palmeiras", "Santos", "Corinthians"
                     escreval ("É time paulista.")
            caso  "Coritiba", "Atletico", "Parana"
                     escreval ("É time paranaense.")

            outrocaso
                     escreval ("É de outro estado.")
   fimescolha
fimalgoritmo
```

4.2 Estruturas de Repetição

As estruturas de repetição, também chamadas de estruturas de iteração ou laços, executam testes através dos operados relacionais e lógicos a fim de decidir e controlar quantas vezes uma ação (ou conjunto de ações) deve ser executada. Essas estruturas são obtidas através dos comandos enquanto-faça e para-faça.

Enquanto-faca

Esta estrutura controla a repetição da seguinte forma: enquanto uma determinada condição estiver sendo satisfeita (ou seja, seu resultado for verdadeiro), um determinado conjunto de ações (comandos) é executado diversas vezes. O comando é formado pelas palavras reservadas enquanto, faca e fimenquanto, sendo escrito da seguinte forma:

```
enquanto <expressão-lógica-verdadeira> faca
            <seqüência-de-comandos>
fimenquanto
```

Exemplo – Algoritmo para escrever na tela os números de 1 a 5:

```
algoritmo " escreve_numeros_1_a_5"
   var  i: inteiro
   inicio
   i <- 1
   enquanto  i <= 5  faca
           escreval (i)
           i <- i+1
   fimenquanto
 fimalgoritmo
```

Observa-se que as ações entre as palavras faca e fimenquanto são as que serão repetidas diversas vezes, caracterizando um laço, ou seja, a execução volta diversas vezes para o mesmo ponto. A seguir será mostrado o diagrama chinês que ilustra a execução deste algoritmo:

Valor de i	Teste (i < 10 ?)	O que é escrito
1	VERDADEIRO	1
2	VERDADEIRO	2
3	VERDADEIRO	3
4	VERDADEIRO	4
5	VERDADEIRO	5
6	FALSO	

Para-faca

Esta estrutura produz um efeito parecido com o resultado da estrutura enquanto-faca, porém funciona da seguinte forma: o valor inicial de uma variável e o seu valor final é que controla o numero de repetições. O comando é formado pelas palavras reservadas para, de, ate, passo (opcional), faca e fimpara. O passo 1 está embutido no comando (a variável é aumentada de 1 em 1), devendo o passo ser incluído apenas quando o passo for diferente de 1. A estrutura é escrita da seguinte forma:

para <variável> de <valor-inicial> ate <valor-final> faca

Outra **o**pção, considerando o passo:

para <variável> de <valor-inicial> ate <valor-final> passo <incremento>] faca

Exemplo 1 – Algoritmo para escrever na tela os números de 1 a 5:

```
algoritmo " escreve_numeros_1_a_5"
   var  i: inteiro
   inicio

   para i de 1 ate 5  faca
         escreval (i)
   fimpara
fimalgoritmo
```

Exemplo 2 – Algoritmo para escrever na tela os números de 5 a 1:

```
algoritmo " escreve_numeros_5_a_1"
   var  i: inteiro
   inicio

   para i de 5 ate 1 passo -1 faca
         escreval (i)
   fimpara
fimalgoritmo
```

A seguir será mostrado o diagrama chinês que ilustra a execução deste algoritmo:

Valor de i	O que é escrito
5	5
4	4
3	3
2	2
1	1

5.DADOS ESTRUTURADOS: VETOR

5.1 Definição

Vetor é um tipo de dado estruturados que corresponde a um conjunto de dados simples, do mesmo tipo e agrupados sob o mesmo nome. Cada elemento (dado simples) é identificado por um índice. O vetor pode ter tamanho variado, por exemplo, 10 elementos inteiros, 20 elementos reais, sendo que os índices variam de 1 ate´o tamanho máximo do vetor.

5.2 Declaração

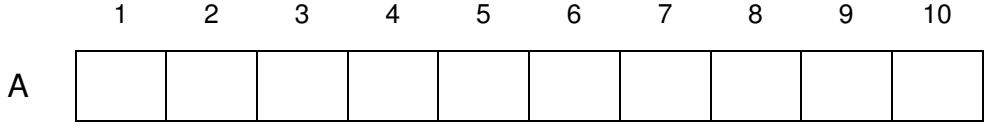
A declaração de um vetor é feita no bloco de declarações, porém contém informações referentes ao tamanho do vetor, ou seja, mostrando como seus índices variam, tendo a seguinte forma:

var <nome> : vetor[1.. <tamanho>] de <tipo>

Exemplo 1: - Declaração de um vetor de nome A contendo 10 elementos inteiros:

var A : vetor[1.. 10] de inteiro

A declaração acima criou uma seqüência de 10 inteiros na memória, que ocupam posições vizinhas, podendo ser visualizado da seguinte forma:



Para acessar os elementos do vetor, basta colocar seu nome seguido do índice, ou seja, A[1] corresponde ao primeiro, A[2] corresponde ao segundo e assim sucessivamente. Cada elemento do vetor é tratado como uma variável qualquer e pode ser utilizada para diversos tipos de operações.

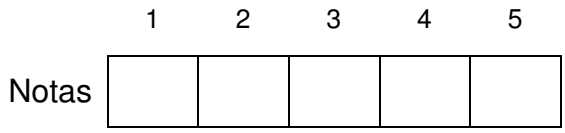
Exemplo 2- Atribuição de valores aos elementos do vetor.

```
leia (A[1])
A[2] <- 3
A[3] <- A[1] + A[2]
```

Exemplo 3 - Declaração de um vetor de nome Notas contendo 5 elementos reais:

```
var Notas : vetor[1.. 10 ] de  real
```

A declaração acima criou uma seqüência de 5 números reais na memória, podendo ser visualizado da seguinte forma:



5.3 Manipulação de Elementos

A manipulação de elementos do vetor envolve processos repetitivos onde muitas vezes apenas o índice varia. A utilização de estruturas de controle de repetição facilita diversas tarefas como leitura, escrita e atribuições diversas.

Exemplo 1 – Leitura de dados de um vetor contendo inteiros

Forma seqüencial:	Usando comando de repetição
leia(A[1])	para
leia(A[2])	para i de 1 ate 10 faca
leia(A[3])	leia(A[i])
leia(A[4])	fimpara
leia(A[5])	
leia(A[6])	enquanto
leia(A[7])	i<-0
leia (A[8])	enquanto i <= 10 faca
leia(A[9])	leia(A[i])
leia(A[10])	i <- i+1
	fimenquanto

O mesmo processo que foi feito para a leitura de elementos, pode ser feito para a escrita e outras ações. A utilização do comando para na manipulação de vetores facilita bastante diversas tarefas.

Exemplo 2 – Algoritmo para ler um vetor de inteiros, calcular e imprimir a média dos elementos.

```
algoritmo "Media_vetor"
var
   k : inteiro
   soma, media : real
   vet : vetor[1..10] de real
inicio
   escreval (" Digite 10 valores ")
   para k de 1 ate 10 faca
      leia(vet[k])
   fimpara
   soma <-0
   para k de 1 ate 10 faca
     soma <-soma + vet[k]
   fimpara
   media <-soma/10
   escreva ("Media calculada = ", media)
fimalgoritmo
```

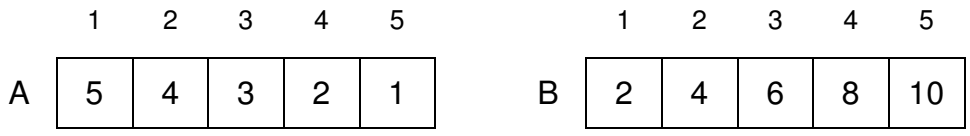
A seguir será mostrado o diagrama chinês que ilustra a execução deste algoritmo:

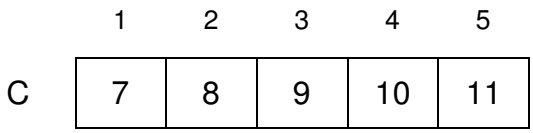
Valor de k	Valor de vet[k] digitado	soma	média
		0	
1	2	2	
2	5	7	
3	4	11	
4	3	14	
5	6	20	
6	10	30	
7	8	38	
7	7	45	
9	11	56	
10	4	60	60/10 = 6

Exemplo 3 – Algoritmo para ler dois vetores A e B, contendo 5 elementos cada um, calcular o vetor C que corresponde à soma de cada elemento de A com cada elemento de B de posições correspondentes:

```
algoritmo "Media_vetor"
var
   k : inteiro
   A, B, C : vetor[1..5] de real
inicio

escreval (" Digite 5 valores para vetor A: ")
para k de 1 ate 5 faca
   leia(A[k])
fimpara

escreval (" Digite 5 valores para vetor  B: ")
para k de 1 ate 5 faca
   leia(B[k])
fimpara

para k de 1 ate 5 faca
   C[k] <- A[k]+ B[k]
fimpara

escreval (" Vetor soma: ")
para k de 1 ate 5 faca
   escreval( C[k])
fimpara

fimalgoritmo
```

Supondo que tenha sido digitados os valores (5, 4, 3, 2,1) para A e (2, 4, 6, 8, 10) para B, tem-se a seguinte representação dos vetores após a execução dos comandos:





REFERÊNCIAS BIBLIOGRÁFICAS

APOIO INFORMÁTICA - Manual de utilização do Software VisuAlg - Editor e Interpretador de Algoritmos. http://www.apoioinformatica.inf.br/ Acesso em 01/07/2007

FARRER, H. *et al.*, **Algoritmos Estruturados**, Editora Guanabara, 1989.

FERNANDES, Antonio Luiz B.; BOTINI, Joana. **Construção de Algoritmos**. Editora Senac Nacional, 1998.

FORBELLONE, A. L. V., EBERSPÄCHER, H. F.; **Lógica de Programação – A Construção de Algoritmos e Estrutura de Dados**, Makron Books, 2000.

MARTINS, Luiz Eduardo G. ; ZÍLIO, Valéria Maria D'Arezzo. **Apostila**: Introdução à Programação, Unimep, 2003.

MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C - Módulo 1**, Editora MacGraw-Hill, 1990

MOARES, Paulo Sérgio; **Apostila**: Lógica de Programação, Unicamp, 2000.

SCHIMITZ , Eber Assis Schimitz - **Pascal e Técnicas de Programação**, Editora LTC, 1985.