FIT - Faculdade de Informática de Taquara
Curso de Sistemas de Informação

Apostila de Lógica de Programação
- ALGORITMOS -

Profa. Flávia Pereira de Carvalho

Março de 2007

Sumário

Página

1 Introdução

Nesta apostila estudaremos **Lógica de Programação** e, para isto, é importante ter uma visão geral do processo de desenvolvimento de programas (softwares), visto que o objetivo final é ter um bom embasamento para a prática da programação de computadores [MAR03].

Para o desenvolvimento de qualquer programa, deve-se seguir basicamente as seguintes etapas, conhecidas como Ciclo de Vida do Sistema [BUF03]:

1) Estudo da Viabilidade (Estudos Iniciais)

2) Análise detalhada do sistema (Projeto Lógico)

3) Projeto preliminar do sistema (Projeto Físico)

4) Projeto detalhado do sistema (**Algoritmos**)

5) Implementação ou Codificação do sistema (na Linguagem de Programação escolhida)

6) Testes do sistema

7) Instalação e Manutenção do sistema

No desenvolvimento de um sistema, quanto mais tarde um erro é detectado, mais dinheiro e tempo se gasta para repará-lo. Assim, a responsabilidade do programador é maior na criação dos algoritmos do que na sua própria implementação, pois quando bem projetados não se perde tempo tendo que refazê-los, reimplantá-los e retestá-los, assegurando assim um final feliz e no prazo previsto para o projeto [BUF03].

Pode-se encontrar na literatura em informática várias formas de representação das etapas que compõem o ciclo de vida de um sistema. Essas formas de representação podem variar tanto na quantidade de etapas quanto nas atividades a serem realizadas em cada fase [MAR03].

Como pode-se observar, nesse exemplo de ciclo de vida de um sistema (com sete fases) apresentado acima, os algoritmos fazem parte da quarta etapa do desenvolvimento de um programa. Na verdade, os algoritmos estão presentes no nosso dia-a-dia sem que saibamos, pois uma receita culinária, as instruções de uso de um equipamento ou as indicações de um instrutor sobre como estacionar um carro, por exemplo, nada mais são do que algoritmos.

Um algoritmo pode ser definido como um conjunto de regras (instruções), bem definidas, para solução de um determinado problema. Segundo o dicionário Michaelis, o conceito de algoritmo é a "utilização de regras para definir ou executar uma tarefa específica ou para resolver um problema específico."

A partir desses conceitos de algoritmos, pode-se perceber que a palavra algoritmo não é um termo computacional, ou seja, não se refere apenas à área de informática. É uma definição ampla que agora que você já sabe o que significa, talvez a utilize no seu cotidiano normalmente.

Na informática, o algoritmo é o "projeto do programa", ou seja, antes de se fazer um programa (software) na Linguagem de Programação desejada (Pascal, C, Delphi, etc.) deve-se fazer o algoritmo do programa. Já um programa, é um algoritmo escrito numa forma compreensível pelo computador (através de uma Linguagem de Programação), onde todas as ações a serem executadas devem ser especificadas nos mínimos detalhes e de acordo com as regras de sintaxe[1] da linguagem escolhida.

[1] *Sintaxe*: segundo o dicionário Aurélio, é a parte da gramática que estuda a disposição das palavras na frase e a das frases no discurso, bem como a relação lógica das frases entre si. Cada Linguagem de Programação tem a sua sintaxe (instruções, comandos, etc) que deve ser seguida corretamente para que o programa funcione. O conjunto de palavras e regras que definem o formato das sentenças válidas chama-se de sintaxe da linguagem.

Um algoritmo não é a solução de um problema, pois, se assim fosse, cada problema teria um único algoritmo. Um algoritmo é um 'caminho' para a solução de um problema e, em geral, existem muitos caminhos que levam a uma solução satisfatória, ou seja, para resolver o mesmo problema pode-se obter vários algoritmos diferentes.

Nesta disciplina estudaremos os passos básicos e as técnicas para a construção de algoritmos através de três métodos para sua representação, que são alguns dos mais conhecidos. O objetivo ao final da disciplina, é que você tenha adquirido capacidade de transformar qualquer problema em um algoritmo de boa qualidade, ou seja, a intenção é que você aprenda a Lógica de Programação dando uma base teórica e prática suficientemente boa, para que você domine os algoritmos e esteja habilitado a aprender uma Linguagem de Programação posteriormente [BUF03].

Para resolver um problema no computador é necessário que seja primeiramente encontrada uma maneira de descrever este problema de uma forma clara e precisa. É preciso que encontremos uma seqüência de passos que permitam que o problema possa ser resolvido de maneira automática e repetitiva. Esta seqüência de passos é chamada de algoritmo [GOM04].

A noção de algoritmo é central para toda a computação. A criação de algoritmos para resolver os problemas é uma das maiores dificuldades dos iniciantes em programação em computadores [GOM04].

Uma das formas mais eficazes de aprender algoritmos é através de muitos exercícios. Veja na Tabela 1 algumas dicas de como aprender e como não aprender algoritmos:

Algoritmos **não se aprende**	Algoritmos **se aprende**
Copiando algoritmos	Construindo algoritmos
Estudando algoritmos prontos	Testando algoritmos

Tabela 1: Dicas de como aprender e como não aprender algoritmos

O aprendizado da Lógica é essencial para a formação de um bom programador, servindo como base para o aprendizado de todas as Linguagens de Programação, estruturadas ou não. De um modo geral esses conhecimentos serão de supra importância, pois ajudarão no cotidiano, desenvolvendo um raciocínio rápido [COS04].

2 Formas de Representação de Algoritmos

Os algoritmos podem ser representados de várias formas, como por exemplo:

a) Através de uma língua (português, inglês, etc.): forma utilizada nos manuais de instruções, nas receitas culinárias, bulas de medicamentos, etc.

b) Através de uma linguagem de programação (Pascal, C, Delphi, etc.): esta forma é utilizada por alguns programadores experientes, que "pulam" a etapa do projeto do programa (algoritmo) e passam direto para a programação em si.

c) Através de representações gráficas: são bastante recomendáveis, já que um "desenho" (diagrama, fluxograma, etc.) muitas vezes substitui, com vantagem, várias palavras.

Cada uma dessas formas de representar um algoritmo, tem suas vantagens e desvantagens, cabe a pessoa escolher a forma que melhor lhe convir. Nesta disciplina serão apresentadas três formas de representação de algoritmos (que são algumas das mais utilizadas), são elas:

- Diagrama de Nassi-Shneiderman (Diagrama de Chapin)
- Fluxograma (Diagrama de Fluxo)
- Português Estruturado (Pseudocódigo, Portugol ou Pseudolinguagem)

Não existe consenso entre os especialistas sobre qual é a melhor maneira de representar um algoritmo. Eu aconselho a utilização do Diagrama Nassi-Shneiderman, mais conhecido como Diagrama de Chapin, por achar que é a forma mais didática de aprender e representar a lógica dos problemas e durante a disciplina usarei esse diagrama nos exemplos e exercícios. Mas, fica a critério de cada um escolher a forma que achar mais conveniente ou mais fácil de entender. Nos próximos capítulos são apresentadas breves explicações sobre cada uma dessas três formas de representar algoritmos e alguns exemplos.

2.1 Diagrama Nassi-Shneiderman

Os Diagramas Nassi-Shneiderman, também conhecidos como Diagramas de Chapin, surgiram nos anos 70 [YOU04] [SHN03] [CHA02] [NAS04] como uma maneira de ajudar nos esforços da abordagem de programação estruturada. Um típico diagrama Nassi-Shneiderman é apresentado na Figura 1 abaixo. Como você pode ver, o diagrama é fácil de ler e de entender, cada "desenho" representa uma ação (instrução) diferente.

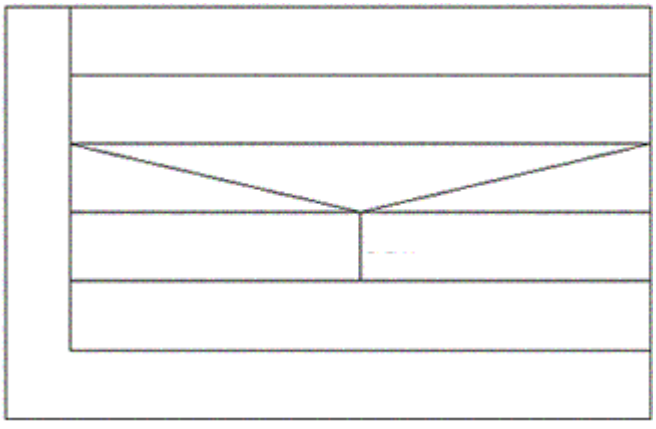


Figura 1: Exemplo de Diagrama Nassi-Shneiderman

A idéia básica deste diagrama é representar as ações de um algoritmo dentro de um único retângulo, subdividindo-o em retângulos menores, que representam os diferentes blocos de seqüência de ações do algoritmo. Para saber mais sobre o histórico desses diagramas e conhecer os seus criadores acesse o site: *http://www.cs.umd.edu/hcil/members/bshneiderman/nsd/*. Para ter acesso ao primeiro artigo elaborado pelos autores do Diagrama de Chapin, escrito em 1973, acesse o seguinte endereço onde você pode fazer o download do artigo: *http://fit.faccat.br/~fpereira/p12-nassi.pdf.*

2.2 Fluxograma

Os Fluxogramas ou Diagramas de Fluxo, são uma representação gráfica que utilizam formas geométricas padronizadas ligadas por setas de fluxo, para indicar as diversas ações (instruções) e decisões que devem ser seguidas para resolver o problema em questão.

Eles permitem visualizar os caminhos (fluxos) e as etapas de processamento de dados possíveis e, dentro destas, os passos para a resolução do problema. A seguir, na Figura 2, é apresentado um exemplo de fluxograma [GOM04] [MAR03].

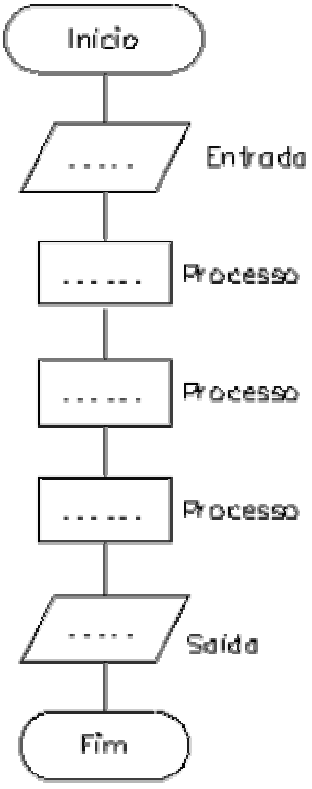


Figura 2: Exemplo de Fluxograma

2.3 Português Estruturado

O Português Estruturado, é uma forma especial de linguagem bem mais restrita que a Língua Portuguesa e com significados bem definidos para todos os termos utilizados nas instruções (comandos).

Essa linguagem também é conhecida como Portugol (junção de Português com Algol [ALG96] [PRO04]), Pseudocódigo ou Pseudolinguagem. O Português Estruturado na verdade é uma simplificação extrema da língua portuguesa, limitada a pouquíssimas palavras e estruturas que têm significado pré-definido, pois deve-se seguir um padrão. Emprega uma linguagem intermediária entre a linguagem natural e uma linguagem de programação, para descrever os algoritmos.

A sintaxe do Português Estruturado não precisa ser seguida tão rigorosamente quanto a sintaxe de uma linguagem de programação, já que o algoritmo não será executado como um programa [TON04].

Embora o Português Estruturado seja uma linguagem bastante simplificada, ela possui todos os elementos básicos e uma estrutura semelhante à de uma linguagem de programação de computadores. Portanto, resolver problemas com português estruturado pode ser uma tarefa tão complexa quanto a de escrever um programa em uma linguagem de programação qualquer só não tão rígida quanto a sua sintaxe, ou seja, o algoritmo não deixa de funcionar porque esquecemos de colocar um ';' (ponto-e-vírgula) por exemplo, já um programa não funcionaria. A Figura 3 apresenta um exemplo de algoritmo na forma de representação de português estruturado.

```
início
    <instruções>
    se <teste> então
        <instruções>
    senão
        <instruções>
    fim_se
fim
```

Figura 3: Exemplo de Português Estruturado

3 Conceitos Importantes

Neste capítulo são apresentados e explicados três conceitos fundamentais para a construção de algoritmos, são eles: Constante, Variável e Atribuição.

3.1 Constantes

São chamadas de constantes, as informações (dados) que não variam com o tempo, ou seja, permanecem sempre com o mesmo conteúdo, é um valor fixo (invariável). Como exemplos de constantes pode-se citar: números, letras, palavras etc.

3.2 Variáveis

O bom entendimento do conceito de variável é fundamental para elaboração de algoritmos e, conseqüentemente de programas. Uma variável, é um **espaço da memória do computador** que "reservamos" para guardar informações (dados). Como o próprio nome sugere, as variáveis, podem conter valores diferentes a cada instante de tempo, ou seja, seu conteúdo pode variar de acordo com as instruções do algoritmo.

As variáveis são referenciadas através de um nome (identificador) criado por você durante o desenvolvimento do algoritmo. Exemplos de nomes de variáveis: produto, idade, a, x, nota1, peso, preço, etc. O conteúdo de uma variável pode ser alterado, consultado ou apagado quantas vezes forem necessárias durante o algoritmo. Mas, ao alterar o conteúdo da variável, a informação anterior é perdida, ou seja, sempre "vale" a última informação armazenada na variável. Uma variável armazena 'apenas' um conteúdo de cada vez.

Uma variável pode ser vista como uma caixa com um rótulo (nome) colado nela, que em um dado momento guarda um determinado objeto. O conteúdo desta caixa não é algo fixo, permanente. Na verdade, essa caixa pode ter seu conteúdo alterado diversas vezes. No exemplo abaixo, a caixa (variável) rotulada como FATOR, contém o valor 5. Em outro momento essa caixa poderá conter qualquer outro valor numérico. Entretanto, a cada instante, ela conterá um, e somente um, valor [TON04].

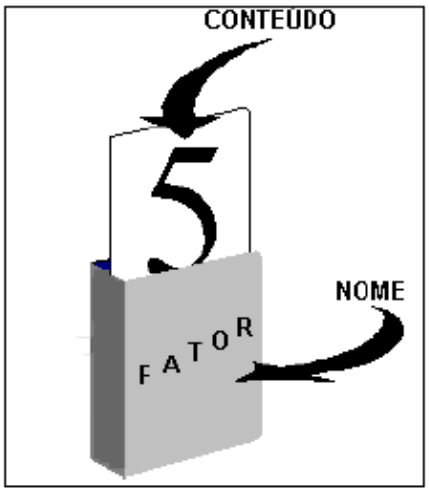


Figura 4: Ilustração de Variável [TON04]

3.3 Atribuição

A atribuição é uma notação utilizada para atribuir um valor a uma variável, ou seja, para armazenar um determinado conteúdo em uma variável. A operação de atribuição, normalmente, é representada por uma seta apontando para a esquerda, mas existem outros símbolos para representar a atribuição, depende da forma de representação do algoritmo. Na Tabela 2 a seguir, são apresentados alguns exemplos de atribuições possíveis:

Atribuições Possíveis	Exemplos
variável ⟵ constante	idade ⟵ 12 (lê-se: idade *recebe* 12)
variável ⟵ variável	preço ⟵ valor
variável ⟵ expressão	A ⟵ B + C

Tabela 2: Exemplos de Atribuições

Uma observação importante a ser feita em relação a atribuições é que na parte esquerda (a que vai "receber" algo) não pode haver nada além da variável, ou seja, é só variável que "recebe" algum conteúdo, não é possível ter um cálculo por exemplo, ou uma constante, recebendo alguma coisa. Veja por exemplo, esta notação:

nota1 + nota2 ⟵ valor

Esta operação apresentada acima não é possível, não está correta esta atribuição.

4 Instrução Escrever

Existem basicamente duas instruções principais em algoritmos (e em programação em geral) que são: Escrever e Ler. Neste capítulo veremos como funciona a instrução *Escrever*.

A instrução *Escrever* é utilizada quando deseja-se mostrar informações na tela do computador, ou seja, é um comando de **saída de dados**. Para simplificar, usa-se a instrução *Escrever*, quando necessita-se **mostrar algum dado para o usuário** do algoritmo (e posteriormente do programa).

Tanto no Diagrama de Chapin quanto em Português Estruturado representa-se a saída de dados através da palavra *Escrever* (ou Escreva). Já em Fluxogramas a representação da saída de dados é feita através de uma forma geométrica específica [GOM04] [MAR03].

Exemplos:

1) Escreva um algoritmo para armazenar o valor 20 em uma variável X e o valor 5 em uma variável Y. A seguir, armazenar a soma do valor de X com o de Y em uma variável Z. Escrever (na tela) o valor armazenado em X, em Y e em Z. *(Capítulo 17: Respostas dos Exemplos).*

2) Escreva um algoritmo para armazenar o valor 4 em uma variável A e o valor 3 em uma variável B. A seguir, armazenar a soma de A com B em uma variável C e a subtração de A com B em uma variável D. Escrever o valor de A, B, C e D e também escrever a mensagem 'Fim do Algoritmo'.

Observação:

Note que quando queremos **escrever alguma mensagem na tela** (letra, frase, número etc.) literalmente, devemos utilizar **aspas** para identificar o que será escrito, pois *o que estiver entre aspas no algoritmo, será exatamente o que aparecerá na tela do computador.* Diferente de quando queremos escrever o conteúdo de uma variável, pois neste caso não utiliza-se aspas.

5 Operadores Aritméticos

Muitas vezes, ao desenvolvermos algoritmos, é comum utilizarmos expressões matemáticas para a resolução de cálculos. Neste capítulo são apresentados os operadores aritméticos necessários para determinadas expressões. Veja a Tabela 3 a seguir.

Operação	Símbolo	Prioridade
Multiplicação (Produto)	*	1a.
Divisão	/	1a.
Adição (Soma)	+	2a.
Subtração (Diferença)	-	2a.

Tabela 3: Operadores Aritméticos

Nas linguagens de programação e, portanto, nos exercícios de algoritmos que iremos desenvolver, as expressões matemáticas sempre obedecem às regras matemáticas comuns, ou seja:

- As expressões dentro de parênteses são sempre resolvidas antes das expressões fora dos parênteses. Quando existem vários níveis de parênteses, ou seja, um parêntese dentro de outro, a solução sempre inicia do parêntese mais interno até o mais externo (de dentro para fora).
- Quando duas ou mais expressões tiverem a mesma prioridade, a solução é sempre iniciada da expressão mais à esquerda até a mais à direita.

Desta forma, veja os seguintes exemplos e os respectivos resultados:

ExemploA: 2 + (6 * (3 + 2)) = 32

ExemploB: 2 + 6 * (3 + 2) = 32

6 Instrução Ler

Como vimos no capítulo *4 Instrução Escrever*, existem basicamente duas instruções principais em algoritmos (e em programação em geral) que são: Escrever e Ler. No capítulo 4, foi apresentada a instrução *Escrever*, agora, neste capítulo, veremos como funciona a instrução *Ler*.

A instrução *Ler* é utilizada quando deseja-se obter informações do teclado do computador, ou seja, é um comando de **entrada de dados**. Para simplificar, usa-se a instrução *Ler*, quando necessita-se que **o usuário do algoritmo digite algum dado** (e posteriormente do programa).

Tanto no Diagrama de Chapin quanto em Português Estruturado representa-se a entrada de dados através da palavra *Ler* (ou Leia). Já em Fluxogramas a representação da entrada de dados é feita através de uma forma geométrica específica [GOM04] [MAR03].

Exemplo 3:

Escreva um algoritmo para ler dois valores e armazenar cada um em uma variável. A seguir, armazenar a soma dos dois valores lidos em uma terceira variável. Escrever o resultado da soma efetuada.

7 Horizontalização

Para o desenvolvimento de algoritmos que possuam cálculos matemáticos, as expressões aritméticas devem estar horizontalizadas, ou seja, linearizadas e também não esquecendo de utilizar os operadores corretamente. Na Tabela 4 a seguir, é apresentado um exemplo de uma expressão aritmética na forma tradicional e como deve ser utilizada nos algoritmos e em programação em geral (linearmente).

Matemática Tradicional	Algoritmo
$M = \frac{N1 + N2}{2}$	**M ← (N1 + N2) / 2**

Tabela 4: Exemplo de Horizontalização de Expressões Aritméticas

As expressões matemáticas na forma horizontalizada não são apenas utilizadas em algoritmos, mas também na maioria das linguages de programação.

8 Algoritmos com Seleção

Até agora estávamos trabalhando com algoritmos puramente seqüenciais, ou seja, todas as instruções eram executadas seguindo a ordem do algoritmo (normalmente, de cima para baixo). Neste capítulo começaremos a estudar estruturas de seleção. Uma estrutura de seleção, como o próprio nome já diz, permite que determinadas instruções sejam executadas ou não, dependendo do resultado de uma condição (teste), ou seja, o algoritmo vai ter mais de uma saída, uma opção que será executada de acordo com o teste realizado.

Exemplo 4:

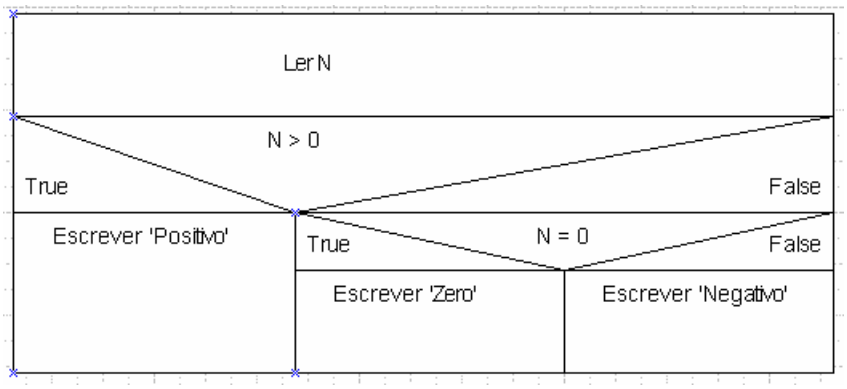
Escreva um algoritmo para ler um valor. **Se** o valor lido for igual a 6, escrever a mensagem *'Valor lido é o 6'*, caso contrário escrever a mensagem *'Valor lido não é o 6'*.

Quando estivermos utilizando algoritmos com seleção, podemos utilizar dois tipos de estruturas diferentes, dependendo do objetivo do algoritmo, chamadas de "Seleção Múltipla", cujos tipos são: Estrutura Aninhada e Estrutura Concatenada. Os capítulos 8.1 e 8.2 a seguir, apresentam estas duas estruturas com suas características.

8.1 Estrutura de Seleção Aninhada

A estrutura de seleção aninhada normalmente é utilizada quando estivermos fazendo várias *comparações (testes) sempre com a mesma variável*. Esta estrutura é chamada de aninhada porque na sua representação (tanto em Chapin quanto em Português Estruturado) fica *uma seleção dentro de outra seleção*.

Vamos utilizar a resposta do exercício número 27 da nossa lista de exercícios como exemplo destes dois tipos de estruturas. Abaixo é apresentada a resposta do exercício 27 em Chapin utilizando-se a estrutura aninhada:

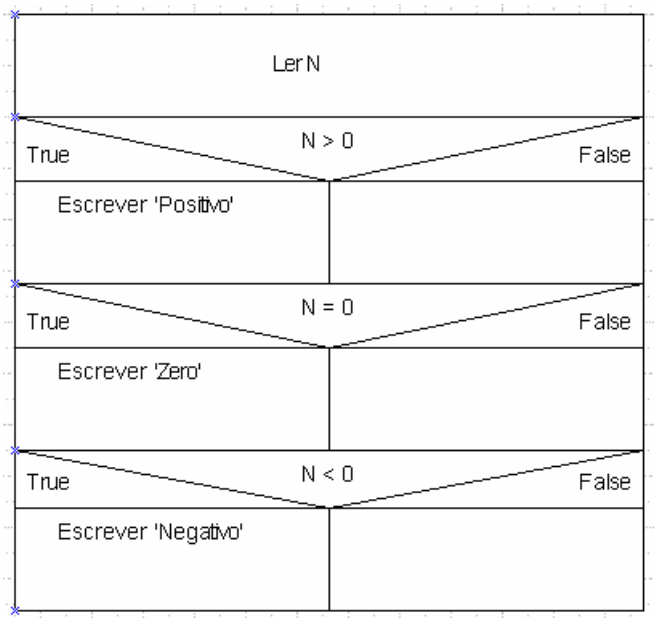


Exemplo de Estrutura de Seleção Aninhada: Resposta do Exercício 27

8.2 Estrutura de Seleção Concatenada

A estrutura de seleção concatenada normalmente é utilizada quando estivermos *comparando (testando) variáveis diferentes*, ou seja, independentes entre si. Esta estrutura é chamada de concatenada porque na sua representação (tanto em Chapin quanto em Português Estruturado) as seleções ficam separadas uma da outra (não existe o lado "falso" do Chapin, ou o "Senão" do Português).

Abaixo é apresentada a resposta do exercício número 27 da nossa lista de exercícios, utilizando a estrutura de seleção concatenada em Chapin:



Exemplo de Estrutura de Seleção Concatenada: Resposta do Exercício 27

Como pode ser observado nessas duas respostas apresentadas para o exercício 27 (estrutura aninhada e estrutura concatenada), existe uma grande diferença entre as duas estruturas, ou seja, uma característica de execução do algoritmo. Você saberia dizer qual é esta diferença?

9 Operadores Relacionais

Operações relacionais são as *comparações* permitidas entres valores, variáveis, expressões e constantes. A Tabela 5 a seguir, apresenta os tipos de operadores relacionais.

Símbolo	Significado
>	maior
<	menor
=	igual
> =	maior ou igual
< =	menor ou igual
< >	diferente

Tabela 5: Operadores Relacionais

A seguir, na Tabela 6, são apresentados alguns exemplos de comparações válidas:

Comparação Válida	Exemplo
variável e constante	X = 3
variável e variável	A < > B
variável e expressão	Y = W + J
expressão e expressão	(X + 1) < (Y + 4)

Tabela 6: Exemplos de Comparações Válidas

10 Operadores Lógicos

Os operadores lógicos permitem que mais de uma condição seja testada em uma única expressão, ou seja, pode-se fazer mais de uma comparação (teste) ao mesmo tempo. A Tabela 7 a seguir, apresenta os operadores lógicos que utilizaremos nesta disciplina.

Operação	Operador
Negação	**não**
Conjunção	**e**
Disjunção (não-exclusiva)	**ou**
Disjunção (exclusiva)	**xou** (lê-se: *"ou exclusivo"*)

Tabela 7: Operadores Lógicos

Note que a Tabela 7 acima, apresenta os operadores lógicos já ordenados de acordo com suas prioridades, ou seja, se na mesma expressão tivermos o operador **ou** e o operador **não**, por exemplo, primeiro devemos executar o **não** e depois o **ou**.

De uma forma geral, os resultados possíveis para os operadores lógicos podem ser vistos na Tabela 8 abaixo, conhecida como **Tabela Verdade**:

A	B	A e B	A ou B	não A	A xou B
F	F	F	F	V	F
F	V	F	V	V	V
V	F	F	V	F	V
V	V	V	V	F	F

Tabela 8: Possíveis Resultados dos Testes Lógicos - Tabela Verdade

Exemplos de Testes utilizando Operadores Lógicos:

➔ Exemplos usando operadores lógicos [KOZ06]:

Expressão	Quando eu não saio?
Se chover *e* relampejar, eu não saio.	Somente quando chover e relampejar ao mesmo tempo (apenas 1 possibilidade).
Se chover *ou* relampejar, eu não saio.	Somente quando chover, somente quando relampejar ou quando chover e relampejar ao mesmo tempo (3 possibilidades).
Se chover *xou* relampejar, eu não saio.	Somente quando chover, ou somente quando relampejar (2 possibilidades).

Exemplos A, B e C abaixo são apresentados em Português Estruturado:

A)
```
Se (salario > 180) e (salário < 800) Então
        Escrever 'Salário válido para financiamento'
Senão
        Escrever 'Salário fora da faixa permitida para financiamento'
FimSe
```

B)
```
Se (idade < 18) ou (idade > 95) Então
        Escrever 'Você não pode fazer carteira de motorista'
Senão
        Escrever 'Você pode possuir carteira de motorista'
FimSe
```

C)
```
Se (idade > = 18) e (idade < = 95) e (aprovado_exame = 'sim') Então
        Escrever 'Sua carteira de motorista estará pronta em uma semana'
Senão
        Escrever 'Você não possui idade permitida ou não passou nos testes'
FimSe
```

➔ **Exemplos de expressões utilizando operadores lógicos [KOZ06]**:

Para **A = V, B = F** e **C = F** , as expressões abaixo fornecem os seguintes resultados:

Expressão	Resultado
a) não A	não V = **F**
b) A e B	V e F = **F**
c) A ou B xou B ou C	(V ou F) xou (F ou F) = V xou F = **V**
d) não (B ou C)	não (F ou F) = não F = **V**
e) não B ou C	(não F) ou F = V ou F = **V**

Exemplo utilizando Operadores Lógicos em Chapin:

5) Escreva um algoritmo para ler um valor e escrever se ele está entre os números 1 e 10 ou não está.

Ao verificar as respostas (errada e correta) do exemplo número 5 acima, pode-se constatar que quando precisamos fazer mais de um teste ao mesmo tempo, deve-se utilizar os operadores lógicos apresentados neste capítulo na tabela 7.

11 Algoritmos com Repetição

Nos exemplos e exercícios que vimos até agora, sempre foi possível resolver os problemas com uma seqüência de instruções que eram executadas apenas uma vez. Existem três estruturas básicas para a construção de algoritmos, que são: algoritmos seqüenciais, algoritmos com seleção e algoritmos com repetição. A combinação dessas três estruturas permite-nos a construção de algoritmos para a resolução de problemas extremamente complexos [MAR03]. Nos capítulos anteriores vimos a estrutura puramente seqüencial e algoritmos com seleção (capítulo 8). Neste capítulo veremos as estruturas de repetição possíveis em algoritmos e existentes na maioria das Linguagens de Programação.

Uma estrutura de repetição permite que uma seqüência de instruções (comandos) seja executada várias vezes, até que uma condição (teste) seja satisfeita, ou seja, repete-se um conjunto de instruções sem que seja necessário escrevê-las várias vezes. As estruturas de repetição também são chamadas de Laços ou Loops [MAR03].

Para sabermos quando utilizar uma estrutura de repetição, basta analisarmos se uma instrução ou uma seqüência de instruções precisa ser executada várias vezes, se isto se confirmar, então deve-se utilizar uma estrutura de repetição. As estruturas de repetição, assim como a de decisão (seleção), envolvem a avaliação de uma condição (teste). Então as estruturas de repetição permitem que um trecho do algoritmo (conjunto de instruções) seja repetido um número determinado (ou indeterminado) de vezes, sem que o código correspondente, ou seja, as instruções a serem repetidas tenham que ser escritas mais de uma vez [TON04].

Exemplo 6:

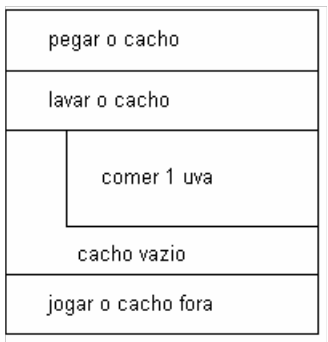
Escreva um algoritmo para comer um cacho de uva.

➔ A resposta apresentada (em Chapin e em Português Estruturado) está correta?

Na solução do exemplo 6 apresentada na Seção de Respostas dos Exemplos *(Capítulo 17)*, não foi utilizada uma 'estrutura de repetição', por isto **o algoritmo não está correto.** Nesse exemplo número 6 é necessária uma estrutura de repetição, pois a instrução *"comer 1 uva"* precisa ser repetida várias vezes.

Existem três tipos de estruturas de repetição: **Repita-Até**, **Enquanto-Faça** e **Para-Até-Faça**, cada uma com suas peculiaridades e apropriada para cada problema, normalmente é possível resolver um mesmo problema usando qualquer uma das estruturas de repetição, mas, na maioria das situações, haverá uma mais adequada. Neste capítulo veremos as características de cada uma destas estruturas. Mas antes, veja a resposta do exemplo número 6 utilizando uma estrutura de repetição (Repita-Até):

Chapin



Português Estruturado

```
Início
    pegar o cacho
    lavar o cacho
    repita
        comer 1 uva
    até cacho vazio
    jogar o cacho fora
Fim
```

11.1 Estrutura de Repetição: REPITA-ATÉ

Na estrutura Repita-Até as instruções a serem repetidas são executadas, no mínimo uma vez, já que o *teste (a condição) fica no final da repetição*. Nesta estrutura, a repetição é finalizada quando o teste for Verdadeiro (V), ou seja, o algoritmo fica executando as instruções que estiverem dentro do laço até que o teste seja verdadeiro. Nas Figuras 5 e 6 abaixo, é apresentada a forma geral da estrutura Repita-Até em Chapin e em Português Estruturado, respectivamente.

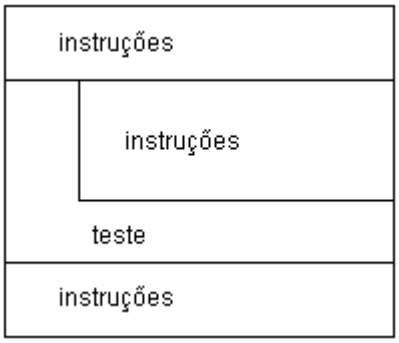


Figura 5: Estrutura Repita-Até em Chapin

```
Início
    instruções
    repita
        instruções
    até teste
    instruções
Fim
```

Figura 6: Estrutura Repita-Até em Português Estruturado

Observações da estrutura de repetição REPITA-ATÉ:

1) A repetição (o laço) se encerra quando a condição (teste) for verdadeira.

2) As instruções a serem repetidas são executadas pelo menos 1 vez, porque o teste é no final da repetição.

Pergunta: No exemplo 6 anterior, utilizando uma estrutura de repetição, que no caso utilizou-se a estrutura Repita, o algoritmo ficou correto?

11.2 Estrutura de Repetição: ENQUANTO-FAÇA

Na estrutura Enquanto-Faça as instruções a serem repetidas podem não ser executadas nenhuma vez, pois o *teste fica no início da repetição*, então a execução das instruções (que estão "dentro" da repetição) depende do teste. Nesta estrutura, a repetição é finalizada quando o teste é Falso (F), ou seja, enquanto o teste for Verdadeiro as instruções serão executadas e, quando for Falso, o laço é finalizado. Veja nas Figuras 7 e 8 abaixo a forma geral da estrutura Enquanto-Faça em Chapin e em Português Estruturado.

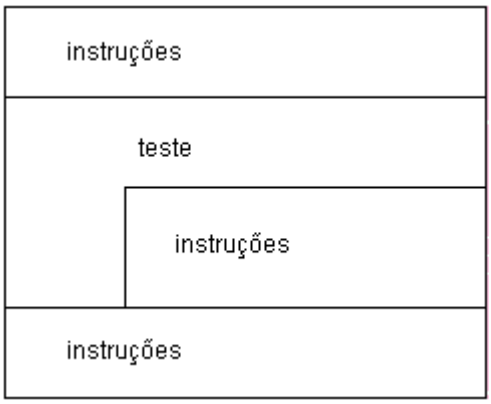


Figura 7: Estrutura Enquanto-Faça em Chapin

```
início
    instruções
    enquanto teste faça
        instruções
    fim_enquanto
    instruções
fim
```

Figura 8: Estrutura Enquanto-Faça em Português Estruturado

Observações da estrutura de repetição ENQUANTO-FAÇA:

1) A repetição (o laço) se encerra quando a condição (teste) for falsa.

2) As instruções a serem repetidas podem nunca ser executadas, porque o teste é no início da repetição.

Atenção: Resolva o exemplo 6 anterior, utilizando a estrutura de repetição Enquanto-Faça!

11.3 Estrutura de Repetição: PARA-ATÉ-FAÇA

A estrutura de repetição Para-Até-Faça é um pouco diferente das outras duas (Repita-Até e Enquanto-Faça), pois possui uma **variável de controle**, ou seja, com esta estrutura é possível executar um conjunto de instruções um número determinado de vezes. Através da variável de controle, define-se a quantidade de repetições que o laço fará.

Exemplo 7:

Escreva um algoritmo para escrever 5 vezes a palavra FACCAT na tela.

Funcionamento da estrutura PARA:

Na resposta do exemplo 7 acima, o X é a variável de controle, ou seja, uma variável qualquer (com qualquer nome) que vai determinar o número de repetições do laço. O valor 1 é o valor inicial que será atribuído à variável X e o valor 5 é o valor final atribuído à variável X, com isto, tem-se 5 repetições da instrução (ou das instruções) que estiver dentro do laço.

Cada vez que a variável é incrementada (aumenta +1) as instruções de dentro da repetição são executadas, então a variável, no caso o X, inicia com o valor 1 e a cada execução (repetição) ele aumenta +1 (é incrementado) até chegar ao valor final, que também é determinado (no caso é o 5).

ATENÇÃO: Você saberia dizer qual é a grande diferença entre a estrutura de repetição **Para** e as estruturas **Repita** e **Enquanto**?

12 Dizer SIM para Continuar ou NÃO para Finalizar

Exemplo 8:

Escreva um algoritmo para ler dois valores. Após a leitura deve-se calcular a soma dos valores lidos e armazená-la em uma variável. Após o cálculo da soma, escrever o resultado e escrever também a pergunta 'Novo Cálculo (S/N)?'. Deve-se ler a resposta e se a resposta for 'S' (sim), deve-se repetir todos os comandos (instruções) novamente, mas se a resposta for 'N' (não), o algoritmo deve ser finalizado escrevendo a mensagem 'Fim dos Cálculos'.

13 Contadores e Acumuladores

Em algoritmos com estruturas de repetição (Repita, Enquanto ou Para) é comum surgir a necessidade de utilizar variáveis do tipo contador e/ou acumulador. Neste capítulo são apresentados esses conceitos detalhadamente.

13.1 Contadores

Um contador é utilizado para contar o número de vezes que um evento (uma instrução) ocorre, ou seja, contar a quantidade de vezes que uma instrução é executada.

Forma Geral: **VARIÁVEL ← VARIÁVEL + CONSTANTE**

Exemplo: X ← X + 1

Explicação: um contador é uma variável (qualquer) que recebe ela mesma mais um valor (uma constante), no caso do exemplo acima, a variável X está recebendo o valor dela mesma mais 1. Normalmente a constante que será somada no contador é o valor 1, para contar de 1 em 1, mas pode ser qualquer valor, como por exemplo 2, se quisermos contar de 2 em 2.

Observações dos Contadores:

1) A variável (do contador) deve possuir um valor inicial conhecido, isto é, ela deve ser inicializada. Normalmente inicializa-se a variável do contador com zero, ou seja, zera-se a variável antes de utilizá-la. Para zerar uma variável basta atribuir a ela o valor zero: **VARIÁVEL ← 0**

2) A constante (que é geralmente o valor 1) determina o valor do incremento da variável (do contador), ou seja, o que será somado (acrescido) a ela.

Exemplo 9:

Escreva um algoritmo para ler a nota de 10 alunos e contar quantos foram aprovados, sendo que, para ser aprovado, a nota deve ser maior ou igual a 6,0. Escrever o número de aprovados.

ATENÇÃO: Se quiséssemos contar o número de reprovados também, no exemplo 9 acima, o que deveria ser feito?

13.2 Acumuladores (ou Somadores)

Um acumulador, também conhecido como Somador, é utilizado para obter somatórios (Σ).

Forma Geral: **VARIÁVEL1 ← VARIÁVEL1 + VARIÁVEL2**

Exemplo: X ← X + Y

Explicação: um acumulador (somador) é uma variável (qualquer) que recebe ela mesma mais uma outra variável, no caso do exemplo acima, a variável X está recebendo o valor dela mesma mais o valor da variável Y. A variável Y representa o valor a ser somado, acumulado na variável X.

Observações dos Acumuladores:

1) A variável1 (do acumulador) deve possuir um valor inicial conhecido, isto é, ela deve ser inicializada. Normalmente inicializa-se a variável do acumulador com zero, ou seja, zera-se a variável antes de utilizá-la. Para zerar uma variável basta atribuir a ela o valor zero: **VARIÁVEL1 ← 0**

2) A variável2 indica o valor a ser acumulado, somado e armazenado na variável1.

Exemplo 10:

Altere o exemplo 9 para calcular também a média geral da turma e, depois de calculada, escrever a média.

ATENÇÃO: Normalmente inicializa-se as variáveis que serão utilizadas como contador ou como acumulador com o valor zero, mas pode-se inicializá-las com o valor que desejarmos de acordo com a necessidade.

14 Determinação do MAIOR e/ou MENOR valor em um Conjunto de Valores

Em muitos algoritmos surge a necessidade de determinarmos qual o maior ou o menor valor dentro de um conjunto de valores e, para isto, não existe uma estrutura especial, apenas utilizamos os conhecimentos que já aprendemos, como mostrado no exemplo a seguir.

Exemplo 11:

Escreva um algoritmo para ler a nota de 10 alunos e escrever a nota mais alta, ou seja, a maior nota entre as 10 notas lidas.

IMPORTANTE: Quando sabe-se os limites dos valores possíveis, ou seja, por exemplo com as notas sabemos que os valores serão de 0 a 10, então sabe-se quais são os valores limites (o valor mínimo e o valor máximo), não teremos nota menor que 0 e nem nota maior que 10. Nesses casos é mais fácil descobrir o maior ou o menor valor, pois pode-se inicializar a variável Maior, por exemplo, com o valor 0 e a variável Menor com o valor 10 que funcionará perfeitamente. Acontece que se não sabe-se os valores dos limites aí complica um pouco, pois não sabemos com que valor vamos inicializar as variáveis para comparação. Então temos que inicializar tanto a variável Maior quanto a Menor com o "primeiro valor lido" e depois vamos comparando os próximos valores lidos com o primeiro! (os nomes "Maior" e "Menor", são apenas exemplos, pode-se denominar as variáveis que serão usadas para os testes como quiser).

15 Repetição Aninhada

Assim como vimos que é possível ter uma Seleção dentro de outra, também podemos ter uma Repetição dentro de outra, dependendo do problema a ser resolvido. Pode ser necessária uma estrutura de Repita dentro de um Enquanto, por exemplo, ou vice-versa. Ou um Repita dentro de outro Repita, enfim, as combinações são inúmeras. A seguir veremos um exemplo de uma estrutura de repetição Para dentro de outro Para, que é bastante utilizado para leitura e escrita de Matrizes, por exemplo.

Exemplo 12:

Dado o algoritmo a seguir (Figura 9), representado em Chapin, diga o que ele faz, ou seja, o que seria escrito na tela ao ser executado.

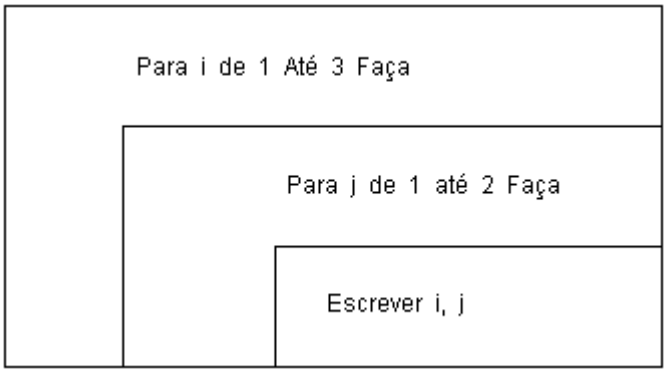


Figura 9: Exemplo de Repetição Aninhada

Funcionamento de "um PARA dentro de outro PARA":

1) A execução inicia pelo Para de fora (mais externo), depois desvia para o Para de dentro e só volta para o Para de fora quando terminar toda execução do Para de dentro (quando a variável de controle chegar no valor final).

2) Um Para fica "parado" enquanto o outro Para é executado, ou seja, enquanto sua variável de controle varia até chegar no valor final determinado para ela.

16 Vetores

Podemos definir um Vetor como uma variável dividida em vários "pedaços", em várias "casinhas", onde cada pedaço desses é identificado através de um número, referente à posição de uma determinada informação no vetor em questão. O número de cada posição do vetor é chamado de **índice**.

Conceito: Vetor é um conjunto de variáveis, onde cada uma pode armazenar uma informação diferente, mas todas compartilham o mesmo nome. São associados índices a esse nome, que representam as posições do vetor, permitindo assim, individualizar os elementos do conjunto.

Podemos imaginar que na memória do computador o vetor seja mais ou menos da seguinte forma:

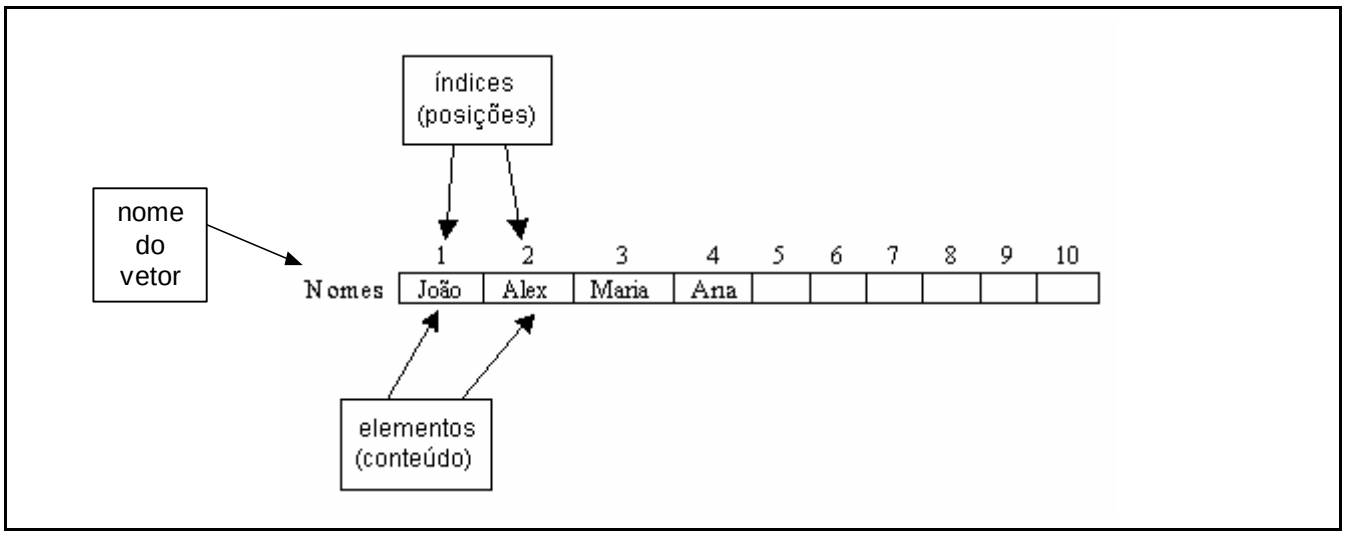


Figura 10: Exemplo de um Vetor na Memória do Computador

Exemplos de Manipulação de Vetores:

1) Escrever Nomes [3]

- Esta instrução escreve na tela, o conteúdo da Posição 3 do Vetor chamado Nomes, ou seja, escreve a palavra **'**Maria**'**.

ATENÇÃO: Sempre que estivermos nos referindo a um Vetor, devemos colocar o **Nome do Vetor** e a **Posição** (o Índice) correspondente entre *colchetes*.

2) Nomes [5] ← 'André'

- Esta instrução armazena (atribui) a palavra 'André' na posição 5 do Vetor chamado Nomes.

3) Escrever Nomes [X]

Esta instrução escreve o conteúdo da posição X do Vetor Nomes, ou seja, é possível utilizarmos variáveis para indicar a posição (o índice) do Vetor. Neste exemplo 3, o que será escrito depende do valor da variável X. Por exemplo, se antes da instrução Escrever, tivéssemos a instrução: **X ← 4**. Então seria escrito o conteúdo da posição 4 do Vetor Nomes, neste caso, seria escrita a palavra **'**Ana**'**.

4) Escrever Nomes [X-2]

Esta instrução é para mostrar que também pode-se utilizar cálculos (expressões) para indicar a posição (o índice). Digamos que o X tenha recebido o valor 4, como no exemplo 3 acima, então, neste caso, seria escrita a palavra **'**Alex**'**.

16.1 Como LER um Vetor (Preencher)

Para Ler um vetor, ou seja, para **preencher um vetor** com informações (dados) (armazenar informações em um vetor) é necessária uma estrutura de repetição, pois um vetor possui várias posições e temos que preencher uma a uma. A estrutura de repetição normalmente utilizada para vetores é o **Para-Até-Faça**, então veja no exemplo abaixo como preencher (ler) um vetor de 10 posições:

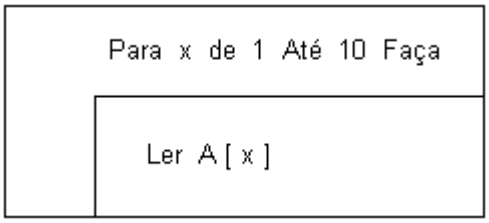


Figura 11: Exemplo para Ler um Vetor

Na Figura 11, acima, está demonstrado como preencher um vetor chamado A de 10 posições, ou seja, serão lidas 10 informações (valores, nomes, letras, etc.) e cada uma será armazenada em uma posição do vetor A. Sendo que utiliza-se a própria variável da repetição Para para representar a posição (índice) do vetor.

16.2 Como ESCREVER um Vetor

Para Escrever um vetor, ou seja, para escrever o conteúdo de cada posição de um vetor, também precisamos utilizar uma estrutura de repetição, já que os vetores possuem mais de um conteúdo (mais de uma posição). Como explicado no capítulo anterior (16.1 Como Ler um Vetor), normalmente utiliza-se a estrutura **Para-Até-Faça** também para escrever o vetor. Veja no exemplo abaixo, como escrever um vetor de 10 posições, isto é, como escrever o conteúdo de cada uma das 10 posições do vetor:

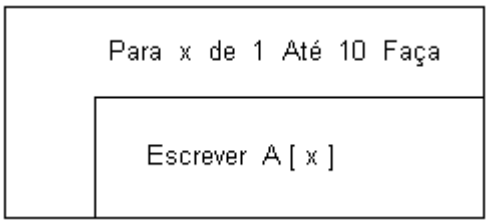


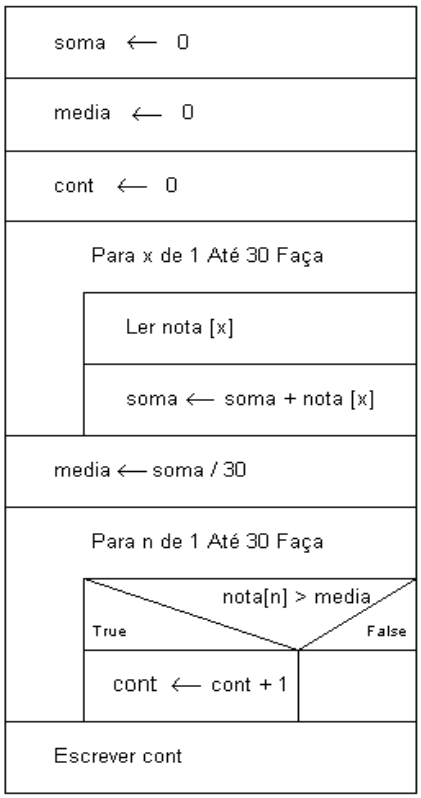
Figura 12: Exemplo para Escrever um Vetor

Na Figura 12, acima, está demonstrado como escrever um vetor chamado A de 10 posições, ou seja, ao executar essa instrução seria escrito o conteúdo de cada uma das 10 posições do vetor A na tela do computador.

Exemplo 13:

Escreva um algoritmo para ler a nota de 30 alunos, calcular a média geral da turma e escrever quantos alunos tiveram a nota acima da média calculada.

Resposta em Chapin



Resposta em Português Estruturado

```
Início
    soma ← 0
    media ← 0
    cont ← 0
    Para x de 1 Até 30 Faça
        Ler nota [x]
        soma ← soma + nota [x]
    FimPara
    Media ← soma / 30
    Para n de 1 Até 30 Faça
        Se nota [n] > media Então
            cont ← cont + 1
        FimSe
    FimPara
    Escrever cont
Fim
```

17 Respostas dos Exemplos

Neste capítulo são apresentadas as respostas de todos os exemplos encontrados no decorrer da apostila.

Exemplo 1:

```
X ← 20
Y ← 5
Z ← X + Y
Escrever X
Escrever Y
Escrever Z
```
Escrever X, Y, Z

Exemplo 2:

```
A ← 4
B ← 3
C ← A + B
D ← A - B
Escrever A, B, C, D
Escrever 'Fim do Algoritmo'
```

Exemplo 3:

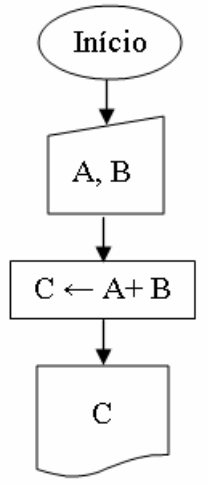
Resposta em Chapin:

```
Ler A
Ler B
C ← A + B
Escrever C
```
Ler A, B

Resposta em Português Estruturado:

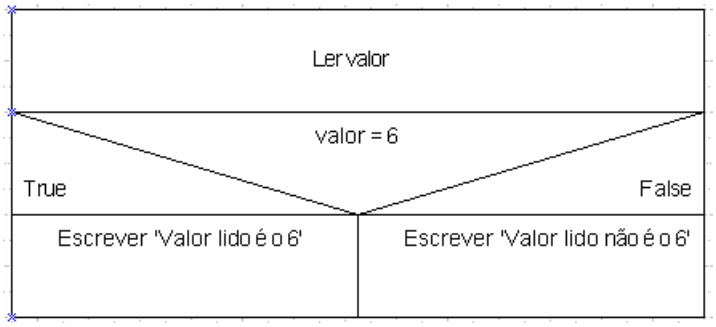
```
Início
   Ler A, B
   C ← A + B
   Escrever C
Fim
```

Resposta em Fluxograma:



Exemplo 4:

Resposta em Chapin:

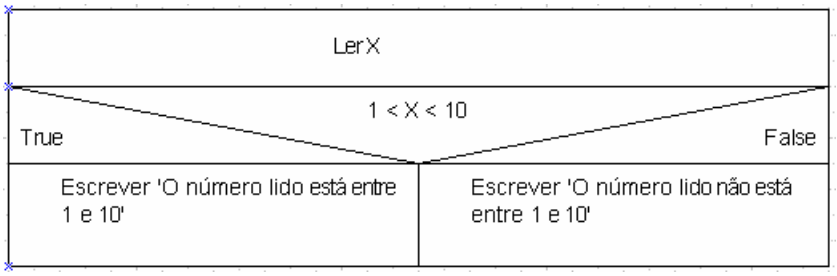


Resposta em Português Estruturado:

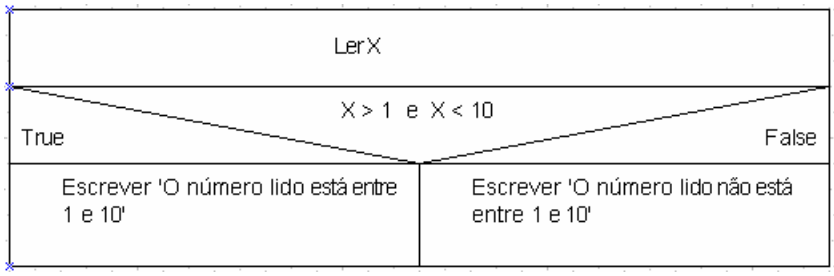
```
Início
    Ler  valor
    Se  valor = 6  Então
        Escrever 'Valor lido é o 6'
    Senão
        Escrever 'Valor lido não é o 6'
    FimSe
Fim
```

Exemplo 5:

Resposta **Errada**:

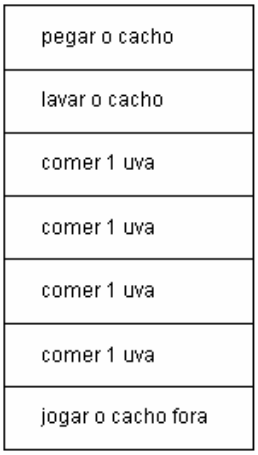


Resposta **Correta**:



Exemplo 6:

Resposta em Chapin:



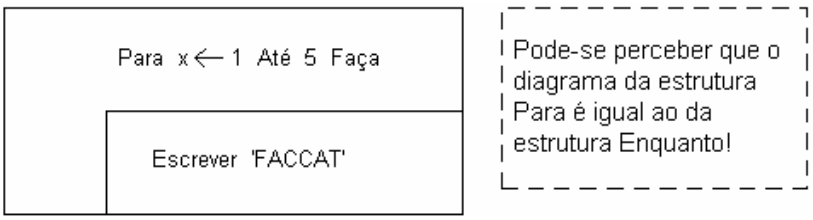
Resposta em Português Estruturado:

```
Início
    pegar o cacho
    lavar o cacho
    comer 1 uva
    comer 1 uva
    comer 1 uva
    comer 1 uva
    jogar o cacho fora
Fim
```

Explicação: **Esta resposta funciona perfeitamente se o cacho tiver 4 uvas, pois se tiver mais de 4, algumas irão fora e se tiver menos de 4, o algoritmo iria "trancar" durante a execução.**

Exemplo 7:

Resposta em Chapin:

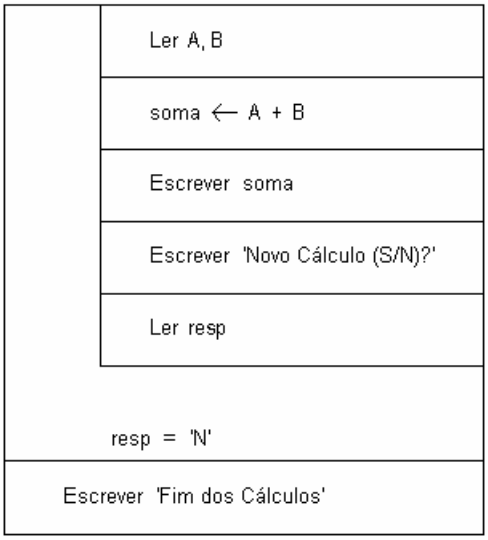


Resposta em Português Estruturado:

```
Início
   Para x de 1 Até 5 Faça
      Escrever 'FACCAT'
   FimPara
Fim
```

Exemplo 8:

Resposta em Chapin

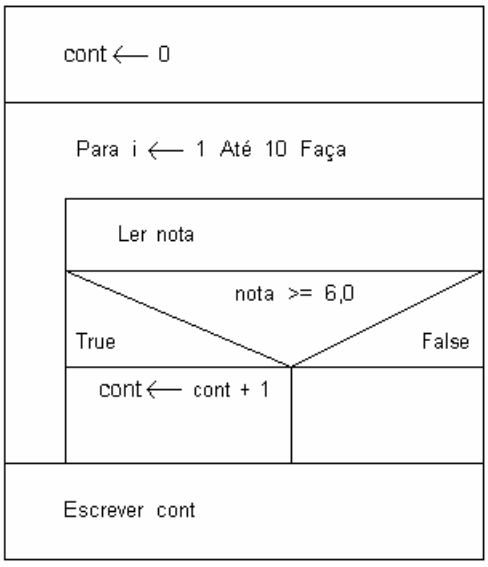


Resposta em Português Estruturado

```
Início
   Repita
      Ler A, B
      soma ⟵ A + B
      Escrever soma
      Escrever 'Novo Cálculo (S/N)?'
      Ler resp
   Até resp = 'N'
   Escrever 'Fim dos Cálculos'
Fim
```

Exemplo 9:

Resposta em Chapin

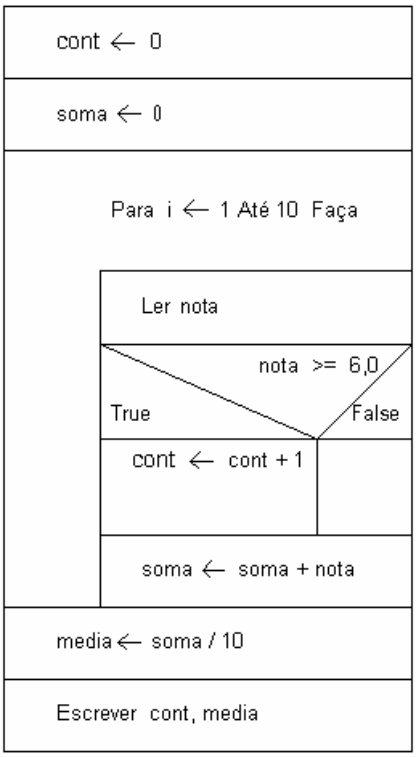


Resposta em Português Estruturado

```
Início
   cont ⟵ 0
   Para i de 1 Até 10 Faça
      Ler nota
      Se nota >= 6,0 então
         cont ⟵ cont + 1
      FimSe
   FimPara
   Escrever cont
Fim
```

Exemplo 10:

Resposta em Chapin

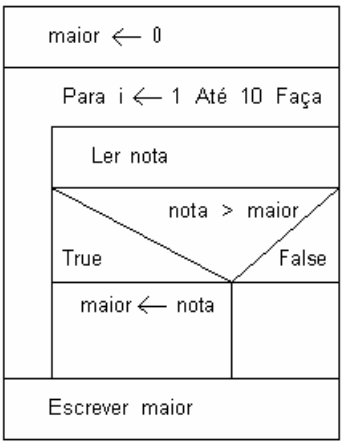


Resposta em Português Estruturado

```
Início
   cont ⟵ 0
   soma ⟵ 0
   Para i de 1 Até 10 Faça
      Ler nota
      Se nota >= 6,0 Então
         cont ⟵ cont + 1
      FimSe
      soma ⟵ soma + nota
   FimPara
   media ⟵ soma / 10
   Escrever cont, media
Fim
```

Exemplo 11:

Resposta em Chapin



Resposta em Português Estruturado

```
Início
   maior ⟵ 0
   Para i de 1 Até 10 Faça
      Ler nota
      Se nota > maior Então
         maior ⟵ nota
      FimSe
   FimPara
   Escrever maior
Fim
```

Referências Bibliográficas

Para elaboração e construção desta apostila foram consultados vários tipos de materiais, como por exemplo: livros, outras apostilas, páginas web etc. Algumas das referências consultadas estão apresentadas neste capítulo, mas grande parte do material disponibilizado na apostila, como exemplos e exercícios foram utilizados das aulas que tive na disciplina de Algoritmos e Programação durante a faculdade de Análise de Sistemas, na UCPel - Universidade Católica de Pelotas, com o **Prof. Ricardo Andrade Cava (***http://lp.ucpel.tche.br*, *http://cpu.ucpel.tche.br*, *http://graphs.ucpel.tche.br*).

[ALG96] **The ALGOL Programming Language**. Disponível em: http://www.engin.umd.umich.edu/CIS/course.des/cis400/algol/algol.html. Acesso em: Jun. 2006.

[BUF03] BUFFONI, Salete. **Apostila de Algoritmo Estruturado - 4ª edição**. Disponível em: http://www.saletebuffoni.hpg.ig.com.br/algoritmos/Algoritmos.pdf. Acesso em: Mar. 2004.

[CHA70] CHAPIN, Ned. **Flowcharting with the ANSI Standard: A Tutorial**. ACM Computing Surveys, Volume 2, Number 2 (June 1970), pp. 119-146.

[CHA74] CHAPIN, Ned. **New Format for Flowcharts, Software—Practice and Experience**. Volume 4, Number 4 (October-December 1974), pp. 341-357.

[CHA02] CHAPIN, Ned. **Maintenance of Information Systems**. Disponível em: http://www.iceis.org/iceis2002/tutorials.htm. Acesso em: Jun. 2006.

[COS04] COSTA, Renato. **Apostila de Lógica de Programação - Criação de Algoritmos e Programas**. Disponível em: http://www.meusite.pro.br/apostilas2.htm. Acesso em: Jun. 2006.

[GOM04] GOMES, Abel. **Algoritmos, Fluxogramas e Pseudo-código - Design de Algoritmos.** Disponível em: http://mail.di.ubi.pt/~programacao/capitulo6.pdf. Acesso em: Jun. 2006.

[KOZ06] KOZAK, Dalton V. **Técnicas de Construção de Algoritmos**. Disponível em: http://minerva.ufpel.edu.br/~rossato/ipd/apostila_algoritmos.pdf. Acesso em: Jun. 2006.

[MAR03] MARTINS, Luiz E. G.; ZÍLIO, Valéria M. D. **Apostila da Disciplina Introdução à Programação.** Disponível em: http://www.unimep.br/~vmdzilio/apostila00.doc. Acesso em: Jun. 2006.

[NAS73] NASSI, Ike; SHNEIDERMAN, Ben. **Flowchart Techniques for Structured Programming.** ACM SIGPLAN Notices, Volume 8, Number 8 (August 1973), pp.12-26.

[NAS04] NASSI, Ike. Ike Nassi's Home Page. Disponível em: http://www.nassi.com/ike.htm . Acesso em: Jun. 2006.

[PRO04] **Programming Languages**. Disponível em: http://www.famed.ufrgs.br/disciplinas/inf_med/prog_ling.htm. Acesso em: Mar.2004.

[SAN04] SANTANA, João. **Algoritmos & Programação**. Disponível em: http://www.iesam.com.br/paginas/cursos/ec/1ano/aulas/08/joao/APunidade-1.pdf. Acesso em: Mar. 2004.

[SHN03] SHNEIDERMAN, Ben. **A short history of structured flowcharts (Nassi-Shneiderman Diagrams).** Disponível em: http://www.cs.umd.edu/~ben/ Acesso em: Jun. 2006.

[TON04] TONET, Bruno; KOLIVER, Cristian. **Introdução aos Algoritmos.** Disponível em: http://dein.ucs.br/napro/Algoritmo/manuais/Manual 20Visualg.pdf. Acesso em: Mar. 2004.

[YOU04] YOURDON, Ed. **Ed Yourdon's Web Site**. Disponível em: http://www.yourdon.com/books/msa2e/CH15/CH15.html. Acesso em: Mar. 2004.