**Introdução a lógica**

O que é lógica (logos) discurso, razão ou regra. O estudo do raciocínio válido.

**Raciocínio lógico**

Tomada de decisão para execução de uma sequência de ações para execução de uma tarefa.

Instruções são os passos de uma sequência lógica para conseguir atingir o objeto em uma tarefa. Estas instruções indicam aos computadores o que deve ser executado e sua sequência.

Quando utilizamos uma linguagem de programação para escrever uma sequência lógica estamos criando um programa de computador.

Os programadores são profissionais responsáveis por criar, testar e fazer manutenção dos programas.

Dúvida, qual linguagem estudar?

Normalmente os programas funcionam de uma mesma forma com entrada(input) - processo computacional - saída(output)

**linguagem de baixo nível:**

sintaxe complexa

mais próximas da linguagem de máquina

maior tempo de desenvolvimento

exemplo: Assembly

**linguagem de alto nível:**

sintaxe simples

mais próxima da linguagem humana

menor tempo de desenvolvimento

exemplos: java, c-sharp, php, html

**Tradutores**

Todo código precisa passar pelo processo de tradução antes de ser enviado ao computador, A tradução tem como objetivo transformar de linguagem de programação em um código binário de maquina.

Existem três tipos básicos de tradutores: Montador, Interpretador e Compilador.

**Montador**

Traduz o código assembly diretamente para linguagem de máquina

**Interpretador**

A tradução em código de máquina e execução são executadas no mesmo momento, isso pode afetar o desempenho de execução, mas possibilita uma maior flexibilidade quando estamos programando. Realiza a tradução e a execução simultaneamente, não gerando o código-objeto (linguagem de máquina) em disco.

**Compilador**

Após a validação do código pelo verificador ele é compilado gera um código chamado módulo-objeto que passa por um segundo processo de tradução que então gera o código de máquina para ser gerado pelo computador.

**Tipos de programação (Paradigmas)**

É a maneira como estruturamos nossos pensamentos ao codificar e criar programas de computadores.

**Programação orientada a objetos (POO)**

A programação orientada a objetos se baseia na interação entre as unidades do software,

chamadas de objetos. Nesse estilo de programação, são definidas diversas classes que

determinam o comportamento e os estados de cada objeto.

**Programação Procedural ou Estruturada**

Na programação tradicional (programação procedural), toda a lógica é descrita em sequência

e executada nesta ordem. As ações do usuário são “engessadas”, já que a ordem não pode

ser alterada por ele.

**Programação Orientada a Eventos**

Na programação orientada a eventos, nossa lógica é “quebrada” em vários módulos

(procedimentos), que são executados na ordem determinada pelas ações do usuário.

**Tipos de eventos**

Um evento consiste em um fato que possa influenciar na execução de um programa. Sem a

ocorrência de um evento, o programa que depende dele para ser executado ficaria em situação

de espera eternamente.

Existem três tipos básicos de eventos:

**Eventos de mouse**

Ocorre quando executamos qualquer tipo de ação com o mouse, como mover o mouse sobre

um objeto, pressionar um dos botões do mouse sobre um objeto, soltar um dos botões do

mouse, clicar (pressionar e soltar) com o botão esquerdo do mouse sobre um objeto ou aplicar

um duplo-clique com o botão esquerdo do mouse sobre um objeto.

**Eventos de Teclado**

Ocorre quando executamos qualquer tipo de ação com o teclado, como quando pressionamos uma tecla quando o foco está sobre um objeto, quando soltamos uma tecla quando o foco está sobre um objeto ou quando pressionamos e soltamos uma tecla quando o foco está sobre um objeto.

**Eventos de clock**

É gerado a pedido do programador, em intervalos de tempo regulares, cuja unidade de medida

é milissegundos.

**Algoritmo**

Descrições sequências de passos a serem executados por um programa. São como receitas a sequência de passos para fazer algo.

Algoritmo é composto por 3 elementos, ação e decisão ou laço(loop).

**Ação**

Qualquer coisa que iremos realizar para alcançar um objetivo proposto pelo algoritmo.

**Decisão**

Estrutura para verificação de condição para tomada de decisão no algoritmo, no mínimo possuirá dois caminhos, o lado verdadeiro [True] ou falso [False] da indagação.

**Decisões encadeada**

Colocamos um ”SE” dentro do outro.

**Comando Caso**

Representa diversas alternativas em uma decisão. Não a alternativa verdadeira ou falsa, executa uma ação de acordo com uma sentença.

**Laço (Loop)**

Trecho do código que executa uma ação diversas vezes até que seja alcançada determinada situação ou estado.

**Enquanto**

Utilizado para repetir trechos do algoritmo enquanto uma ou mais condições forem satisfeitas.

Escreva um algoritmo para fazer uma ligação telefônica a partir de um telefone fixo que já esteja devidamente ligado e conectado:

1. Início

2. Retirar o fone do gancho

3. Colocar fone na orelha

4. Verificar se tem linha

se sim: próximo passo do código 5

se não: Colocar o fone no gancho

Ir para o passo 2

5. digitar número do telefone

6. verificar se está chamando

se sim: próximo passo do código 7

se não: ir para o passo 2

7. Ligação atendida?

se sim: Conversar

ir próximo passo do código 8

se não: tentar mais tarde

ir próximo passo do código 8

8. Colocar fone no gancho.

9. Fim

**Variáveis, Operadores e Funções**

**Variáveis**

Variáveis são áreas na memória, utilizadas em programação, que servem para armazenar

dados. O conteúdo de uma variável pode ser alterado, mas uma variável só pode conter um

dado por vez.

As áreas na memória são divisões na memória. O computador identifica cada divisão por meio

de um endereço no formato hexadecimal, ou seja, para facilitar a localização dos dados, as

variáveis são encontradas pelos endereços de memória, assim como uma casa é encontrada

pelo seu endereço. Utilizamos variáveis frequentemente em programação.

**Tipos de variáveis**

Alfanumérica: Atribuímos qualquer tipo de caractere a elas, ou seja, letras, números ou

sinais;

Numérica: Atribuímos somente números a elas, pois podemos usá-las para efetuar cálculos:

Data: Atribuímos somente datas a elas;

Lógica: Atribuímos somente valores verdadeiros ou falsos (V/F). São utilizadas para testes

lógicos;

Objeto: Atribuímos uma referência a um objeto.

**Nomear variáveis**

Para atribuir um nome a uma variável, devemos observar alguns aspectos:

Não é possível começar um nome de variável com número;

Só é permitido o uso de underline (\_) no nome. Espaço ou qualquer outro sinal é proibido;

Para maior facilidade, devemos usar sempre nomes autoexplicativos;

Devemos, no entanto, cuidar para que as variáveis não tenham nomes de comandos,

funções ou campos de um banco de dados, pois isto pode causar problemas durante a

execução do programa.

**Declaração de variáveis**

As variáveis precisam ser declaradas. Para declarar uma variável, devemos informar o nome

e o tipo da variável.

Cada um dos tipos de variáveis ocupam espaços diferentes de memória, de acordo com os

dados que irão armazenar. A definição do tipo de variável otimiza a utilização do espaço de

memória.

Normalmente a variável é declarada no início do programa para que possa ser utilizada no

programa inteiro, sendo assim, a declaramos no início do algoritmo.

Veja, a seguir, exemplos de como iremos declarar as variáveis nos algoritmos:

**Declara A, B, C numéricas, D, E alfanuméricas**

**Declara VALORX numérica, TEXTOY alfanumérica**

**Declara NOTA\_BIMESTRAL1, NOTA\_BIMESTRAL2 numéricas**

**Declara MENSAGEM5, MENSAGEM9 alfanuméricas, HOJE data**

**Comando de atribuição**

O comando de atribuição serve para armazenar um valor numa variável. As variáveis numéricas

que serão utilizadas no programa e não têm um valor inicial predefinido geralmente recebem

o valor zero (0).

Utilizaremos o sinal de igual (=) para representar o comando de atribuição.

Onde tiver =, lê-se recebe:

**A = 10**

**X = 0**

**Valor = 0**

**Nota = 7,5**

**Nome\_Aluno = “Leandro”**

**MensagemFinal = “Boa noite.”**

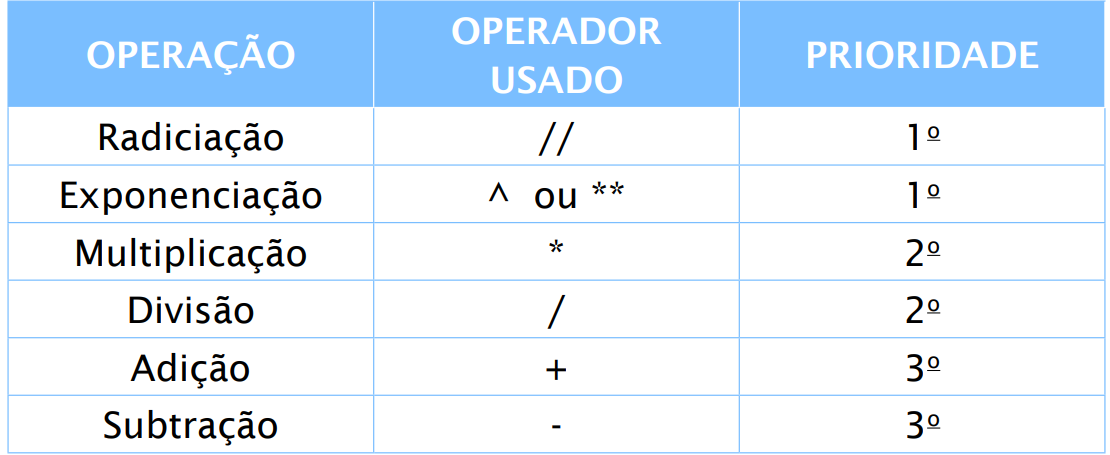
**Constantes**

Ao contrário das variáveis, as constantes possuem valor fixo e não sofrem alteração durante

o processamento.

**Operadores aritméticos**

Em programação, usamos os mesmos operadores aritméticos que são usados na matemática, e com as mesmas prioridades. Em primeiro lugar são executadas as exponenciações e as radiciações, em segundo as multiplicações e as divisões, e depois as adições e as subtrações.

****

Usamos também os seguintes operadores para operações matemáticas não-convencionais:

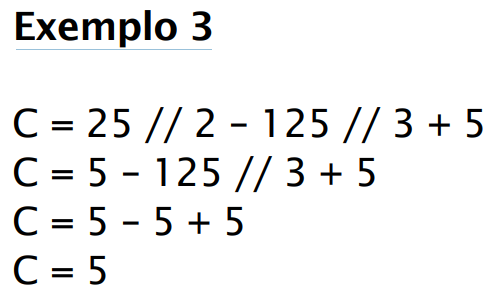
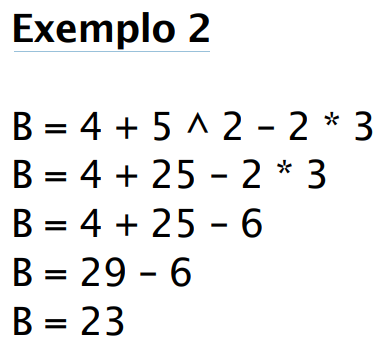
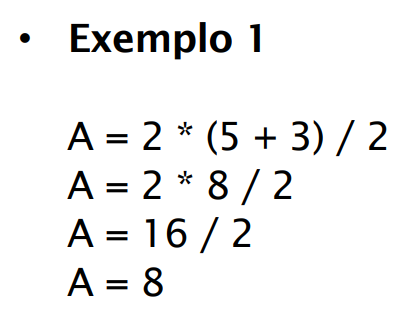
****

Normalmente, as operações são executadas da esquerda para a direita, a não ser que mudemos

sua ordem usando parênteses.

Veja, a seguir, o valor de cada variável com a resolução das fórmulas matemáticas respeitando

a prioridade de execução dos operadores.

****

**Contadores e acumuladores**

**Contador**

Um contador é construído a partir de uma variável que recebe o valor dela mesma mais outro

valor.

A linha que descreve um contador A é:

A = A + 1

Nesse exemplo a variável A está recebendo o valor dela mesma mais

1, ou seja, está contando de 1 em 1. O valor 1 é uma constante.

**Acumulador**

Um acumulador é uma variável que recebe o valor dela mesma mais o valor de outra variável.

A linha que descreve um acumulador B de A é:

B = B + A

Nesse exemplo, a variável B está recebendo o valor dela mesma mais o valor da variável A. A

variável A representa o valor a ser somado, acumulado na variável B.

**Acumulador de Conta**

A linha que descreve um acumulador C do dobro de B é:

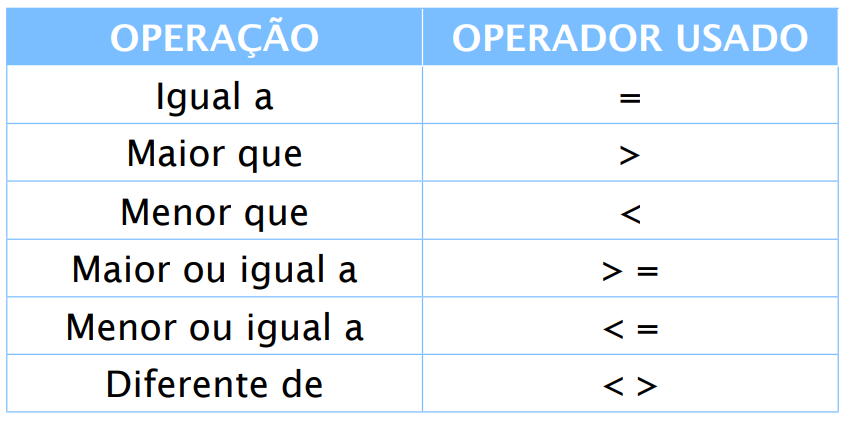
C = C + 2 \* B

Nesse exemplo, a variável C está recebendo o valor dela mesma mais duas vezes o valor da variável B.

**Operadores Relacionais**

Quando encontramos situações nas quais é necessário usar condições, adotamos os seguintes

operadores para estabelecer relações:



**Exemplos**

Num algoritmo:

Se o Cargo for igual a Gerente

Então conceder um aumento de 10% no salário

Senão conceder um aumento de 5% no salário

Fim Se

Numa linguagem de programação:

IF Cargo = “Gerente”

Salario = Salario \* 1.10

ELSE

Salario = Salario \* 1.05 ENDIF

**Comandos If Else**

**Operadores Lógicos**

São usados quando existem duas ou mais condições em um teste lógico, e assim como os operadores de comparação, retornam um resultado VERDADEIRO ou FALSO:

• NÃO (NOT) – Ordem de Prioridade: 1ª

Operador lógico que inverte a lógica de uma expressão. Se ela for verdadeira, torna-se falsa, e vice-versa.

• E (AND) – Ordem de Prioridade: 2ª

Operador lógico em que a resposta da operação é verdadeira somente se as duas condições forem verdadeiras.

Se o Cargo for igual a Gerente E a Idade for maior ou igual a 50 anos

Então conceder um aumento de 10% no salário

Senão conceder um aumento de 5% no salário

Fim Se

OU (OR) – Ordem de Prioridade: 3a

Operador lógico em que a resposta da operação é verdadeira se pelo menos uma das condições for verdadeira.

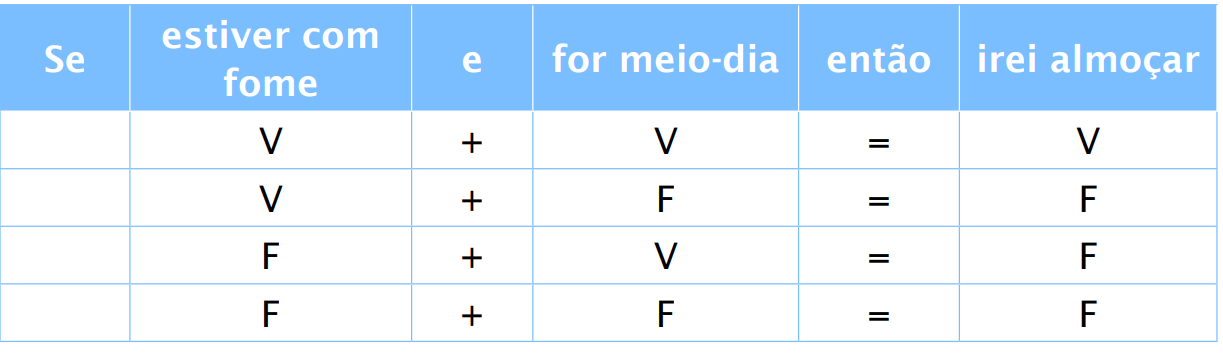
Se o cargo for igual a Gerente OU a Idade for maior ou igual a 50 anos

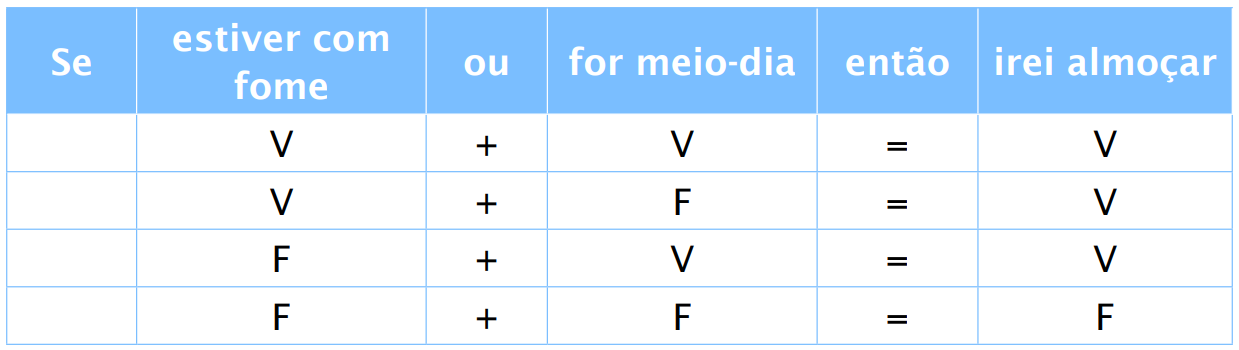
Então conceder um aumento de 10% no salário

Senão conceder um aumento de 5% no salário

Fim Se

**Tabela Verdade (Tabela de Decisão)**





**Funções**

Função é uma rotina que retorna um valor específico.

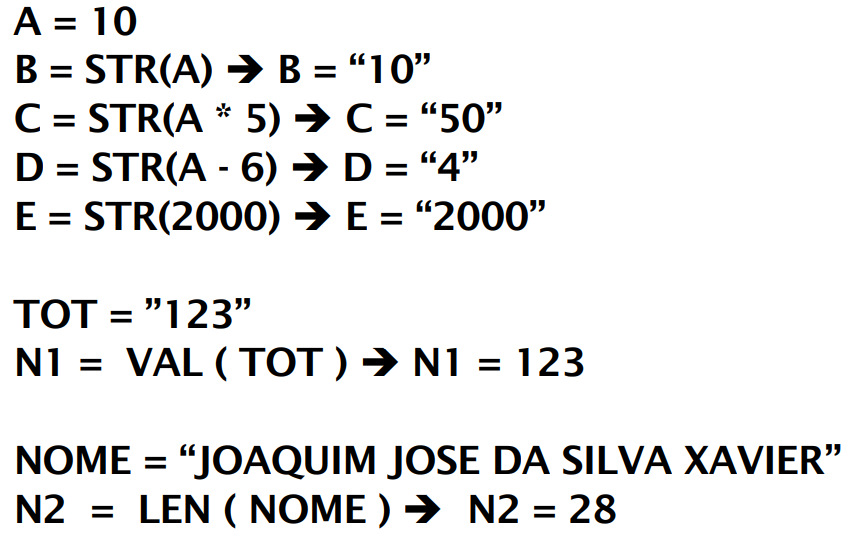
• STR( ): Transforma número em caracteres numéricos;

• VAL( ): Transforma caracteres numéricos em número;

• LEN( ): Retorna um inteiro contendo o número de caracteres de uma sequência de

caracteres, ou seja, conta os caracteres.

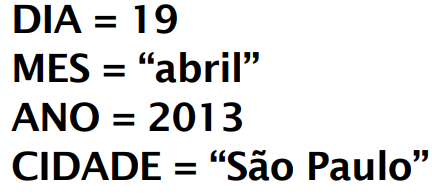
Exemplos:



**Concatenação de alfanuméricos**

Os símbolos para concatenação de alfanuméricos são: & ou + .

Exemplo:



A seguir, temos um exemplo de como fica a criação da variável DATA utilizando as variáveis anteriores para resultar no texto São Paulo, 19 de abril de 2013:

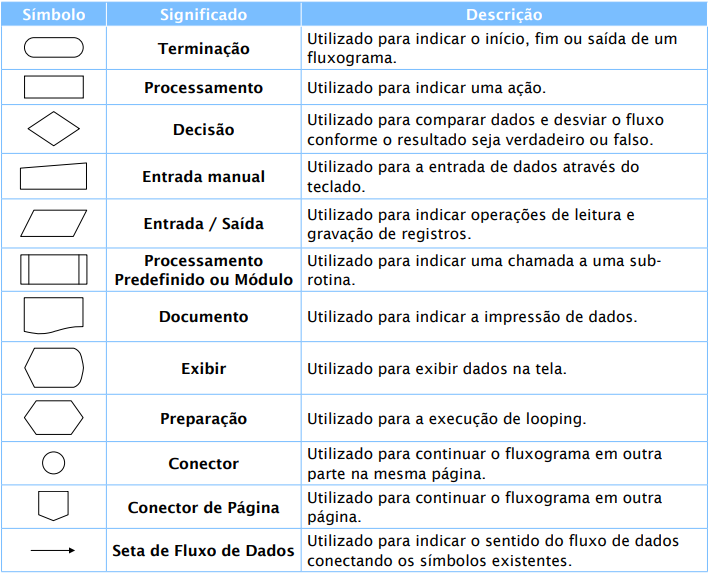


**Fluxograma**

**Fluxograma**, ou **Diagrama de Blocos**, é a representação gráfica de um algoritmo, sendo constituído de blocos funcionais que mostram o fluxo dos dados e as operações efetuadas com eles.

**Símbolos Geométricos**

O fluxograma utiliza símbolos que permitem a descrição da sequência dos passos a serem executados de forma clara e objetiva. A tabela a seguir descreve a simbologia utilizada em fluxogramas:



**Criando fluxogramas**

INÍCIO

Abrir a torneira

Molhar as mãos

Pegar o sabonete

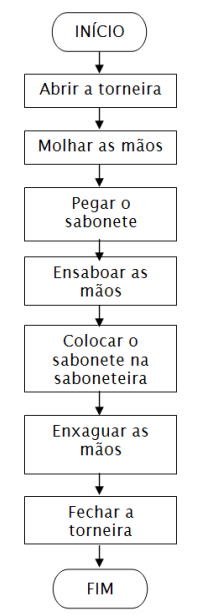
Ensaboar as mãos

Colocar o sabonete na saboneteira

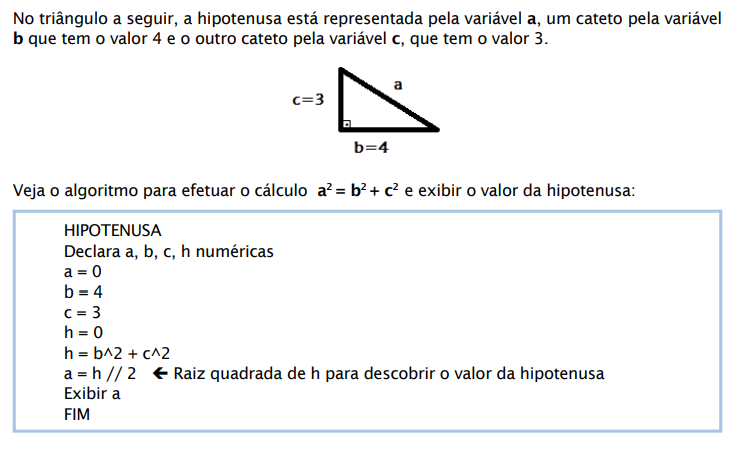
Enxaguar as mãos

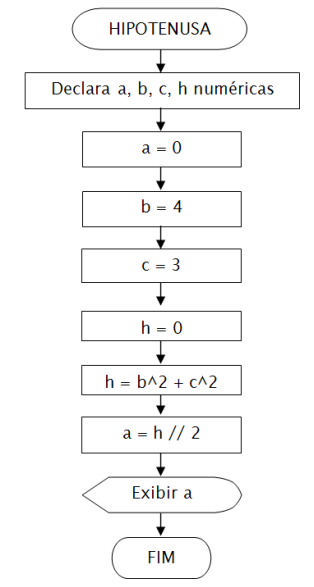
Fechar a torneira

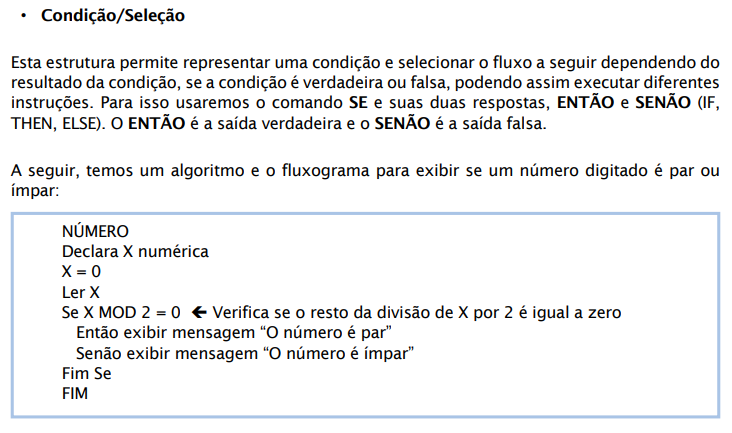
FIM

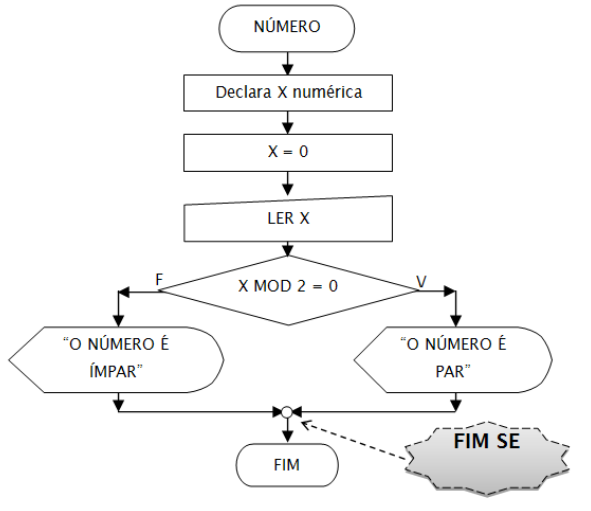


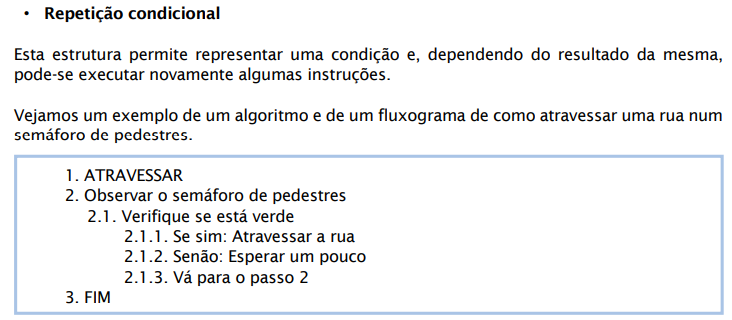
Outro exemplo, cálculo da hipotenusa:

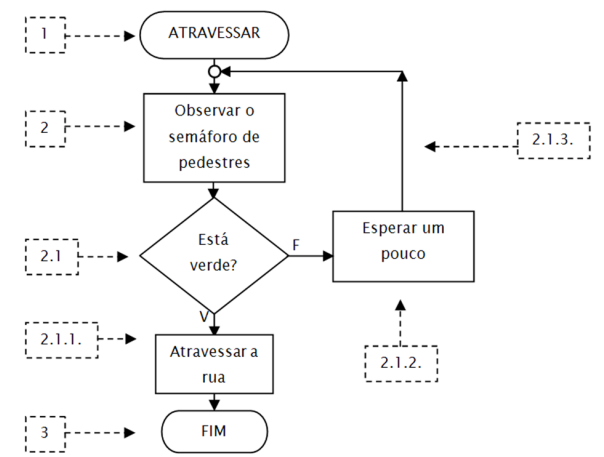


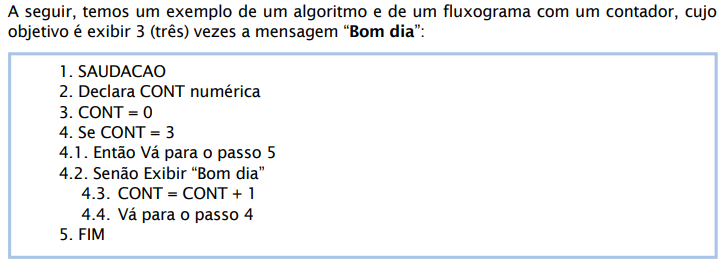


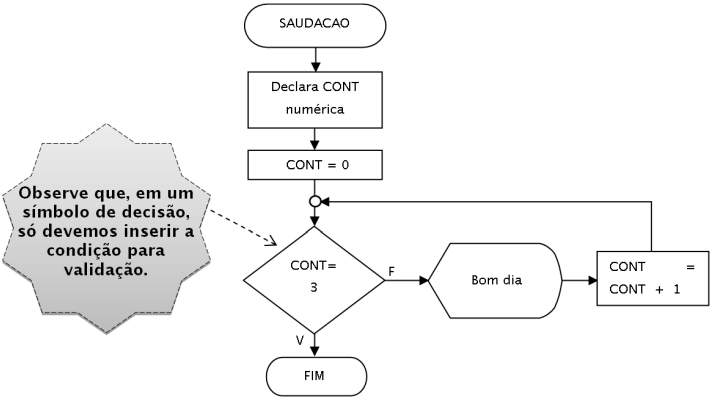
****

****

****

****

****

****

**Teste de Mesa**

Teste de Mesa é a simulação da execução de um algoritmo, programa ou fluxograma, sem utilizar o computador, empregando apenas lápis e papel.

A simulação da execução do fluxo para descobrir qual o valor da hipotenusa, conforme vimos anteriormente, representa o Teste de Mesa daquele fluxo.

Note no fluxo anterior que, inicialmente, o contador CONT recebeu o valor zero (0) e depois, na execução, recebeu os valores 1, 2 e 3.

CONT = 0

CONT = CONT + 1 -> CONT = 1

CONT = CONT + 1 -> CONT = 2

CONT = CONT + 1 -> CONT = 3

O resultado mostrado representa o Teste de Mesa exibindo todos os valores que a variável CONT recebeu.

**Laço ou Loop de Repetição**

Estruturas de laços e repetições são construídas para executar trechos de uma lógica várias vezes. Um laço em programação significa um retorno a linhas de programa já executadas para que sejam executadas novamente. Em inglês, laço significa loop, no sentido de retornar dando uma volta, fazendo um círculo.

Os laços de repetição se encarregam de repetir determinados comandos enquanto uma determinada condição for satisfeita.

Os comandos descritos a seguir são utilizados para efetuar um laço dentro de uma lógica de

programação.

**Comando FOR…NEXT (PARA...PRÓXIMO)**

Um loop com o comando FOR é realizado quando uma condição com operação matemática

é checada para executar um trecho de programa até o passo em que essa condição seja

satisfeita (verdadeira).

Na sintaxe do comando FOR, declaramos sua inicialização, sua condição e seu incremento.

A cada vez que o loop é executado, a variável do comando que é utilizada como contador

aumenta ou diminui de acordo com o incremento, até que a condição do comando não seja

mais satisfeita. Esse tipo de loop é uma repetição contável.



SAUDAÇÃO

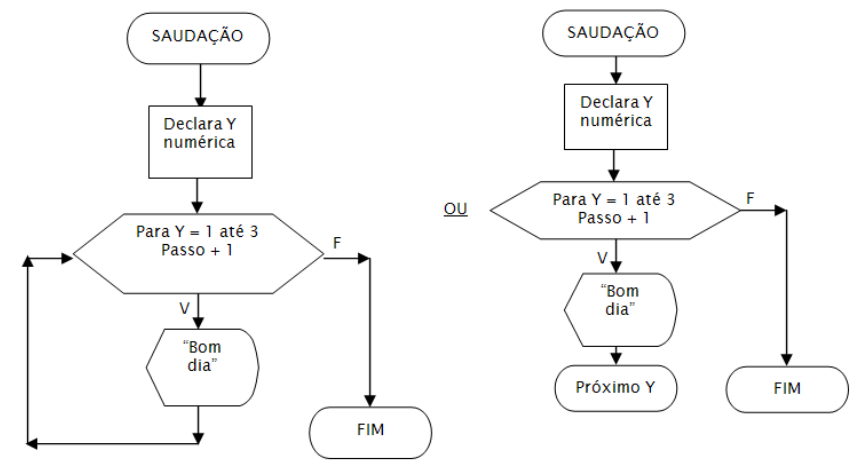
Declara Y numérica

Para Y = 1 até 3 passo + 1

Exibir “Bom dia”

Próximo Y

FIM

****

SOMAR 1 ATÉ 10

Declara SOMA, I numéricas

SOMA = 0

Para I = 1 até 10 passo + 1

SOMA = SOMA + I

Exibir I

Próximo I

Exibir SOMA

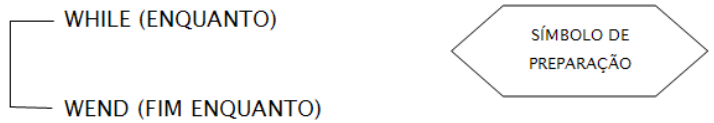
FIM

**Comando WHILE (ENQUANTO)**

Um loop com o comando WHILE é realizado quando uma condição é checada para executar um trecho de programa até o passo em que essa condição seja satisfeita (verdadeira). Trata-se de um loop de repetição contável.

Na sintaxe do comando WHILE, declaramos apenas sua condição. Sua inicialização e seu incremento são ações executadas separadamente.

O loop com o comando WHILE é utilizado quando não conhecemos o número de iterações que temos que realizar. Esta é a diferença do loop WHILE com o comando FOR.



SAUDAÇÃO

Declara Y numérica

Y = 1

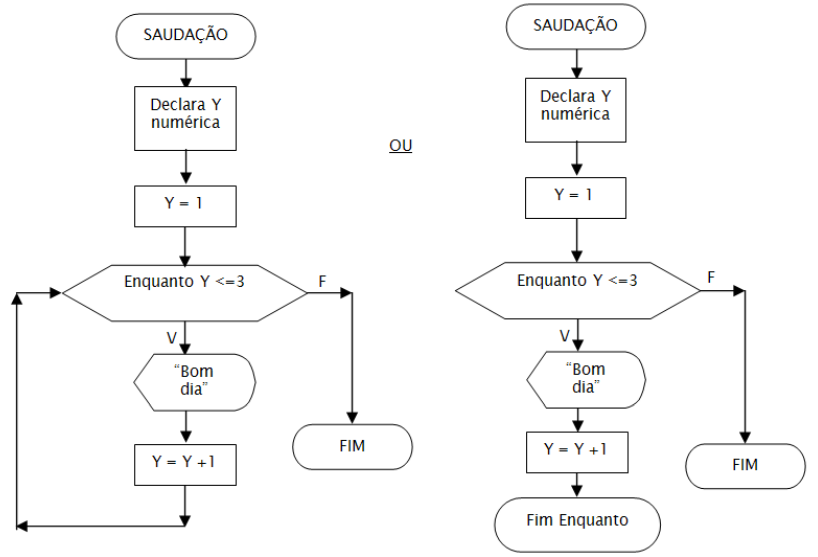
Enquanto Y <= 3

Exibir “Bom dia”

Y = Y + 1

Fim Enquanto

FIM



SOMAR 1 ATÉ 10

Declara SOMA, I numéricas

SOMA = 0

I = 1

Enquanto I <= 10

Exibir I

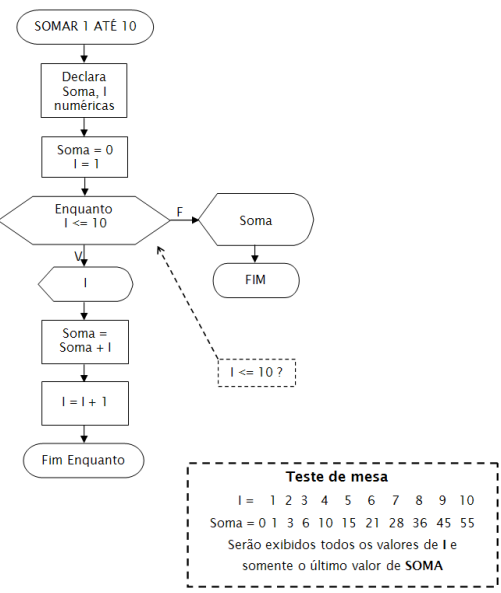
SOMA = SOMA + I

I = I + 1

Fim Enquanto

Exibir SOMA

FIM



**Comando DO...WHILE (FAÇA....ENQUANTO)**

Um loop com o comando DO...WHILE é utilizado quando é necessário executar um trecho de código ao menos uma vez, antes da condição do loop ser checada para continuar ou não a iteração. Isso quer dizer que, mesmo quando a condição estabelecida pelo loop é falsa, quando utilizamos DO...WHILE, o trecho de código será executado uma vez antes de ter sua iteração negada. Assim como o comando WHILE, o DO...WHILE é utilizado quando não conhecemos o número de iterações que iremos realizar. A seguir, temos um exemplo de algoritmo que utiliza o comando DO...WHILE para somar números até que o usuário digite o número zero (0):

SOMAR NÚMEROS

Declara SOMA, X numéricas

SOMA = 0

Faça:

Ler X

Se X != 0:

SOMA = SOMA + X

Enquanto X != 0

Exibir SOMA

FIM

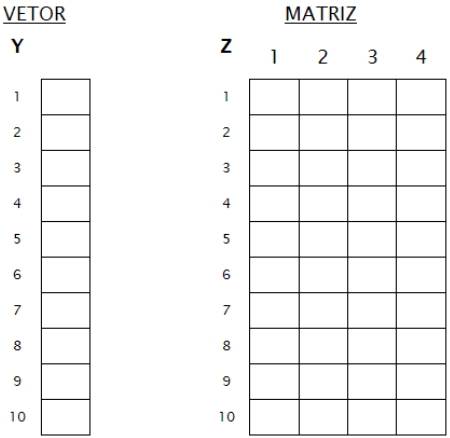
Veja que, como já havíamos atribuído o valor zero à variável SOMA antes de iniciar o loop, se utilizássemos a estrutura WHILE, o trecho de código não seria executado nenhuma vez. Com o uso do comando DO WHILE, é possível forçar a execução do trecho de código e obter resultados muito diferentes.

**Vetores e matrizes**

Neste capítulo aprenderemos a utilizar as variáveis indexadas em programação. Variáveis indexadas são um conjunto de variáveis que apresentam o mesmo nome, são do mesmo tipo, mas são diferentes no valor de seu índice. As variáveis indexadas podem ter várias dimensões:

• Vetores: uma dimensão;

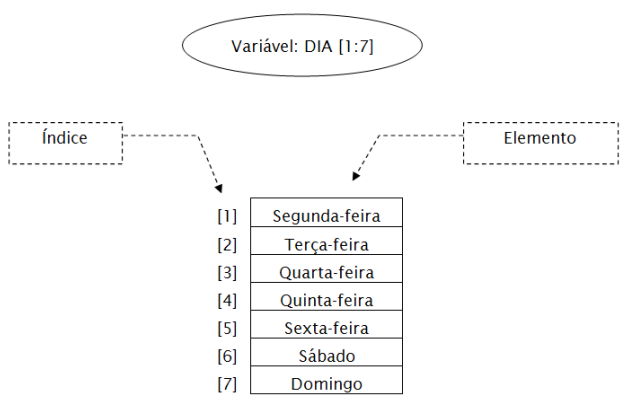
• Matrizes: n dimensões.



Dados dois números inteiros positivos m e n, chama-se matriz m x n a tabela formada por m.n

números reais, dispostos em m linhas (horizontais) e n colunas (verticais).

Um elemento de uma tabela pode ser referenciado de duas formas: Implícita e Explícita. Vejamos a seguinte tabela:



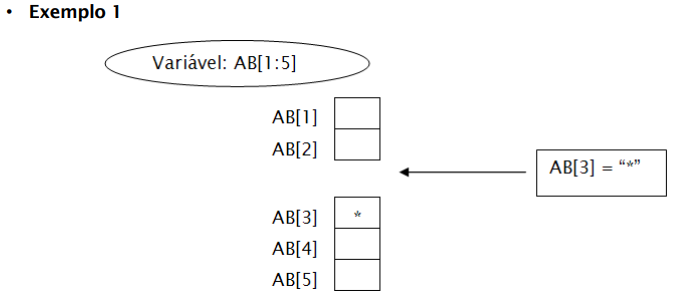
**• Referência implícita:** Usamos o índice para nos referenciar a um certo elemento da tabela. A seguir, temos um exemplo de referência implícita, considerando a tabela de dias da semana mostrada anteriormente:

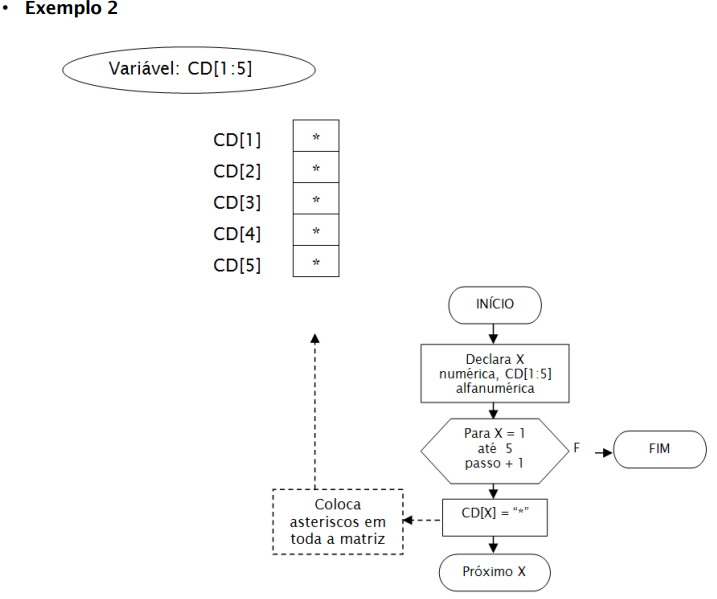
A = 2 -> DIA[ A ] = Terça-feira

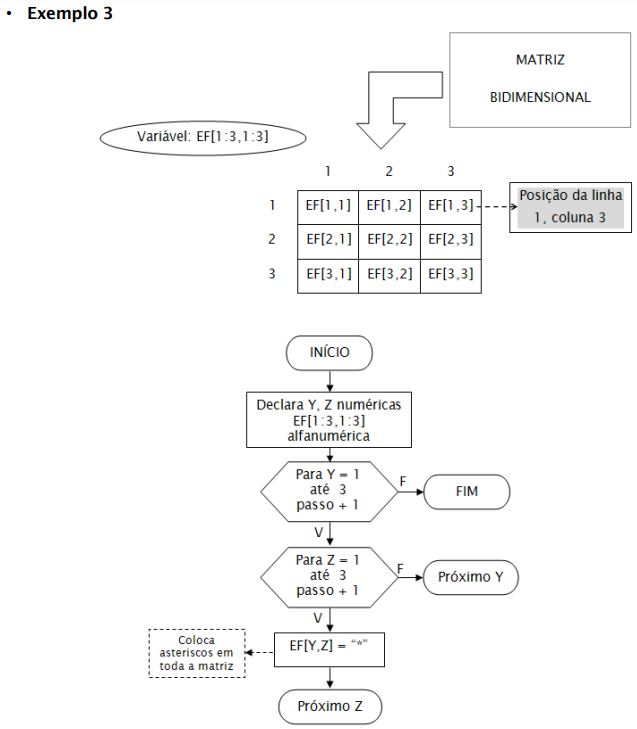
**• Referência explícita:** Referenciamos diretamente ao elemento desejado. A seguir, temos um exemplo de referência explícita, considerando nossa tabela de dias da semana:

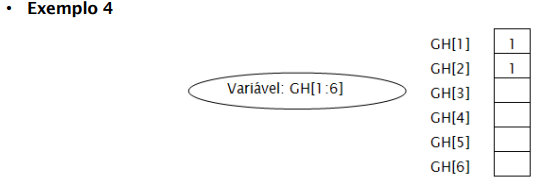
DIA[ 2 ] = Terça-feira

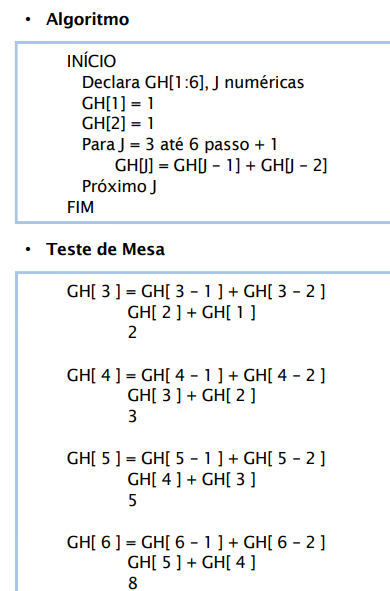
Outros exemplos:



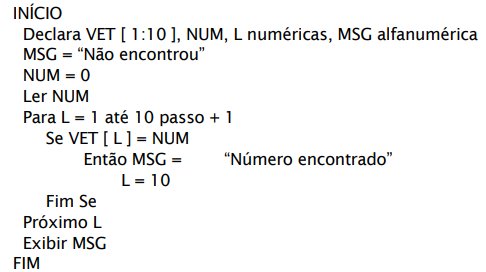


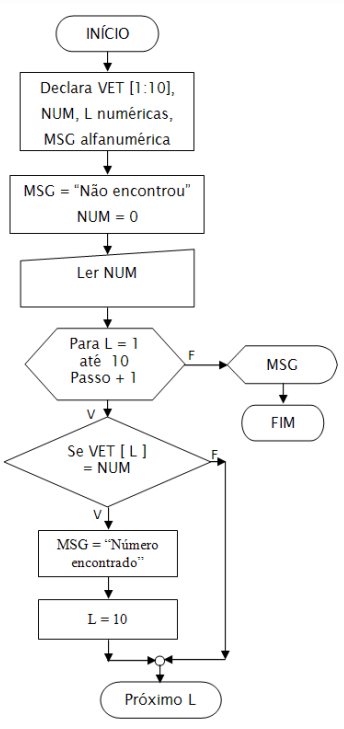






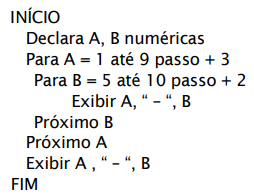
A seguir, temos um algoritmo e o fluxograma correspondente que verifica se um número digitado é encontrado num vetor. Será exibida uma mensagem informando se o número foi encontrado ou não. Como vetor, vamos considerar VET [ 1:10 ].

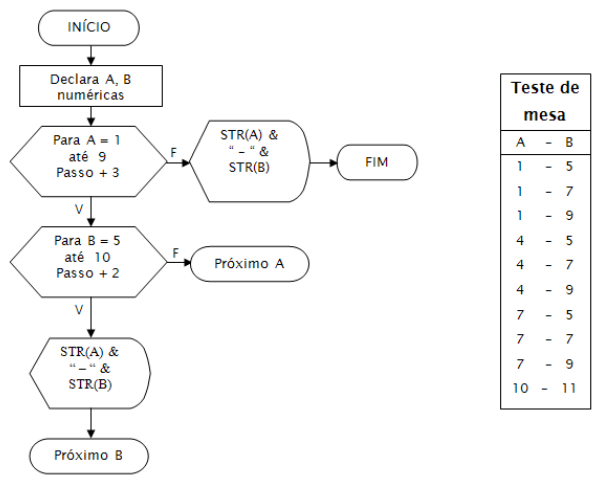




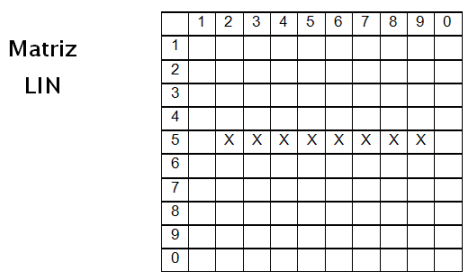
**Laços encadeados**

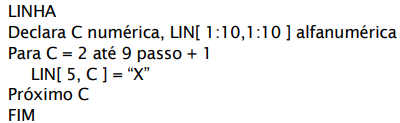
Laços ou Loops encadeados são laços executados dentro de outros laços. No caso do comando PARA, o primeiro a ser criado é o último a ser fechado. O comando PRÓXIMO fecha o laço. O último comando PARA aberto é o primeiro a ser fechado com o comando PRÓXIMO, ou seja, o último loop criado é o primeiro a ser fechado. Vejamos um exemplo de laço encadeado a seguir:

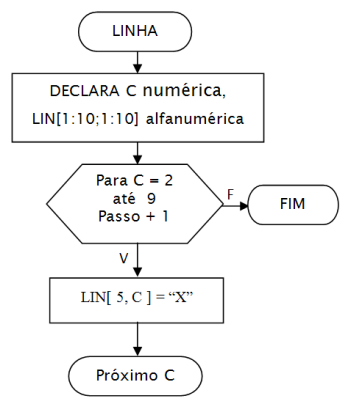




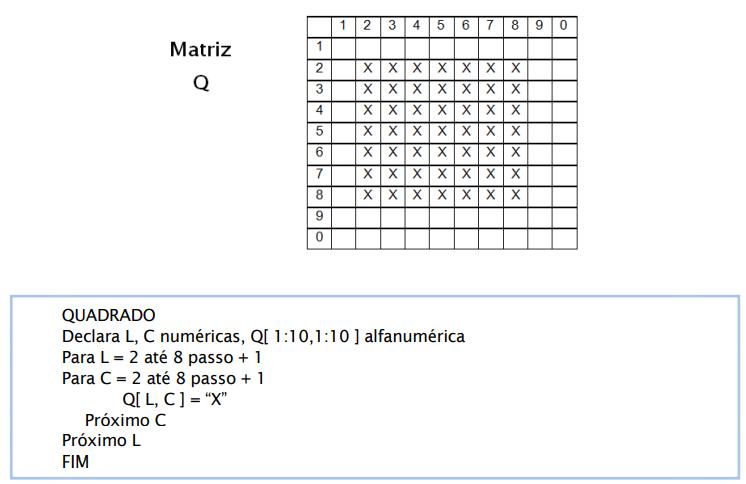
Neste outro exemplo, temos um algoritmo que preenche a matriz conforme a figura utilizando apenas um laço (loop):



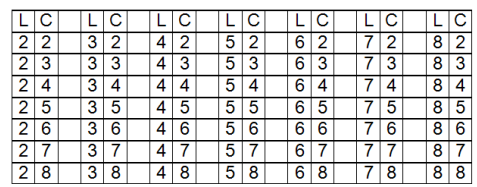




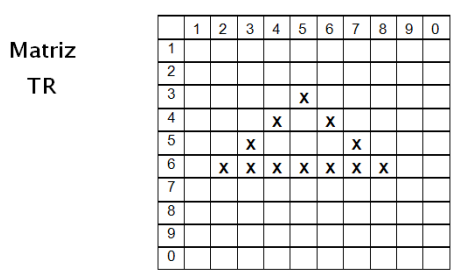
A seguir, temos um algoritmo que preenche a matriz conforme a figura utilizando laço (loop) encadeado:

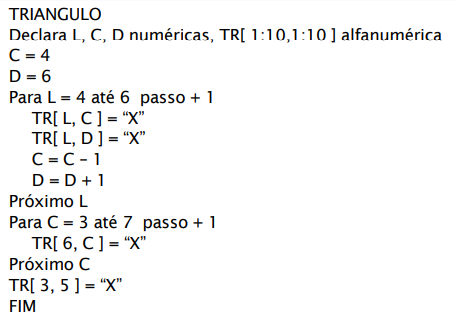


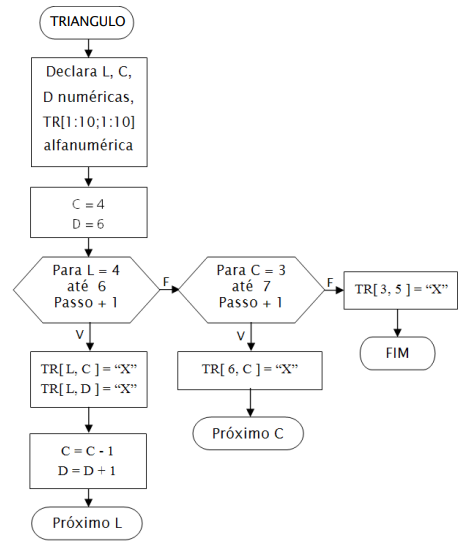
Vejamos o teste de mesa do último exemplo. Considerando que a variável L está sendo usada para a posição da linha e a variável C está sendo usada para a posição da coluna, veja, na tabela a seguir, que foi preenchida toda a matriz, desde a posição [2,2] até a posição [8,8].



A seguir, temos um algoritmo que preenche a matriz conforme a figura utilizando dois laços:



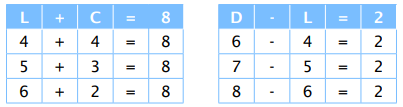


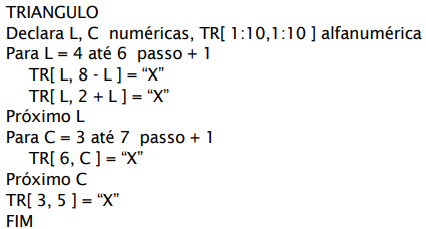


Considerando o exemplo do triângulo na matriz TR, observe, a seguir, as linhas e colunas preenchidas pelos valores das variáveis L, C e D:



Note que, se somarmos os valores das variáveis L e C, o resultado é sempre 8 (oito), portanto C = 8 – L. Após encontrarmos essa solução matemática, não será mais preciso utilizar a variável C no primeiro loop. Perceba, também, que, se subtrairmos dos valores da variável D os valores da variável L, o resultado é sempre 2 (dois), portanto D = 2 + L. Após encontrarmos essa solução matemática, não será mais preciso utilizar a variável D.





**Processamento Predefinido**

Processamento Predefinido é um programa que pode ser usado em outro programa. No contexto de linguagens de programação, um **subprograma**, **sub-rotina**, **função** ou **procedimento** consiste numa parte do programa que resolve um problema específico.

O conceito de **função** difere de **procedimento** porque ela retorna um valor, sendo que, em algumas linguagens, esta distinção não existe.

Uma **sub-rotina** pode ser usada em várias partes do programa ou sistema e assim podemos ter aproveitamento de uma determinada ação em diversos momentos diferentes, com a vantagem do gerenciamento da lógica dessa ação estar centralizada. Esta sub-rotina também pode ser reaproveitada em outros programas ou sistemas.

Os **parâmetros** são os argumentos do processamento predefinido. Eles são a comunicação da sub-rotina com os demais programas que a chamarão em algum momento e, através deles, a sub-rotina pode receber e retornar valores que serão processados para um objetivo final.

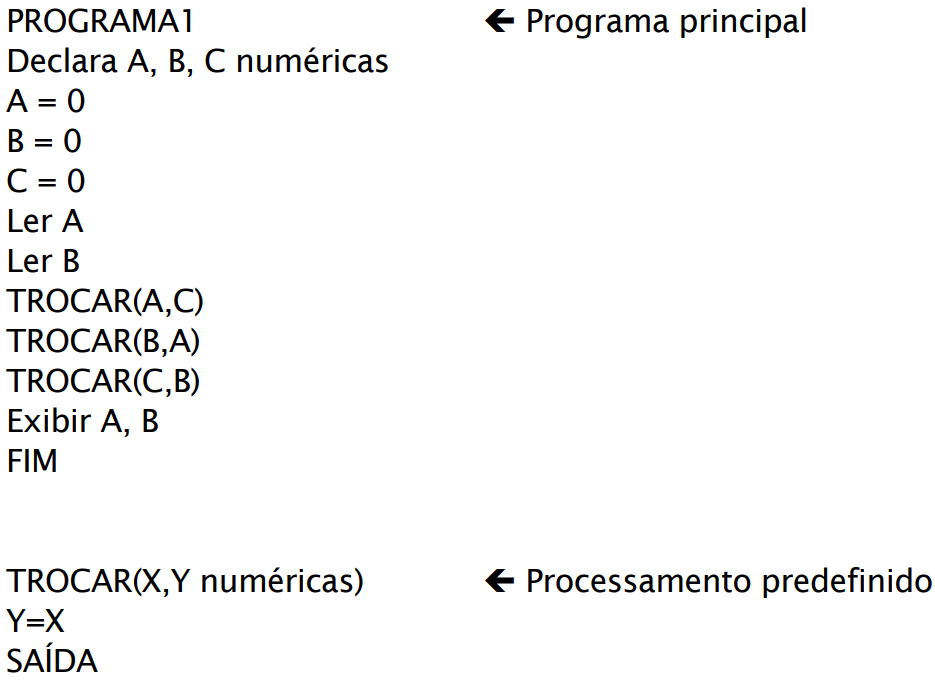
**Construindo um processamento predefinido**

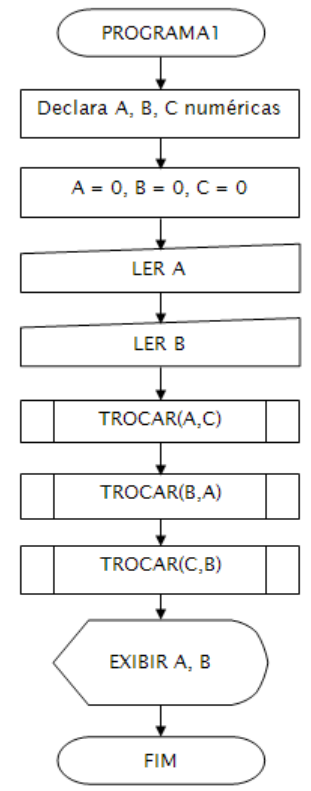
Para construir um processamento predefinido, considera-se um trecho de programa. Vejamos um exemplo de um algoritmo para:

• Receber 2 números nas variáveis A e B;

• Trocar os conteúdos das 2 variáveis, ou seja, o conteúdo da variável A deve ser transferido para a variável B, e o da variável B, para a variável A;

• Mostrar na tela os valores de A e B, para conferir se os conteúdos foram realmente trocados.





**Banco de Dados**

Um banco de dados é uma coleção de informações relacionadas a um determinado assunto ou finalidade, como um cadastro de fornecedores, um cadastro de produtos no estoque de uma empresa ou uma agenda.

As informações armazenadas em um banco de dados podem ser consultadas, comparadas, alteradas, impressas ou excluídas.

Ao criar um banco de dados, é fundamental que haja um planejamento voltado para o objetivo e forma de utilização desse banco, ou seja, é necessário considerar que tipos de informações ele deve conter.

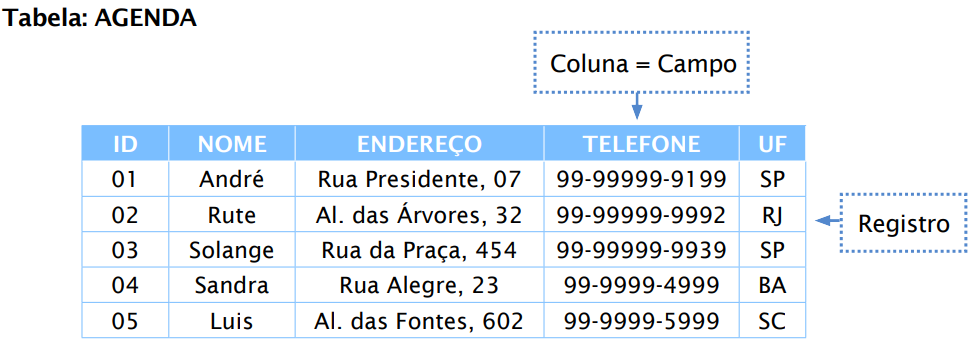
A estrutura de dados é uma área na qual definimos quais são os dados que queremos armazenar:

• **Campo**: É composto por um caractere ou um conjunto de caracteres. Os campos correspondem às colunas da tabela;

• **Registro**: Um campo ou um conjunto de campos. Os registros correspondem às linhas da tabela;

• **Tabela**: É composta por um registro ou um conjunto de registros, e um campo ou um conjunto de campos;

• **Banco de dados**: É composto por uma tabela ou um conjunto de tabelas.



Os bancos de dados que possuem tabelas relacionadas entre si são chamados de bancos de dados relacionais.

**Considerações para tipos de dados**

Para decidir a espécie de tipo de dados a ser utilizada para um campo, podemos tomar como base as seguintes considerações:

• A espécie de valores que desejamos armazenar no campo. Por exemplo, não é possível armazenar texto em um campo com um tipo de dados DATA;

• O espaço de armazenamento que desejamos utilizar para os valores neste campo;

• Os tipos de operações que desejamos efetuar com os valores do campo. Por exemplo, não é possível somar valores em campo com tipo de dados TEXTO, mas em campos do tipo NÚMERO é possível;

• A classificação dos valores de um campo. Os números são classificados como sequências de caracteres em um campo do tipo TEXTO (1, 10, 2, 20, 3, 30, e assim por diante) e não como valores numéricos. Para classificar números como valores numéricos utilizamos um campo do tipo NÚMERO.

**Tipos de dados**

• **Texto**: Texto ou combinações de textos e números, como endereços ou números que não exijam cálculos, como números de telefone ou códigos postais;

• **Número**: Dados numéricos a serem utilizados em cálculos matemáticos;

• **Moeda**: Valores monetários. Evita o arredondamento durante os cálculos;

• **Data/Hora**: Datas e horas;

• **Lógico** (**Booleano**): Campos que irão conter somente um entre dois valores, como Sim/ Não, Verdadeiro/Falso ou Ativado/Desativado;

• **Objeto**: Objetos criados em outros programas (como documentos do Microsoft Word, planilhas do Microsoft Excel, figuras, sons ou outros dados binários).

**Modelo de dados**

É o diagrama contendo as estruturas de dados e os relacionamentos.

**Relacionamento**

O relacionamento é um componente que define como duas tabelas se relacionam. As duas tabelas que se deseja ligar devem, obrigatoriamente, ter um campo em comum. Este campo recebe o nome de chave.

**Chave primária**

O campo **CHAVE PRIMÁRIA** determina de forma exclusiva cada registro armazenado. Não existem dois registros com o mesmo dado em um campo CHAVE PRIMÁRIA de uma mesma tabela.

**Chave estrangeira**

Chamamos de **CHAVE ESTRANGEIRA** o campo que possui um relacionamento com uma **CHAVE PRIMÁRIA** de outra tabela. Esse tipo de chave, que pode ocorrer repetidas vezes, estabelece um relacionamento entre a tabela em que ela está localizada e a tabela que contém a **CHAVE PRIMÁRIA.**

**Modelo Entidade-Relacionamento**

O diagrama de um sistema que contém todas as suas tabelas e seus relacionamentos recebe o nome de Modelo Entidade-Relacionamento. Esse diagrama deve ser desenvolvido enquanto estamos fazendo a modelagem de dados, tarefa esta que consiste em definir e estruturar os dados que serão manipulados e/ou gerados no sistema em questão. Esse modelo físico deve apresentar todos os detalhes das tabelas e seus relacionamentos. Nas tabelas, definimos os campos, seus tipos de dados e os índices.

**Índice**

Definimos um campo como índice para auxiliar na ordenação de dados e para agilizar processos de busca.

**Regras de validação**

Estas regras são usadas para garantir a consistência dos dados nos campos. Obrigatoriamente, os dados que serão digitados em um determinado campo devem obedecer às regras especificadas na consistência, ou seja, elas são uma expressão lógica para aceitação do dado.

**Texto de validação**

Trata-se da mensagem a ser exibida quando é quebrada a regra de validação.

**Criação de tabelas**

A seguir, veremos um exemplo de uma AGENDA.

**PLANEJAMENTO**

Banco de Dados: **DBAGENDA**

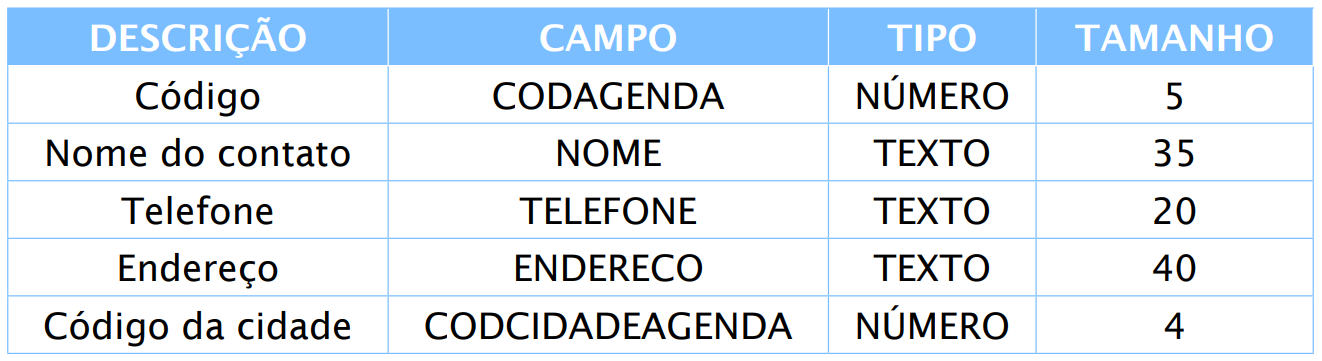
Tabela **TABAGENDA**

Campo chave primária = **CODAGENDA**



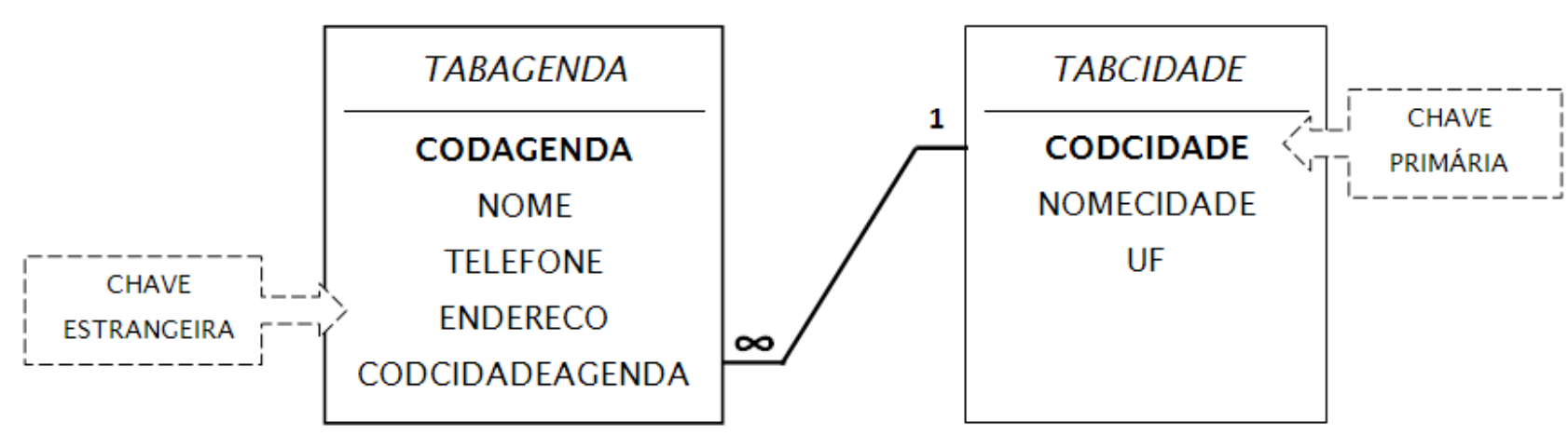
Tabela **TABCIDADE** (Tabela de Cidades)

Campo chave primária = **CODCIDADE**



**Relacionamento das tabelas**

Vemos adiante um exemplo de relacionamento entre tabelas:



• Os campos denominados **CHAVE PRIMÁRIA** estão em negrito;

• Um campo **CHAVE PRIMÁRIA** não pode ter valores duplicados, por isso neste caso não podemos utilizar o campo **NOME**, pois pode haver mais de um registro com o mesmo nome;

• Os nomes dos campos relacionados, ou seja, da chave primária e da chave estrangeira, não precisam ser os mesmos, mas os dados nos campos precisam coincidir;

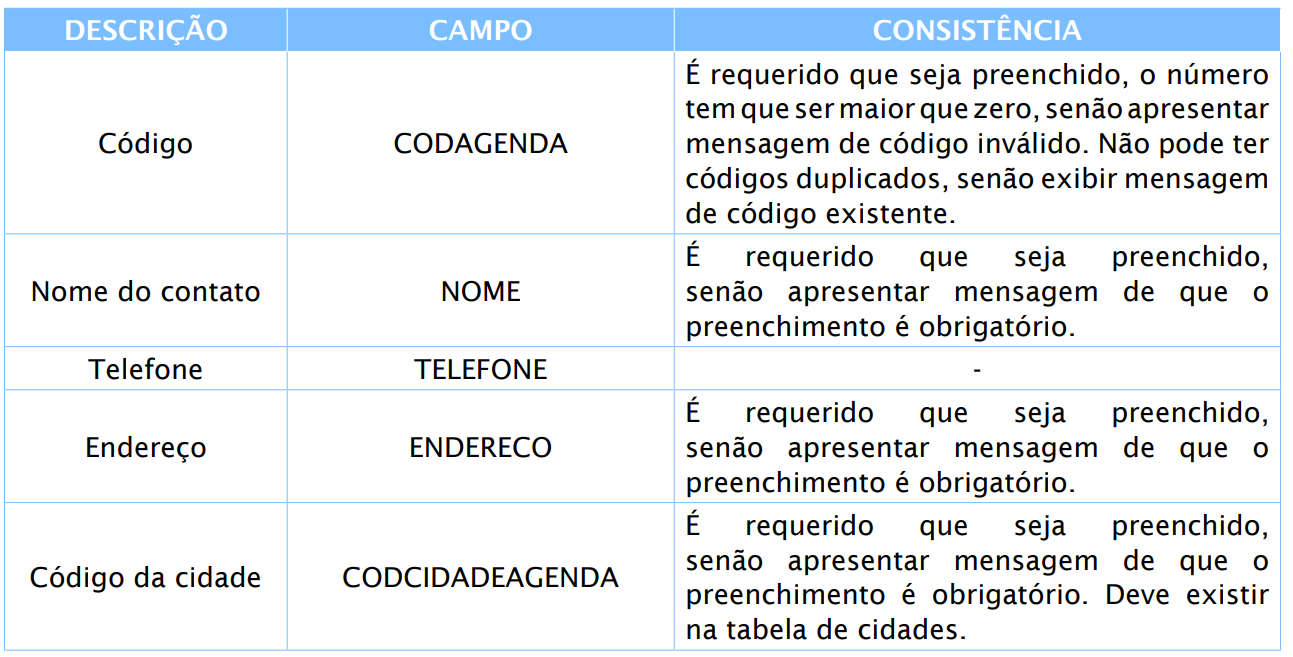
• Note que na TABCIDADE só pode haver um registro com o mesmo código, já na TABAGENDA pode haver muitos registros com o mesmo código de cidade. Relacionamento de um para muitos.

**Consistência dos campos**

A seguir, temos um exemplo de elaboração de regras de consistências, em que somente o preenchimento do telefone não é obrigatório na tabela da agenda.

Tabela **TABAGENDA**

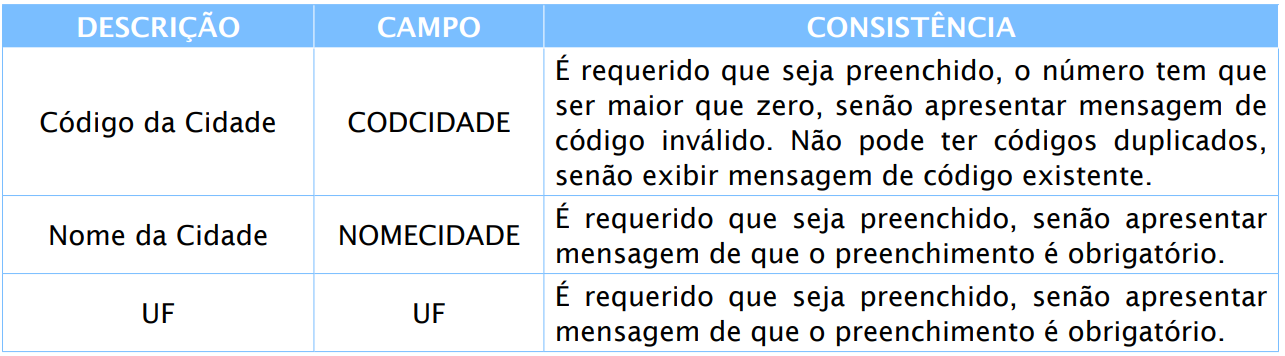
Campo chave primária = **CODAGENDA**



A seguir, temos a tabela relacionada:

Tabela **TABCIDADE** (Tabela de Cidades)

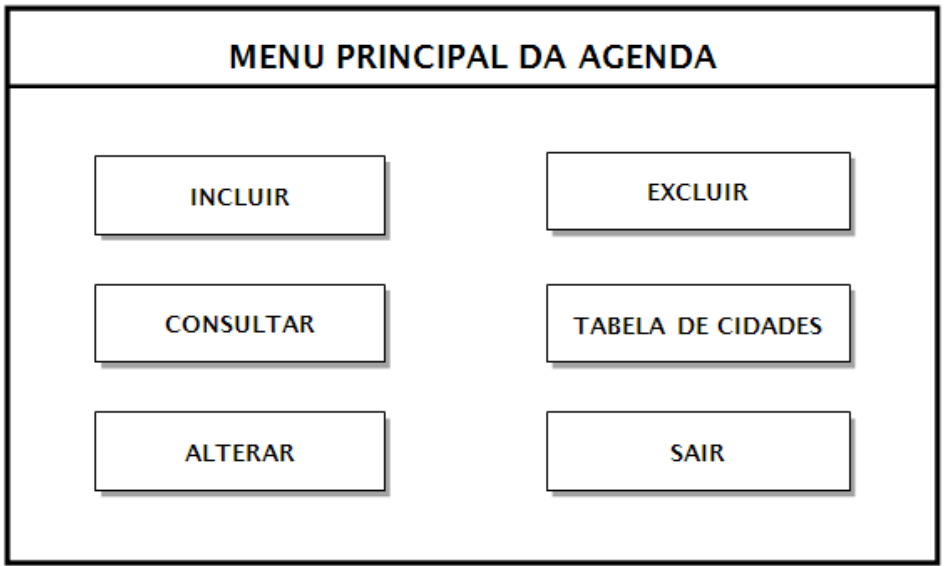
Campo chave primária = **CODCIDADE**



No sistema a seguir, considere que as consistências são efetuadas nas janelas das propriedades dos campos, não sendo necessário expressá-las no algoritmo.

**Sistema de controle de cadastro**

Independente da linguagem a ser utilizada, a lógica principal para atingir o objetivo é a mesma. A seguir, apresentamos um exemplo de tela inicial para o sistema de controle do cadastro da AGENDA.



**• Objetivo**

Os botões da tela anterior tem a finalidade de chamar os programas de inclusão, consulta, alteração e exclusão dos registros da tabela da agenda, e também chamar o programa principal da tabela de registros de cidades.

• **Processo**

Banco de dados utilizado: **DBAGENDA**

Tabela: **TABAGENDA**

• Botão **INCLUIR**: Chama o programa de INCLUSÃO de registros;

• Botão **CONSULTAR**: Chama o programa de CONSULTA de registros;

• Botão **ALTERAR**: Chama o programa de ALTERAÇÃO de registros;

• Botão **EXCLUIR**: Chama o programa de EXCLUSÃO de registros;

• Botão **TABELA DE CIDADES**: Chama a tela principal da tabela de cidades;

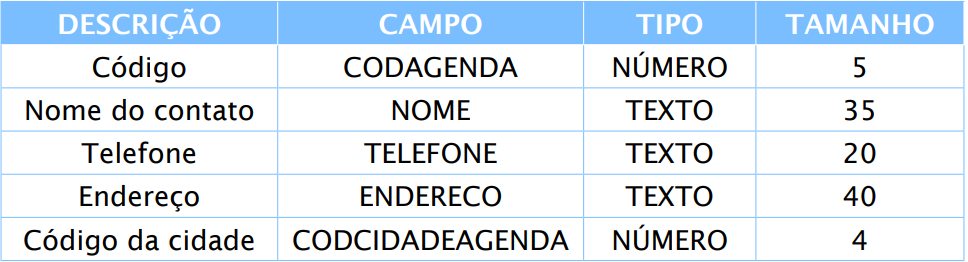
• Botão **SAIR**: Fecha o banco de dados e a tela, saindo do sistema.

**Programa de inclusão**

A seguir apresentamos o planejamento para o programa de inclusão de registros.

• **Objetivo**

Incluir registros na tabela **TABAGENDA**.



**Processo**

Para que os campos não sejam acessados diretamente, os dados a serem digitados nas telas serão armazenados em variáveis e depois gravados num registro da tabela. Serão utilizadas as variáveis a seguir para receber os respectivos dados:

• **VARCODAGENDA** = Código do registro

• **VARNOME** = Nome do contato

• **VARTELEFONE**= Telefone do contato

• **VARENDERECO** = Endereço do contato

• **VARCODCID** = Código da cidade

**Algoritmo**

1. INCLUSÃO

2. Declara VARCODAGENDA, VARCODCID numéricas e VARNOME, VARTELEFONE,

VARENDERECO alfanuméricas

3. VARCODAGENDA = 0

4. VARNOME = “ ”

5. VARTELEFONE = “ “

6. VARENDERECO = “ “

7. VARCODCID = 0

8. Abrir tabelas TABAGENDA e TABCIDADE

9. Abrir tela de inclusão de registros na tabela da TABAGENDA

10. Ler VARCODAGENDA

11. Buscar na tabela TABAGENDA no campo CODAGENDA = VARCODAGENDA

12. Encontrou o código?

12.1. Se sim: Exibir “Código já existente”

12.2. Vá para o passo 23

12.3. Se não: Próximo passo

13. Ler VARNOME, VARTELEFONE, VARENDERECO, VARCODCID

14. Buscar na tabela TABCIDADE no campo CODCIDADE = VARCODCID

15. Encontrou?

15.1. Se sim: Próximo passo

15.2. Se não: Exibir “Código inválido. Cidade não Existente.”

15.3. Vá para o passo 23

16. \*\*\* Comentário: Fazer os campos receberem o conteúdo das variáveis e depois

gravar o registro

17. CODAGENDA = VARCODAGENDA

18. NOME = VARNOME

19. TELEFONE = VARTELEFONE

20. ENDERECO = VARENDERECO

21. CODCIDADEAGENDA = VARCODCID

22. Gravar o registro na tabela TABAGENDA

23. Fechar tela de inclusão de registros da TABAGENDA

24. Fechar tabelas TABAGENDA e TABCIDADE

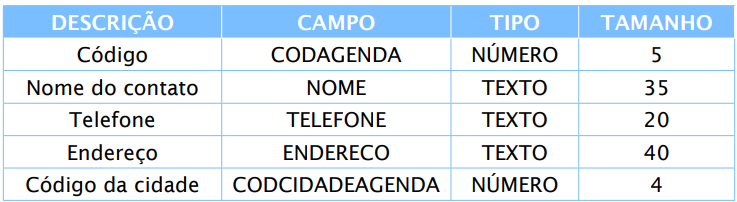
25. SAÍDA

**Programa de consulta**

A seguir, apresentamos o planejamento para o programa de consulta de registros.

**• Objetivo**

Consultar registros na tabela **TABAGENDA**



• **Processo**

A busca será feita pelo nome e, caso exista mais de um contato com o nome procurado, exibir na sequência um registro do outro. Após encontrar o registro na tabela, copiar os dados dos campos para as variáveis. Os dados serão exibidos nas telas em variáveis. Exibir o nome da cidade em vez do código.

Serão utilizadas as variáveis a seguir para receber os respectivos dados:

• VARCODAGENDA = Código do registro

• VARNOME = Nome do contato

• VARTELEFONE= Telefone do contato

• VARENDERECO = Endereço do contato

• VARCODCID = Código da cidade

• VARNOMECIDADE = Nome da cidade

**• Algoritmo**

1. CONSULTA

2. Declara VARCODAGENDA numérica e VARNOME, VARTELEFONE, VARENDERECO,

VARNOMECIDADE alfanuméricas

3. VARCODAGENDA = 0

4. VARNOME = “ ”

5. VARTELEFONE = “ “

6. VARENDERECO = “ “

7. VARNOMECIDADE = “ “

8. Abrir tabelas TABAGENDA e TABCIDADE

9. Abrir tela de consulta de registros na tabela da TABAGENDA

10. Ler VARNOME

11. Ordenar a tabela TABAGENDA pelo campo NOME

12. Buscar na tabela TABAGENDA no campo NOME = VARNOME

13. Encontrou o nome?

13.1. Se sim: Próximo passo

13.2. Se não: Exibir “Contato não Existente.”

13.3. Vá para o passo 22

14. Buscar na tabela TABCIDADE no campo CODCIDADE = CODCIDADEAGENDA

15. VARCODAGENDA = CODAGENDA

16. VARTELEFONE = TELEFONE

17. VARENDERECO = ENDERECO

18. VARNOMECIDADE = NOMECIDADE

19. Exibir na tela as variáveis VARCODAGENDA, VARNOME, VARTELEFONE,

VARENDERECO e VARNOMECIDADE

20. \*\*\* Comentário: Após a visualização do registro, verificar se existe outro registro

com o mesmo nome, senão fechar a tela.

21. O próximo registro tem o mesmo nome?

21.1. Se sim: Exibir os dados do outro registro indo para o passo 13

21.2. Se não: Próximo passo

22. Fechar tela de consulta de registros da TABAGENDA

23. Fechar tabelas TABAGENDA e TABCIDADE

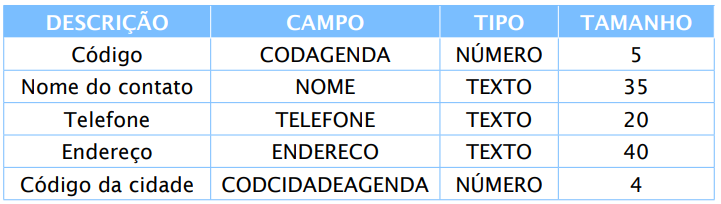
24. SAÍDA

**Programa de alteração**

A seguir, apresentamos o planejamento para o programa de alteração de registros.

**• Objetivo**

Alterar registros na tabela TABAGENDA



**• Processo**

A busca será feita pelo código do registro. Após encontrar o registro na tabela, copiar os dados dos campos para as variáveis. Os dados serão exibidos nas telas em variáveis. Exibir o código e o nome da cidade. Será permitida a alteração dos dados em algumas variáveis.

Serão utilizadas as variáveis a seguir para receber os respectivos dados:

• VARCODAGENDA = Código do registro

• VARNOME = Nome do contato

• VARTELEFONE= Telefone do contato

• VARENDERECO = Endereço do contato

• VARCODCID = Código da cidade

• VARNOMECIDADE = Nome da cidade

**• Algoritmo**

1. ALTERAÇÃO

2. Declara VARCODAGENDA, VARCODCID numéricas e VARNOME, VARTELEFONE,

VARENDERECO, VARNOMECIDADE alfanuméricas

3. VARCODAGENDA = 0

4. VARNOME = “ ”

5. VARTELEFONE = “ “

6. VARENDERECO = “ “

7. VARCODCID = 0

8. VARNOMECIDADE = “ “

9. Abrir tabelas TABAGENDA e TABCIDADE

10. Abrir tela de alteração de registros da TABAGENDA

11. Ler VARCODAGENDA

12. Buscar na tabela TABAGENDA no campo CODAGENDA = VARCODAGENDA

13. Encontrou o código?

13.1. Se sim: Próximo passo

13.2. Se não: Exibir “Código não existente”

13.3. Vá para o passo 30

14. Buscar na tabela TABCIDADE no campo CODCIDADE = CODCIDADEAGENDA

15. VARCODAGENDA = CODAGENDA

16. VARNOME = NOME

17. VARTELEFONE = TELEFONE

18. VARENDERECO = ENDERECO

19. VARCODCID = CODCIDADEAGENDA

20. VARNOMECIDADE = NOMECIDADE

21. Exibir na tela as variáveis VARCODAGENDA, VARNOME, VARTELEFONE,

VARENDERECO, VARCODCID e VARNOMECIDADE

22. Desproteger permitindo alteração as variáveis VARNOME, VARTELEFONE,

VARENDERECO e VARCODCID

23. \*\*\* Comentário: Fazer os campos receberem o conteúdo das variáveis e depois

gravar o registro

24. CODAGENDA = VARCODAGENDA

25. NOME = VARNOME

26. TELEFONE = VARTELEFONE

27. ENDERECO = VARENDERECO

28. CODCIDADEAGENDA = VARCODCID

29. Gravar o registro na tabela TABAGENDA

30. Fechar tela de alteração de registros da TABAGENDA

31. Fechar tabelas TABAGENDA e TABCIDADE

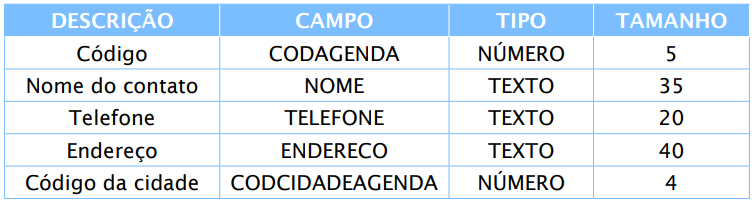
32. SAÍDA

**Programa de exclusão**

A seguir apresentamos o planejamento para o programa de exclusão de registros.

**• Objetivo**

Excluir registros na tabela TABAGENDA.



**• Processo**

A busca será feita pelo código do registro. Os dados serão exibidos nas telas em variáveis. Após encontrar o registro na tabela, copiar os dados dos campos para as variáveis. Os dados serão exibidos nas telas em variáveis. Exibir o código e o nome da cidade. Permitir a exclusão do registro exibido. Serão utilizadas as variáveis a seguir para receber os respectivos dados:

• VARCODAGENDA = Código do registro

• VARNOME = Nome do contato

• VARTELEFONE= Telefone do contato

• VARENDERECO = Endereço do contato

• VARCODCID = Código da cidade

• VARNOMECIDADE = Nome da cidade

**• Algoritmo**

1. EXCLUSÃO

2. Declara VARCODAGENDA, VARCODCID numéricas e VARNOME, VARTELEFONE,

VARENDERECO, VARNOMECIDADE alfanuméricas

3. VARCODAGENDA = 0

4. VARNOME = “ ”

5. VARTELEFONE = “ “

6. VARENDERECO = “ “

7. VARCODCID = 0

8. VARNOMECIDADE = “ “

9. Abrir tabelas TABAGENDA e TABCIDADE

10. Abrir tela de exclusão de registros da TABAGENDA

11. Ler VARCODAGENDA

12. Buscar na tabela TABAGENDA no campo CODAGENDA = VARCODAGENDA

13. Encontrou o código?

13.1. Se sim: Próximo passo

13.2. Se não: Exibir “Código não existente”

13.3. Vá para o passo 24

14. Buscar na tabela TABCIDADE no campo CODCIDADE = CODCIDADEAGENDA

15. VARCODAGENDA = CODAGENDA

16. VARNOME = NOME

17. VARTELEFONE = TELEFONE

18. VARENDERECO = ENDERECO

19. VARCODCID = CODCIDADEAGENDA

20. VARNOMECIDADE = NOMECIDADE

21. Exibir na tela as variáveis VARCODAGENDA, VARNOME, VARTELEFONE,

VARENDERECO, VARCODCID e VARNOMECIDADE

22. \*\*\* Comentário: Após exibir o registro perguntar se deseja excluí-lo e caso queira

limpar o conteúdo dos campos do registro

23. Deseja excluir o registro?

23.1. Se sim: Apagar o registro da tabela TABAGENDA

23.2. CODAGENDA = 0

23.3. NOME = “ “

23.4. TELEFONE = “ “

23.5. ENDERECO = “ “

23.6. CODCIDADEAGENDA = 0

23.7. Se não: Próximo passo

24. Fechar tela de exclusão de registros da TABAGENDA

25. Fechar tabelas TABAGENDA e TABCIDADE

26. SAÍDA

**Considerações finais**

A tela do MENU PRINCIPAL DA AGENDA possui o botão TABELA DE CIDADES, que dá acesso à tela do MENU PRINCIPAL DO CADASTRO DE CIDADES.

Este menu contém os botões de inclusão, consulta, alteração e exclusão de registros do cadastro de cidades. As funcionalidades dos botões são as mesmas dos botões do menu da agenda, porém se referem aos campos e registros da tabela TABCIDADE.

A tela do MENU PRINCIPAL DA AGENDA também possui o botão SAIR, que sai do sistema, fechando a tela principal e o banco de dados DBAGENDA.

O programa de CONSULTA exibe os dados na tela, o programa de ALTERAÇÃO exibe os dados na tela, permitindo alteração, e o programa de EXCLUSÃO exibe os dados na tela,

permitindo a exclusão. Poderíamos aproveitar o mesmo programa e, de acordo como botão selecionado, desviar o programa para a funcionalidade específica.

Nesses algoritmos, quando um código não é encontrado, a tela é fechada, retornando à tela do menu principal da agenda. Também pode ser programado para que fosse solicitado novamente um código.