ALGORITMOS E ESTRUTURAS DE DADOS

Algoritmos

Conceitos

- É um conjunto de instruções ou comandos que, quando executados , levam a um conjunto finito de ações.
- É um conjunto ordenado de instruções que quando seguidas desempenham uma tarefa especifica.

Características

- È rigoroso em sua definição
- Legibilidade
- Portabilidade

Programa.

É uma adaptação ou codificação de um algoritmo atendendo as regras de uma linguagem especifica.

Estrutura Básica de um Algoritmo

Um Algoritmo é normalmente dividido em 2 partes:

- A Primeira parte é aonde será declaradas tudo o que é necessário para a execução do algoritmo. Esta parte vai desde a palavra INICIO até a Paravra PROCEDA.
- A segunda parte é aonde serão colocados os comandos que serão executados pelo Algoritmo.Esta parte se inicia com a Palavra PROCEDA e vai até a palavra FIM.

Exemplo:

INICIO

Primeira Parte

PROCEDA

Segunda Parte

FIM.

É algo muito parecido com uma receita de bolo, aonde primeiro colocamos os ingredientes, e depois colocamos o modo de preparo , aonde detalhamos os procedimentos necessários para o preparo da receita.
A única diferença é que no lagoritmo procuramos fazer a “receita” de acordo com uma linguagem especifica. O uso desta linguagem evita que sejam feitas interpretações diferentes da mesma sentença como veremos a seguir:

ALGORITMOS E ESTRUTURAS DE DADOS

Problemas de Narrativa:

Utilização do Não, somente, mas, e/ou, A menos que:

- Conforme as palavra utilizadas para descrever uma função , poderão surgir interpretações diversas como nos exemplos abaixo:
- "Somar A e B a menos que A seja menor que B aonde neste caso subtrair A de B"

- "Somar A e B , entretanto se a for menor que B a resposta será a diferença entre A e B"

- "Somar A e B mas subtrair A de B quando A for menor que B"

- "O Total é a soma de A e B , somente quando A for menor é que a diferença deve ser utilizada no total"

Até , acima, abaixo:

- "Abaixo de 20 unidades não há desconto, acima de 20 unidades da direito a 5% de desconto"
- O que acontece com exatamente vinte unidades?
- Para solucionar este problema utilizar os operadores relacionais:

Ambigüidade do E/OU

"Clientes regulares que adquirirem mais de 1 Milhão e que tem um bom histórico de pagamento ou que estão conosco a mais de 20 anos devem receber tratamento especial"

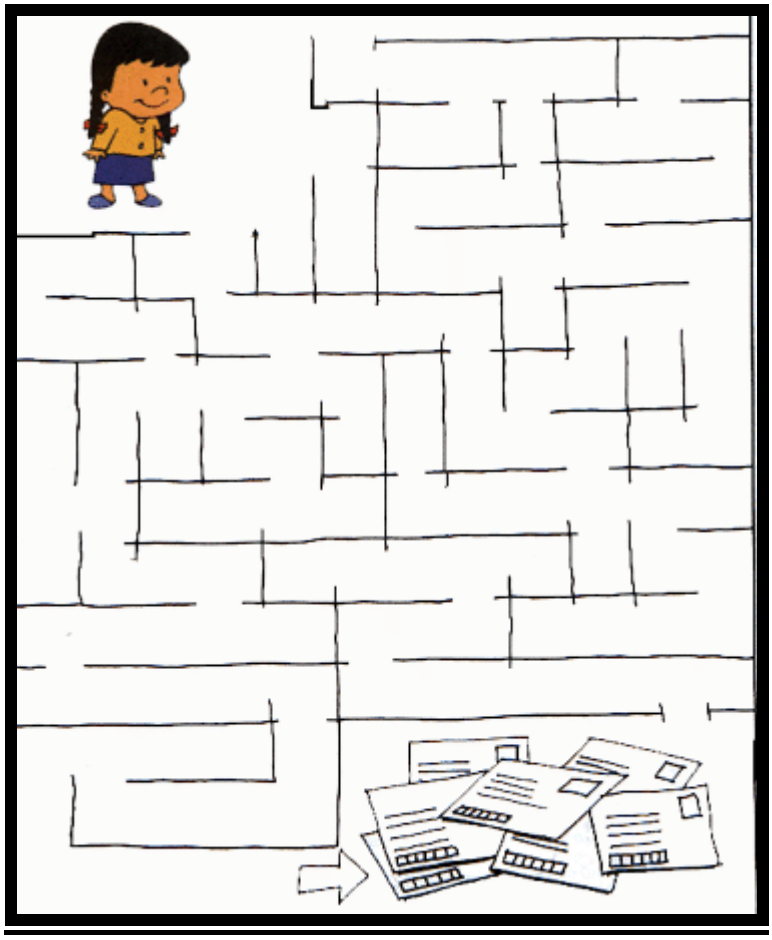
- Dependendo da entonação da frase podem surgir interpretações diferentes.

- Adjetivos indefinidos:
  O que é um bom Histórico?
  O que é um cliente regular?

- A frase "bom histórico" implica na existência de um Mau histórico, ambos deverão ser definidos?

Como pode ser visto acima a utilização da linguagem coloquial pode gerar diversas interpretações, e também muitas duvidas. Desta forma faz-se necessário a utilização de uma linguagem mais próxima a lógica dos computadores. E também , com uma linguagem bem parecida com as linguagens de programação, fica mais facil a transformação do algoritmo em um programa. O Algoritmo é então um intermediário entre a lingagem coloquial utilizada normalmente , e as linguagens de programação em outras palavras "Portugol"

Problemas de Interpretação:



Itens fundamentais:

Para que possamos entender e desenvolver algoritmos vamos precisar de alguns conceitos básicos:

Variáveis

Uma variável corresponde a uma área de memória , cujo conteúdo varia durante a execução do algoritmo.
Devemos encarar uma variável como o endereço de memória aonde esta guardada uma determinada informação. Esta informação pode ser de diversões tipos, mas somente um tipo de cada vez. O algoritmo poderá alterar este valor, de acordo com as suas instruções, quantas vezes forem necessárias, por isto elas são chamadas variáveis.

Tipos de Dados

Todos os dados processados por um algoritmo devem ser previamente definidos, isto porque o computador precisa saber previamente o tipo e o tamanho da informação que será processada.
Como já foi visto anteriormente, nós vamos armazenar estes dados dentro de variáveis, para que possamos alterar o seu valor durante a execução do algoritmo. Sendo assim , cada variável deverá ser declarada previamente e com um tipo já definido que deverá ser um dos tipos abaixo, ou uma de suas variações que serão vistas posteriormente.

Nomes de Variáveis

Para que possamos declarar uma variável devemos dar-lhe um nome. Dentro do algoritmo , toda vez que eu precisar usar o valor armazenado nesta variável , eu vou referencia-la pelo seu nome.

Regras para Nomes de Variáveis:

- Os nomes das variáveis devem representar o que será guardado dentro dela
- O primeiro caracter de um nome deverá ser sempre alfabético
- Não podem ser colocados espaços em branco no nome de variáveis, usar o **UNDERSCORE**.
- A declaração de uma variável é feita no algoritmo informando o seu nome, seguido pelo seu tipo, separados por ":"

ALGORITMOS E ESTRUTURAS DE DADOS

Tipos de Dados Primitivos

São os tipos de dados mais comuns, e que podem dar origem a outros tipos de dados mais complexos.

Inteiro

Números inteiros maiores ou menores que 0 representados por 2 bytes, em uma faixa que vai de -32.768 até 32.767.

Declaração:

```
NUMERO : INTEIRO
```

No exemplo acima foi declarada uma variável do tipo **INTEIRO** que atende pelo nome de "**NUMERO**"

Real

Conjunto dos números racionais. representado por 4 bytes.

Declaração:

```
SALÁRIO : REAL
```

No exemplo acima foi declarada uma variável do tipo **REAL** com o nome de "**SALÁRIO**".

Caractere (Char)

Conjunto dos caracteres alfanuméricos (números , letras , símbolos, etc).Representado por apenas um byte.
Note que as variáveis do tipo **CHAR** podem armazenar apenas 1 caracter.

Declaração:

```
SEXO : CHAR
```

No exemplo acima foi declarada uma variável do tipo **CHAR** com o nome "**SEXO**"

Lógico

Quando assume apenas 2 valores:

- **FALSO**
- **VERDADEIRO**

Declaração

```
FLAG : LÓGICO
```

No exemplo acima, foi declarada uma variável do tipo **LÓGICO** com o nome de **FLAG**.

Agregados de Dados

Os agregados são estruturas formadas a partir dos tipos de dados primitivos, e que permitem o processamento de informações mais aprimoradas, são eles:

Cadeia de Caracteres (String)

É um agradado de dados homogêneo, ou seja, usa somente um tipo primitivo, aonde todos os elementos são do tipo caractere (**CHAR**)

Declaração:

NOME_CLI : STRING [40]

No exemplo acima foi declarada uma variável do tipo **STRING** , que poderá ter no máximo 40 posições e o seu nome é **NOME_CLI**

Matriz

É um agregado de dados do tipo homogêneo , aonde todos os elementos podem ser de qualquer tipo , desde que todos os componentes sejam do mesmo tipo.

Declaração:

NUMERO : MATRIZ [4 , 5] DE INTEIRO

No exemplo acima , foi declarada uma **MATRIZ** com 4 colunas e 5 linhas , aonde todos os elementos serão do tipo **INTEIRO**. Note que o primeiro numero sempre representa o numero máximo de colunas que uma matriz pode ter, e o segundo numero representa o numero máximo de linha que uma matriz pode ter.

Note que podem existir matrizes unidimensionais, os chamados vetores. Neste caso a matriz possui apenas linhas ou colunas. neste caso não será necessário informar a coluna e a linha , mas somente um numero , não importando se ele representa a quantidade de colunas ou linhas.

Registros

É o único agregado de dados heterogêneo, ou seja pode combinar dados de diferentes tipos em uma única estrutura.

Declaração:

```
REG_DADOS : REGISTRO
        NOME : STRING [ 40 ]
        SEXO : CHAR
        IDADE : INTEIRO
        SALARIO : REAL
FIM_REGISTRO
```

Note que o **REGISTRO** é formado por 4 **CAMPOS**, e cada campo é de um tipo diferente, entretanto , todos os campos fazem parte de uma única estrutura, que atende pelo nome de **REG_DADOS**.

Além das variáveis existem outros elementos que irão fazer parte dos algoritmos.

Declaração de Variáveis:

Exercício 01

Declare variáveis que sejam capazes de armazenas as seguintes informações

- Sua Idade
- A Média entre 2 numeros
- O seu Salário
- Seu nome, numero, nota do primeiro bimestre, nota do segundo bimestre e media final
- Os 12 meses dos ano (Só o numro do mês) com os sues respeqtivoas números de dias

Variaveis:

Nome; String;
Media: real
Salario:real

Comentários.

O comentário é um texto delimitado por chaves, cuja função é explicar com mais clareza alguma parte especifica do algoritmo. O comentário é um texto que não deve ser levado em consideração na hora de executar o algoritmo, ele serve apenas para dar informações adicionais a leitura do algoritmo.

ALGORITMOS E ESTRUTURAS DE DADOS

Operadores

Operadores são sinais utilizados nos algoritmos para indica a realização de operações que podem ser de três categorias:

- **Aritméticas**
- **Relacionais**
- **Lógicas**

Operadores Aritméticos

Servem para realizar operações com números inteiros ou reais.
são eles:

OPERADOR	SINAL	PRIORIDADE
ADIÇÃO	+	3
SUBTRAÇÃO	-	3
MULTIPLICAÇÃO	*	2
DIVISÃO	/	2
POTÊNCIA	↑	1

Operadores Relacionais

São operadores que servem para estabelecer relações entre variáveis do mesmo tipo.
São eles

OPERADOR	SINAL	PRIORIDADE
IGUAL	=	1
DIFERENTE	< >	1
MENOR QUE	<	1
MAIOR QUE	>	1
MAIOR OU IGUAL	> =	1
MENOR OU IGUAL	< =	1

Operadores Lógicos

São operadores utilizados para unir duas relações criadas através dos operadores relacionais são eles:

OPERADOR	SIGNIFICADO	PRIORIDADE
E	ESPECIFICA QUE TODAS AS CONDIÇÕES RELACIONADAS DEVEM SER VERDADEIRAS PARA QUE O CONJUNTO SEJA VERDADEIRO	1
OU	ESPECIFICA QUE SE APENAS UMA DAS CONDIÇÕES RELACIONADAS FOR VERDADEIRA , O RESULTADO DELAS TAMBÉM SERÁ VERDADEIRO.	1
NAO	ESPECIFICA A NEGAÇÃO DA CONDIÇÃO RELACIONADA, OU SEJA A INVERSÃO DOS VALORES	1

Resumo das Prioridades.

PRIORIDADES	ITENS
1	PARÊNTESES E FUNÇÕES
2	EXPRESSÕES ARITMÉTICAS
3	RELAÇÕES
4	NÃO
5	E
6	OU

Funções

As funções são algoritmos pré-definidos que irão retornar valores normalmente resultantes de operações aritméticas.
Estas funções têm por finalidade agilizar o desenvolvimento dos nossos algoritmos, já que os procedimentos para calculo destas funções não são de grande importância.
Mais tarde estudaremos mais a fundo as funções , por hora basta saber como elas são utilizadas em nossos algoritmos.

São elas:

FUNÇÃO	**RETORNO**
SQR (P)	RAIZ QUADRADA DE P
ABS (P)	VALOR ABSOLUTO DE P
TRUNCA (P)	PARTE INTEIRA DE P
ARREDONDA(P)	INTEIRO MAIS PROXIMO DE P
EXP (P)	FUNÇÕES EXPONENCIAL DE P
QUOC (P,Q)	QUOEFICIENTE INTEIRO DE P POR Q
RESTO (P,Q)	RESTO DA DIVISÃO INTEIRA DE P POR Q
LOG (P)	LOGARITMO NA BASE 10 DE P
LN (P)	LOGARITMO NEPERIANO DE P

Comando de Atribuição.

É o comando usado para atribuir um valor a uma variável.
A natureza deste valor deve ser compatível com o tipo da variável declarada.
O sinal utilizado para indicar uma atribuição é o **: =** (dois pontos igual)
É importante neste ponto esclarecer com é que funciona o processo de atribuição:
Caso eu esteja atribuindo para uma variável o resultado de uma expressão, o algoritmo primeiro deverá resolver a expressão para depois fazer a atribuição.

Exemplo

```
A: = 3 + (2 * 6)
MOSTRE ( A )  {RESULTADO 15}
```

O resultado deste comando será 15, porque o algoritmo primeiro resolve a expressão e depois guardou o valor conseguido na variável especificada (A).
Também podem ser utilizados os valores armazenados em outras variáveis, para se fazer uma atribuição:

```
A: = 5
B : = 3
C: = (A + B) / 2
MOSTRE (C) {RESULTADO 4}
```

No final desta seqüência de comandos o valor dentro da variável c será 4, pois o algoritmo utilizou o valor das variáveis **A** e **B** para calculara o valor da expressão e depois o armazenou em **C**.
Também pode ser utilizado o valor da própria variável que receberá a atribuição:

```
A: = 7
B: = 5
C : =9
C: = (A + B + C) / 3
MOSTRE (C) {RESULTADO 7}
```

No final desta seqüência o valor de **C** será 7 , pois o algoritmo primeiro utiliza o valor 9 armazenado em **C** para resolver a expressão, e depois de resolvida , o novo valor será armazenado em **C** , apagando o valor anterior.
Isto fica mais claro no exemplo abaixo:

```
CONTADOR : = 1
CONTADOR : = CONTADOR + 1
MOSTRE (CONTADOR) { RESULTADO 2}
```

A variável **CONTADOR** foi inicializada com o valor 1 , e logo abaixo este valor foi utilizado em uma expressão, e o resultado desta expressão foi armazenado dentro da própria variável **CONTADOR**. Mas é importante notar que o algoritmo primeiro utilizou o valor 1 para resolver a expressão , e somente após ter a resposta é que ele alterou o valor da variável **CONTADOR**.
Na verdade eu apenas somei 1 ao valor que já existia na variável.

Comandos de Entrada E Saída

Um comando de entrada permite a entrada de dados do meio externo para o computador, e um comando de saída permite a saída de dados do computador para o meio externo.

São eles:

Comando Leia

Permite que o usuário entre com um valor , que será armazenado em uma variável e poderá ser utilizado pelo algoritmo.
O comando **Leia** interrompe a execução do algoritmo até que o usuário digite o valor solicitado, quando então, armazena este valor na variável especificada no comando e dá proceguimento a execução do Algoritmo.
O comando **Leia** não permite que se faça uma critica dos valores digitados, por isso, se for necessário esta critica , ela deverá ser implementada no próprio algoritmo.
A nível de Algoritmo, não será necessário implementar este tipo de critica.

Sintaxe:

Leia (Variável)

Onde:

- **Variável** é o nome da variável aonde o valor digitado pelo usuário será armazenado. Note que o valor digitado deverá ser do mesmo tipo da variável.

Comando Mostre:

O comando **Mostre** permite que sejam mostrados para o usuário os valores armazenados em variáveis ou então expressões literais.
Uma vez que o algoritmo tenha cumprido as suas funções, o comando **Mostre** é a forma de mostrar ao usuário os resultados do processamento.
Alem disso, o comando **Mostre** também pode ser utilizado para mostrar mensagens que ajudaram o usuário na utilização do algoritmo.

Sintaxe:

Mostre (Variável)
Mostre (Expressão Literal)

Onde:

Variável é o nome da variável cujo valor se deseja mostrar
Expressão Liretal é uma frase ou expressão que se deseja mostra para o usuário. Ela deve estar entre Aspas (““)

Expressões Literais

As expressões literais são frases ou outro tipo de expressões que são colocadas no algoritmo com mensagens ou para atribuição em variáveis do tipo **STRING** ou **CHAR**. Estas expressões são identificadas por estarem entre aspas (" ").
É importante notar que nada impede que um numero seja colocado dentro de uma expressão literal, ou em uma variável **STRING** ou **CHAR**, mas note que este tipo de numero não vai possuir o valor numérico, ou seja, este numero não vai poder ser usado para realizar operações aritméticas. Para este caso nós devemos utilizar números armazenados em variáveis do tipo inteiro ou real.

Exemplo:

```
INICIO
        NUM : INTEIRO
        X : STRING [ 6 ]
PROCEDA
        NUM: = 999999
        X : = "999999"
        NUM : = NUM + 1
        MOSTRE (NUM)    { RESULTADO  IGUAL  A 1000000}
        X: = X + 1          { ESTE COMANDO É INVALIDO JÁ QUE X É DO TIPO STRING}
        MOSTRE (X)          {RESULTADO IGUAL A "999999"}
FIM
```

Note que no exemplo acima eu declarei 2 variáveis, uma do tipo **INTEIRO** e outra do tipo **STRING**, e que eu atribui para ambas o valor 999999, mas no caso da variável do tipo **STRING** eu usei as aspas para indicar que este valor é uma expressão literal, e que por isso não tem valor numérico. Este tipo de expressão é muito útil no caso de termos que armazenar informações como endereço ou telefones, que exigem a presença de números e letras.
No caso da variável do tipo **INTEIRO** o valor 999999 foi atribuído sem as aspas, já que a eu estou utilizando o valor numérico, e poderei usar este numero em qualquer operação aritmética.

Estruturas de Controle.

São as formas como serão executados os comandos do algoritmo, são elas.

Estrutura Seqüencial

É a mais simples, eficiente e legível das estruturas de controle.
Estabelece que uma declaração passa a ser analisada imediatamente após o termino de sua predecessora.
Um algoritmo se inicia com a palavra **INICIO**, após o que, aparecem as declarações de variáveis, seguidas de comandos que, se não houverem indicações ao contrario , serão executados um depois do outro em seqüência.
A finalização do algoritmo é feita com a palavra **FIM**.

Exemplo

```
INICIO
        A,B : INTEIRO
        MEDIA : REAL
PROCEDA MEDIA {CALCULO DA MEDIA ENTRE 2 NUMEROS}
        LEIA ( A )
        LEIA ( B )
        MEDIA : = (A + B ) / 2
        MOSTRE  ( MEDIA )
FIM.
```

Estruturas de Seleção

São usadas quando é necessário fazer uma opção entre 2 ou mais caminhos sendo que o fluxo do algoritmo é desviado segundo uma condição lógico relacional.
Esta estrutura subordina a execução de um comando, ou bloco de comandos, a veracidade ou não de uma condição lógico relacional.
Os comandos utilizados para implementar a estrutura de seleção são :

- **SE**
- **SE SENÂO**
- **CASO**

Comando SE

O comando **SE** especifica se um comando, ou um conjunto de comando será ou não executado, de acordo com uma condição lógico relacional.

Sintaxe:

SE <CONDIÇÃO>

BLOCO
DE
COMANDOS

FIM_SE

O comando **SE** acima especifica que o bloco de comandos que esta entre o inicio do comando até a palavra **FIM_SE** somente será executado se **<CONDIÇÃO>** for verdadeira, caso a condição for falsa, o algoritmo vai prosseguir normalmente após a palavra **FIM_SE.**

Comando SE SENÂO

O comando **SE** ainda permite que seja associado a ele um segunda opção, o comando **SENÃO**.
O comando senão pode ou não ser associado ao comando SE, uma vez associado, ele especifica que os comandos a ele subordinados só serão executado no caso da condição lógico relaciona for falsa, ou seja, só será executado caso os comandos associados ao comando SE não forem executados.

Sintaxe:

SE < CONDIÇÃO >

BLOCO DE COMANDOS A

SENÃO

BLOCO DE COMANDOS B

FIM_SE

No comando acima, o bloco de comandos **A** só será executado no caso da condição lógico relaciona for verdadeira, e o bloco de comando **B** só será executado no caso da condição lógico relacional for falsa. De qualquer forma, após executar tanto o bloco A como o bloco **B** o algoritmo seguirá normalmente a partir da palavra **FIM_SE**.

Comando CASO

É o comando que permite a opção entre várias alternativas de acordo com a verificação de uma condição lógico relacional.

Sintaxe:

```
CASO
        MES = 1
                SALARIO : = SALARIO * 1.53
        MES = 2
                SALARIO : = SALARIO * 1.72
        MES = 3
                SALARIO : = SALARIO * 1.83
FIM_CASO.
```

Note que no comando **CASO**, a mesma variável será testada com várias condições, caso uma delas seja atendida, o comando, ou bloco de comandos associados a esta condição será executado, e logo após o algoritmo seguirá normalmente após a palavra **FIM_CASO**.
No caso de nenhuma condição ser verdadeira, o algoritmo seguirá normalmente após a palavra **FIM_CASO** sem executar nenhum comando associado ao comando **CASO**.
O comando Caso é Equivalente a uma sequancia de comando **SE SENÂO**

Embutimento

Uma estrutura se seleção pode ter uma outra estrutura de seleção dentro dela, e assim por diante. Neste caso nós podemos colocar um comando **SE** subordinado a outro comando **SE**, ou um comando **SE** subordinado a um comando **CASO** , ou vice e versa, sem limite de encadeamento.

É importante ressaltar que para formação das condições dos comandos de seleção serão utilizados operadores relacionais associados ou não a operadores lógicos.

ALGORITMOS E ESTRUTURAS DE DADOS

Exercicio 02:

Monte as estruturas de seleção necessárias para representar as seguintes situações:

- Clientes regulares que adquirirem mais de 1 Milhão e que tem um bom histórico de pagamento ou que estão conosco a mais de 20 anos devem receber tratamento especial caso contrario tratamento normal.
- O subsidio de um escoteiro Lobinho pode variar de acordo com o posto e com o tempo de associado da seguinte forma:
  O subsidio para membros a um ano é de 25 Reais, de 35 reais para membros a dois anos, e de 50 reais para membros a mais de 2 anos, alem disso cada escoteiro recebe um subsidio extra de 10 reais de obteve o posto de Lobo, 20 reais de obteve o posto de Urso, e 20 reais de obteve o posto de Leão.
- Uma Loja da Cidade esta oferecendo aos seus clientes as seguintes promoções:
  - Pagamento a vista 50% de desconto
  - Pagamento em cheque 40% de desconto
  - Pagamento com cheque pré datado 30 dias(20%)
  - Pagamento com cheque pré datado 30 e 60 dias(10%)
  - Pagamento com cheque pré datado 30 60 e 90 dias(5%)
  - Cartão de Credito 30% de desconto
  - Alem disso cliente cadastrado tem um desconto de 5% adicional
- Suponha que o Professor de Educação física nos pediu para pesquisar os registros dos alunos e produzir uma lista com o nome e endereço de todos os alunos e o esporte indicado para eles de acordo com as seguintes condições:
  - Sexo masculino com mais de 1,80 metros de altura=> Basquete
  - Sexo masculino entre 1,70 e 1,80=> Voleibol
  - Sexo feminino com mais de 1,70 => Basquete
  - Sexo feminino entre 1,50 e 1,70 => Voley
  - Ambos os sexos com menos de 1,50 => natação

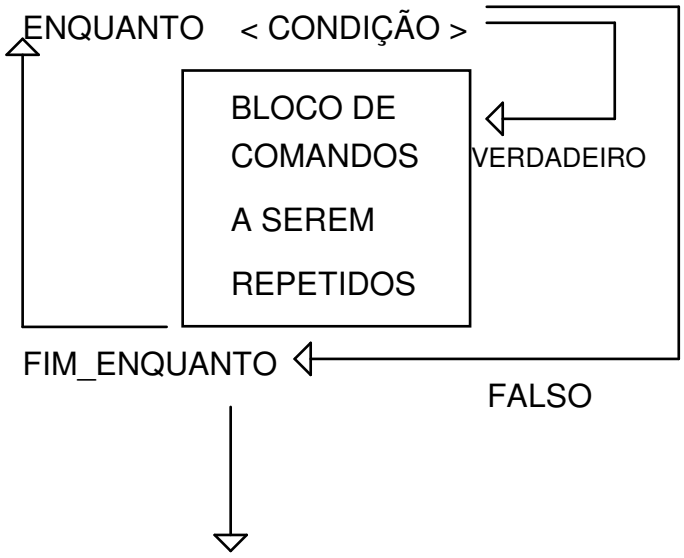
Estruturas de Repetição

São usadas sempre que for necessário repetir um conjunto de declarações para manter a abstração de controle do algoritmo .
São as mais complicadas, e as que exigem o maior cuidado na sua escolha e aplicação.

Comando ENQUANTO.

Prescreve que os comandos a ele subordinados deverão ser repetidos **ENQUANTO** uma condição lógico relacional for verdadeira.
O teste é feito no inicio do comando , e caso o resultado for verdadeiro os comandos serão executados , logo em seguida outro teste será feito. Caso o teste seja falso, o algoritmo continua normalmente após a palavra **FIM_ENQUANTO.**
Note que se o primeiro teste de falso , os comando não serão executados nenhuma vez, e o algoritmo continuará normalmente após a palavra **FIM_ENQUANTO.**

Sintaxe:



Exercício 03:

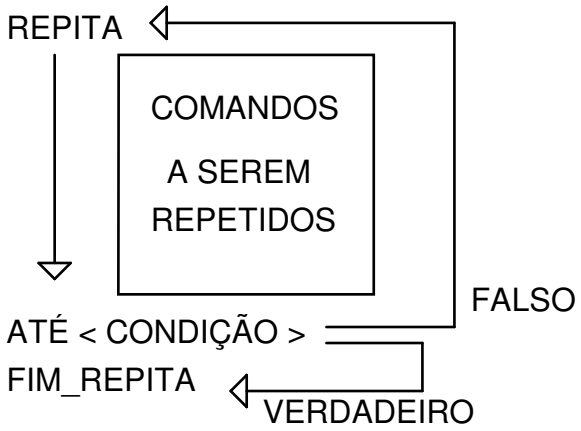
Faça um algoritmo capaz de somar todos os números inteiros compreendidos entre 1 e 10, e mostrar o resultado da soma.

```
INICIO
        SOMA,CONT : INTEIROS
PROCEDA SOMA
        CONT : = 1
        SOMA : = 0
        ENQUANTO CONT < = 10
                SOMA : = SOMA + CONT
                CONT : = CONT + 1
        FIM_ENQUANTO
        MOSTRE ( SOMA)
FIM.
```

Comando REPITA.

Prescreve que os procedimentos a ele subordinados deverão ser repetidos **ATÉ** que uma condição lógico relacional seja verdadeira.
O teste será feito no final do comando, e caso seja **FALSO** , os comando que estão entre o **REPITA** , e o **FIM_REPITA** ,serão repetidos, e novamente será feito um teste. Quando o resultado do teste for verdadeiro, o algoritmo continua normalmente a partir da palavra **FIM_REPITA**.
Note que sendo o teste feito no final do comando, os comando subordinados ao repita serão executados pelo menos uma vez.

Sintaxe:



Exercício 04:

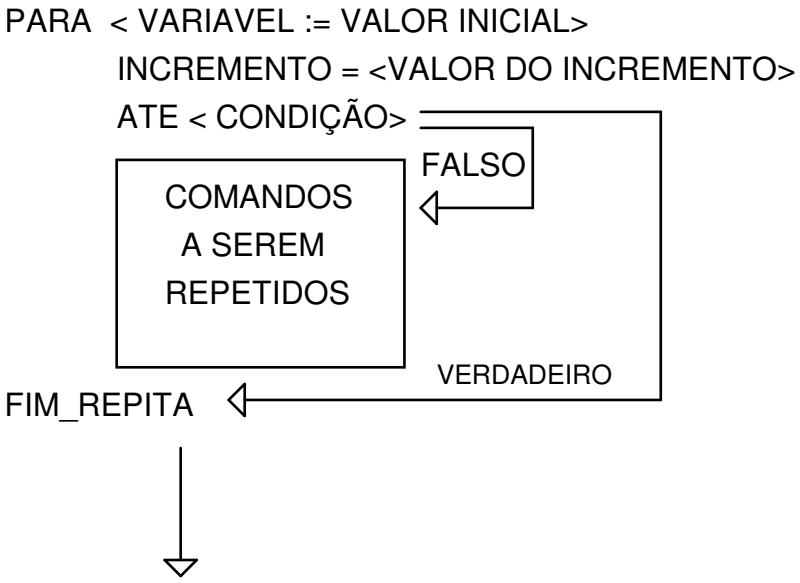
Idem ao execicío 03 só que utilizando o comando **REPITA**.

```
INICIO
        SOMA,CONT : INTEIRO
PROCEDA SOMA
        CONT: = 1
        SOMA : = 0
        REPITA
                SOMA : = SOMA + CONT
                CONT : = CONT + 1
        ATE CONT > 10
        FIM_REPITA
        MOSTRE ( SOMA)
FIM.
```

Comando PARA

O comando **PARA** prescreve que os comandos a ele subordinados deverão ser repetidos , a partir de uma variável "**I**", inicializada pelo comando, com um incremento "**J**" , até que uma condição lógico relacional seja verdadeira.
O teste será feito no inicio dos procedimentos, sendo que se na primeira comparação o resultado for verdadeiro os procedimentos não serão executados nenhuma vez.
A diferença do comando **PARA** em relação aos outros, é que ele automaticamente incremente uma variável de controle, sendo que esta variável de controle deverá ser utilizada na formação da condição de parada do comando. O comando **PARA** é recomendado para situações aonde nos tenhamos um numero fixo de interações, e que sejam necessários incrementos fixo.

Sintaxe:



Exercício 05

Idem ao exercicio 03 só que utilizando o comando **PARA**

```
INICIO
        SOMA, CONT : INTEIRO
PROCEDA SOMA
        SOMA : = 0
        PARA CONT : = 1
        INCREMENTO = 1
        ATE CONTA > 10
                SOMA : = SOMA + CONT
        FIM_PARA
        MOSTRE ( SOMA)
FIM_PARA
```

Regras para Codificação de Algoritmos.

1. FAÇA O ALGORITMO CERTO NA PRIMEIRA VEZ
2. NOMES DEVEM SE ASSEMELHAR AQUILO QUE REPRESENTAM
3. TODO ALGORITMO DEVE , A PARTIR DE UMA ESTRUTURA SIMPLES E FÁCIL , DAR UMA RESPOSTA ADEQUADA AO PROBLEMA QUE SE PROPÕEM
4. RESPEITE A ESTÉTICA, ESTRUTURAS SUBORDINADAS DEVEM SER ENDENTADAS
5. NUNCA ESQUEÇA A PRIMEIRA REGRA.

Regras para Declaração de Variáveis:

Umas das coisas mais difíceis de se fazer quando esta esta no inicio do aprendizado de algoritmos é decidir quais as variaves serão necessárias para o funcionamento de algoritmo.
Seguem algumas resgras que poderão ajudar neste problema.

- O usuário vai informar alguma coisa para o algoritmo?
  Então se deve declarar uma variavel para armazenar cada valor que sera informado pelo usuário.
- O algoritmo vai realizar algum calculo?
  Então deveremos criar uma variavel para armazenar o resultado de cada calculo.
- O algoritmo vai ter estruturas de repetição?
  Então devemos declarar uma variável para controlar esta estrutura de repetição.

È claro que estas regras não cobrem todos os casos, mas servirão para ajudar no inicio do aprendizado.

Exercícios

Exercício 06:

Faça um algoritmo para Somar todos os numero inteiros compreendidos entre 2 numero, sendo que:

- Os dois numeros serão informados pelo usuário
- O numero inicial não poderá ser maior ou igual ao numero final

```
INICIO
        SOMA: INTEIRO {VARIAVEL PARA ARMAZENAR A SOMA DOS NUMEROS}
        NUMERO_INICIAL : INTEIRO { PARA LEITURA DO NUMERO INICIAL DA
        SEQÜÊNCIA}
        NUMERO_FINAL : INTEIRO {PARA LEITURA DO NUMERO FINAL DA
        SEQÜÊNCIA}
        CONT: INTEIRO { VARIÁVEL DE CONTROLE DO LOOP}
PROCEDA CALCULA_SOMA
        REPITA
                LEIA ( NUMERO_INICIAL)    {USUÁRIO INFORMA OS NÚMEROS}
                LEIA (NUMERO_FINAL)
                SE NUMERO_INICIAL > = NUMERO_FINAL
                        MOSTRE ( "NÚMEROS INVÁLIDOS " )  { NÚMEROS INVÁLIDOS}
                FIM_SE
        ATE NUMERO_INICIAL  <  NUMERO_FINAL {VAI REPETIR ATE NUMERO SEJAM
        VÁLIDOS}
        SOMA : = 0
        PARA CONT : = NUMERO_INICIAL
        INCREMENTO = 1
        ATE CONTA CONT > NUMERO_FINAL
                SOMA : = SOMA + CONT    {CALCULO DA SOMA}
        FIM_PARA
        MOSTRE ( SOMA) { MOSTRA OS RESULTADOS}.
FIM.
```

Exercício 07:

Sendo a formula de uma equação do segundo grau:

$X = (-B \pm SQR(B^2 - 4*A*C)) / 2*A$

Desenvolver um algoritmo capaz de calcular as raízes de uma equação sendo que:

- Os valores A , B , C serão informados pelo usuário.
- O Valor de a não pode ser Zero
- O algoritmo deve informar caso não existam raízes reais.

```
INICIO
        A , B , C , DELTA , X1 , X2 : REAL
PROCEDA EQUAÇÃO
        LEIA (A)
        SE A = 0
                MOSTRE("ESTE VALOR NÃO PODE SER ZERO")
        SENÃO
                LEIA (B)    {USUÁRIO INFORMA OS VALORES A,B,C}
                LEIA (C)
                DELTA : = B ↑ 2  - ( 4 * A * C )
                SE DELTA < 0
                        MOSTRE ( "NÃO EXISTEM RAIZES PARA ESTE VALORES")
                SENÃO
                        X1 : = (( - B + SQR (DELTA)) / ( 2*A))
                        X2 : = (( - B -  SQR (DELTA)) / (2*A))   {CALCULO DAS RAIZES}
                        MOSTRE (X1)
                        MOSTRE (X2)             {INFORMA OS RESULTADOS}
                FIM_SE
        FIM-SE
FIM.
```

Exercício 08:

Desenvolver um algoritmo capaz de calcular a media aritmética simples de uma série de números , sendo que:

- O usuário deve informar a quantidade de números da série.
- Se a quantidade for zero ou negativa o algoritmo não deve aceita-la.
- O usuário deve informar um por um todos os números da série.
- O algoritmo deve mostrar como resultado a media aritmética simples calculada através da formula:
  MEDIA = (X + X1 + X2+...XN) / N.

```
INICIO
        SOMA: INTEIRO {PARA ARMAZENAR A SOMA DE TODOS OS NÚMEROS
        DIGITADOS}
        NUM : INTEIRO {PARA LEITURA DE CADA NUMERO INFORMADO}
        N:  INTEIRO {A QUANTIDADE DE NÚMEROS DA SÉRIE}
        MEDIA : REAL {PARA O CALCULO DA MÉDIA}
PROCEDA
        LEIA ( N )  { QUANTIDADE DE NÚMEROS DA SÉRIE}
        SE N < = 0
                MOSTRE ( "NUMERO NEGATIVO")
        SENAO
                CONT : = 1
                SOMA: = 0
                ENQUANTO CONT < =  N
                        LEIA (NUM)
                        SOMA : = SOMA + NUM  {CALCULO DA MÉDIA}
                FIM_ENQUANTO
                MEDIA : = SOMA / N
                MOSTRE (MEDIA)
        FIM_SE
FIM.
```

Exercício 09

Desenvolver um algoritmo capaz de calcular o fatorial de um numero inteiro positivo informado pelo usuário, sendo que:

- O numero para o calculo deve ser informado pelo usuário
- O algoritmo deve informar o fatorial deste numero de acordo com as regras abaixo:
- O fatorial de numero negativos não existe
- O fatorial de 0 é 1
- O fatorial de 1 é 1
- O fatorial de um numero qualquer é da do pela seguinte formula:
  5! = 5*4*3*2*1
  4! = 4*3*2*1
  N! = N * (N-1) * (N-2)...* 1

```
INICIO
        NUM:   INTEIRO {PARA LEITURA DO NUMERO}
        FATORIAL : INTEIRO {PARA O CALCULO DO FATORIAL}
PROCEDA FATORIAL
        LEIA (NUM)
        SE NUM < 0
                MOSTRE (“NUMERO INVÁLIDO”)
        SENAO
                FATORIAL : = 1
                ENQUANTO NUM > 0
                        FATORIAL : = FATORIAL * NUM
                        NUM: = NUM - 1
                FIM_ENQUANTO
                MOSTRE (FATORIAL)
        FIM_SE
FIM.
```

Exercício 10

Escreva um algoritmo para calcular os números harmônicos sendo que

- O usuário deve informar o numero para o calculo
- O numero informado deverá ser maior que 0
- O calculo do numero harmônico é feito da seguinte forma:
- 5H = 1 + (1/2) + (1/3)+ (1/4)+(1/5)
- NH= 1+(1/2) + (1/3) ... (1/N)

```
INICIO
        NUM: INTEIRO {PARA INFORMAR O NUMERO PARA CALCULO}
        HARMONICO: REAL {PARA CALCULO DO HARMONICO}
PROCEDA
        LEIA (NUM)
        SE NUM < = 0
                MOSTRE (“NUMERO INVALIDO")
        SENÃO
                HARMONICO : = 0
                ENQUANTO NUM > = 1
                        HARMONICO : = HARMONICO + (1 / NUM)
                        NUM : = NUM - 1
                FIM_ENQUANTO
                MOSTRE (HARMONICO)
        FIM_SE
FIM.
```

Exercício 11

Escreva um algoritmo capaz de calcular a seqüência de fibonacci sendo que:

- O primeiro numero é sempre 0
- O segundo numero é sempre 1
- Os próximos números são o resultado da soma de seus dois predecessores.

Exemplo:

0,1,1,2,3,5,8,13,21,34,55,89...

O algoritmo deve calcular e mostrar a seqüência de fibonacci ate que o ultimo numero calculado seja maior ou igual a um limite que será informado pelo usuário.

```
INICIO
        NUM:INTEIRO
        FIBONACCI:INTERIO
        ULTIMO : INTEIRO
        PENULTIMO:INTEIRO
        CONT:INTEIRO
PROCEDA
        LEIA (NUM)
        SE NUM < = 0
                MOSTRE ("NÃO EXISTE SEQUENCIA DE FIBONACCI")
        SENÃO
                PENULTIMO : = 0
                ULTIMO: =1
                MOSTRE (PENULTIMO)
                MOSTRE (ULTIMO)
                FIBONACCI : = ULTIMO + PENULTIMO
                ENQUANTO FIBONACCI < NUMERO
                        MOSTRE (FIBONACCI)
                        PENULTIMO : = ULTIMO
                        ULTIMO : = FIBONACCI
                        FIBONACCI : = ULTIMO + PENULTIMO
                FIM_ENQUANTO
                MOSTRE (FIBONACCI)
        FIM_SE
FIM.
```

Exercício 12

Faça um algoritmo capaz de determinar o maior número de uma seqüência de 10 números que deverão ser digitados pelo usuário , e apresente o resultado ao final do processamento.

Exercício 13

Faça um algoritmo capaz de calcular a media das notas de dez alunos que deverão ser informadas pelo usuário, e após o calculo informe o número de alunos que ficou com a nota acima da media calculada.

```
INICIO
        NOTAS : MATRIZ [ 10 ] DE REAL
        SOMA: REAL
        MEDIA: REAL
        CONT:INTEIRO
        ACIMA:INTEIRO
PROCEDA MEDIA_10
        MOSTRE (“DIGITE AS 10 MEDIAS”)
        CONT: =1
        SOMA : = 0
        ENQUANTO CONT < = 10
                LEIA (NOTAS [ CONT ] )
                SOMA: = SOMA + NOTAS[CONT]
        FIM_ENQUANTO
        MEDIA : = (SOMA / 10)
        ACIMA:=0
        CONT : =1
        ENQUANTO CONT < = 10
                SE NOTAS [ CONT ] > MEDIA
                        ACIMA : = ACIMA +1
                FIM_SE
        FIM_ENQUANTO
        MOSTRE (“ A MEDIA DAS NOTAS FOI=>”)
        MOSTRE (MEDIA)
        MOSTRE( “ QUANTIDADE DE NOTAS ACIMA DA MEDIA”)
        MOSTRE (ACIMA)
FIM.
```

Exercício 14

Uma empresa de autopeças classifica os seus funcionários da seguinte forma:

- Funcionário Classe A: É aquele que produziu até 100 pecas por mês
- Funcionário classe B: é Aquele que produziu de 101 a 200 Pecas por mês
- Funcionário classe C: é aquele que produziu mais de 201 pecas em um mês

Sendo ainda que o salário dos funcionários é calculado da seguinte forma:

- Classe A : Um salário mínimo por mês
- Classe B : Um salário mínimo mais 10 centavos por peca produzida acima de 101 pecas
- Classe C : um salário mínimo mais 15 centavos por peca produzida acima de 101 pecas e 20 centavos por peca produzida acima de 201 pecas.

Faca um algoritmo que permita a digitação de 10 nomes de funcionários, e as suas respectivas quantidades de pecas produzidas no mês, e que após o ultimo funcionário tenha sido digitado, o algoritmo deverá informar a quantidade de funcionários de cada classe, a produção total de todos os funcionários, e o total de salário pago pela empresa para todos os funcionários.

Matrizes

Uma matriz é um tipo de dado homogêneo, que suporta a manipulação de qualquer tipo de dado primitivo, desde que seja apenas um tipo. A matriz é uma estrutura de dados dividida em linhas e colunas. Desta forma nós podemos armazenar dentro dela diversos valores ,sendo que para podermos especificar um destes valores devemos identifica-lo através do numero da linha e da coluna aonde o elemento de encontra.
Como se pode perceber, dependendo do numero de colunas e linha de uma matriz, ele pode conter uma infinidade de informações, dai a necessidade de se especificar a linha e a coluna em que se encontram os elementos para que ele possa ser identificado.
Exemplo:

	1	2	3
1	**C**	**F**	**E**
2	**D**	**A**	**B**
3	**H**	**G**	**I**
4	**L**	**M**	**J**

Desta forma, cada elemento armazenado na matriz poderá ser acessado através do cruzamento de sua respectiva linha com a sua coluna.
Exemplo :
Na matriz acima nós temos os valores **A,B,C,D,E,F,G,H,I,J,L,M**, todos eles armazenados na mesma estrutura. Para que possamos acessar um item especifico qualquer nós devemos informar o numero da linha e coluna.
Para acessar uma letra especifica deveremos faze-lo da seguinte forma:

MATRIZ [2 , 2] => A
MATRIZ [1 , 3] => H
MATRIZ [3 , 1] => E

Note que eu sempre devo especificar primeiramente a **COLUNA**, e depois **LINHA**.

Algoritmos de Manipulação de Matrizes.

Exercício 15:

Dada uma matriz 4 x 4 de inteiros faça um algoritmo que permita o preenchimento de toda a matriz com os valores digitados pelo usuário.

Exercício 16:

Dada a matriz 4x4 preenchida no exercício anterior , faça um algoritmo capaz de fazer a soma de todos os valores de cada coluna e apresente o resultado na tela.

Exercício 17:

Escreva um algoritmo capaz de calcular a soma dos elementos que se encontram abaixo da diagonal principal de uma matriz de 10x10 do tipo inteiro, que deverá ser digitada pelo usuário , e apresentar o resultado no final do processamento.

Exercício 18:

Faça um algoritmo capaz de somar todos os valores de uma mesma linha de todas as linhas de uma matriz 4X5 do tipo inteiro, que deverá ser digitadas pelo usuário , e apresentar o resultado da soma de cada linha para o usuário

Exercício 19:

Dada uma matriz de 4x4 do tipo inteiro que deverá ser preenchida pelo usuário, faca um algoritmo capaz de criar uma nova matriz com as mesmas dimensões , baseada na primeira cujos elementos deve obedecer as seguintes regras:

- Quando o número da coluna for maior que o número da linha, o elemento deverá ser multiplicado pelo valor da coluna e colocado ma mesma posição na nova matriz

- Quando o número da linha for maior que o da coluna, o elemento deverá ser multiplicado pelo número da linha e colocado na mesma posição na nova matriz.

- Quando o número da coluna for igual ao da linha, o elemento devera ser multiplicado por 10 e colocado na mesma posição na nova matriz.

Após o processamento o algoritmo deverá mostra a nova matriz elemento a elemento para o usuário.

Exercício 20:

Quadrado Mágico.

15	15	15	15	15
15	**6**	**1**	**8**	15
15	**7**	**5**	**3**	15
15	**2**	**9**	**4**	15
15	15	15	15	15

Fazer um algoritmo capaz de colocar em uma matriz quadrada (aonde a quantidade de linhas é igual a quantidade de colunas) números inteiros, começando pelo 1 até o ultimo numero inteiro necessário para preencher todas as casas da matriz (sem pular nenhum), de forma que a soma de todos os elementos de uma linha qualquer seja igual a soma de todos os elementos de uma coluna qualquer, inclusive também deverá ser igual a soma das diagonais da matriz.

REGRAS:

1. Aceitar a ordem da matriz.(deve ser ímpar)
2. O primeiro numero é colocado na primeira linha e na coluna central
3. Repetir até que o ultimo numero colocado seja maior que o quebrado de ordem da matriz:
    - Subtrair 1 da coluna
    - Subtrair 1 da linha
    - Se a linha for igual a zero então a linha passa a ser a ordem da matriz
    - Se a coluna for igual a zero então a coluna passa a ser a ordem da matriz
    - Se a posição estiver ocupada
        - A linha passa a ser igual a linha anterior +1
        - A coluna passa a ser a coluna anterior

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTO
        ORDEM {ORDEM DO QUADRADO MAGICO}
VARIAVEIS
        ORDEM:INTEIRO
        QUADRADO:MATRIZ [ORDEM , ORDEM] {MATRIZ P/ ARMAZENAR O}
                                                {QUADRADO MAGICO}
        LINHA:INTEIRO    {CONTADOR DE LINHA DA MATRIZ}
        COLUNA:INTEIRO {CONTADOR DE COLUNAS DA MATRIZ}
        LINHA_ANTERIOR:INTEIRO    {ARMAZENAR A LINHA ANTERIOR}
        COLUNA_ANTERIOR:INTEIRO {ARMAZENAR A COLUNAANTERIOR}
        CONT:INTEIRO
PROCEDA QUADRADO_MAGICO
        SE RESTO(ORDEM / 2) = 0
                MOSTRE ( "ORDEM INVALIDA PARA QUADRADO MAGICO")
        SENAO
                CONT : = 1
                LINHA : = 1
                COLUNA : = (ORDEM +1) / 2
                QUADRADO [ COLUNA , LINHA ] : = CONT
                ENQUANTO CONT <= ORDEM * ORDEM
                        LINHA : = LINHA - 1
                        COLUNA : = COLUNA - 1
                        CONT : = CONT + 1
                        SE LINHA = 0
                                LINHA : = ORDEM
                        FIM_SE
                        SE COLUNA : = 0
                                COLUNA : = ORDEM
                        FIM_SE
                        SE QUADRADO [ COLUNA , LINHA ]  < > NULO
                                LINHA : = LINHA ANTERIOR + 1
                                COLUNA : = COLUNA_ANTERIOR
                        FIM_SE
                        QUADRADO [ COLUNA , LINHA ] : = CONTADOR
                        LINHA_ANTERIOR : = LINHA
                        COLUNA_ANTERIOR : = COLUNA
                FIM_ENQUANTO
        FIM_SE
FIM.
```

ALGORITMOS E ESTRUTURAS DE DADOS

Vetores:

Existem também as chamadas matrizes unidimensionais, ou vetores. Neste caso a matriz será formada apenas por linhas ou colunas, sendo necessário informar apenas 1 numero para identificar um elemento dentro da matriz.

EXEMPLO:

	1
1	**A**
2	**B**
3	**C**
4	**D**

No exemplo ao lado a matriz tem apenas uma coluna, e poderia ser declarada da seguinte forma:

LETRAS:MATRIZ[4] DE CHAR

E cada elemento poderá ser identificado da seguinte forma:

LETRAS[3]: = "C"

Algoritmos de Manipulação de Vetores.

Exercício 21:

Dada um vetor de 10 posições do tipo inteiro, faça um algoritmo que permita o preenchimento de toda o vetor com os valores digitados pelo usuário.

```
INICIO
VARIAVEIS;
        NUMEROS:MATRIZ [10 ] DE INTEIROS;
        POS: INTEIRO;{CONTADOR DEPOSIÇÃO DO VETOR}
PROCEDA
        PARA POS : = 1
        INCREMENTO = 1
        ATE POS  > 10
                LEIA ( NUMEROS [POS ] )
        FIM_PARA
FIM.
```

Exercício 22

Dado o vetor de 10 posições do tipo inteiro preenchido no exercício anterior , faça um algoritmo capaz de fazer a soma de todos os seus valores e apresente o resultado na tela.

```
INICIO
VARIAVEIS;
        NUMEROS : MATRIZ [10 ] DE INTEIROS;
        POS:INTEIRO {CONTADOR DA POSICAO NO VETOR}
        SOMA:INTEIRO;{VARIAVEL PARA ARMAZENAR A SOMA DOS VALORES}
 PROCEDA
        SOMA: = 0;
        PARA POS : = 1
        INCREMENTO = 1
        ATE POS  >   10
                SOMA : = SOMA + NUMEROS [POS ]
        FIM_PARA
        MOSTRE ( "O RESULTADO DA COLUNA => ")
        MOSTRE ( SOMA )
FIM.
```

Exercício 23

Dado o vetor do exercício anterior faca um algoritmo capaz de dobrar os valores de todas as posições do vetor.

```
INICIO
VARIAVEIS;
        NUMEROS : MATRIZ [ 4 , 4 ] DE INTEIROS { MATRIZ ORIGINAL}
        POS:INTEIRO {CONTADOR POSICAO}
PROCEDA
        PARA POS : = 1
        INCREMENTO = 1
        ATE POS > 10
                NUMEROS [ POS ] : = NUMEROS [ POS ] * 2
        FIM_PARA
FIM.
```

Exercício 24

Dado o vetor do exercício anterior faça um algoritmo para mostrar na tela somente os valores que estiverem em posições impares dentro do vetor.

```
INICIO
        NUMEROS : MATRIZ [ 10 ] DE INTEIROS {DECLARAÇÃO DA MATRIZ}
        POS:INTEIRO{CONTADOR DE POSICAO}
PROCEDA
        PARA POS : = 1
        INCREMENTO = 2
        ATE POS  > 10
                MOSTRE ( NUMEROS [ POS ] )
        FIM_PARA
FIM
```

Exercício 25

Dado Um Vetor de 80 posições do tipo char, que deverá ser preenchido pelo usuário, faça um algoritmo que, a partir de uma letra digitada pelo usuário conte quantas vezes esta letra aparece dentro do vetor e informe o resultado da contagem ao final do processamento.

Exercício 26

Dado o mesmo vetor preenchido no exercício anterior, faça um novo algoritmo capaz de identificar e contar, quantos pares de letras existem no vetor, ou seja quantas vezes aparecem duas letras repetidas uma após a outra.

Exercício 27

10) - Dado um vetor de 10 posições do tipo char faça uma algoritmo que permita que o usuário digite 10 letras, e que estas letras sejam colocadas nas respectivas posições dentro do vetor. O algoritmo no entanto não poderá permitir que seja digitada uma letra menor que a letra que foi digitadas anteriormente, ou seja a letra que será colocada na posição 2 deverá ser maior que a letra colocada na posição 1 , a letra colocada na posição três deverá ser maior que a letra digitada para a posição 2, e assim respectivamente até o final do vetor. A primeira letra poderá ser digitada livremente pelo usuário sendo que a regra acima somente se aplica as letras que forem colocadas da segunda posição em diante.

Exercício 28

Dado um vetor de 10 posições do tipo inteiro , que deverá ser preenchido pelo usuário, faça um algoritmo capaz de percorrer todo o vetor dobrando todos os numero que forem impares e dividindo pela metade todos os numero que forem pares colocando o resultado das operações nas mesmas posições dentro do vetor

Exercício 29

Calculo do Digito verificador do CPF:

O CPF é formado por 9 digitos, mas 2 digitos verificadores que ficam no final.
Estes dígitos verificadores são calculaods de acordo com os outros 9 digitos do CPF e servem para verificar a validade do CPF.
Apresentamos aqui uma forma simplificada para o calculo destes dígitos:

Somar os produtos de cada um dos dígitos do CPF, pela sua respectiva posição, e pegar o resto da divisão inteira desta soma por 11. este resultado será o digito verificador 1
Para calcular o digito verificador 2 , repetir o processo, somente alterando a multiplicação dos dígitos pela sua posição , de forma que as posições fiquem invertidas.

Faça um algoritmo capas de , uma vez informado os 9 digitos de um cpf qualquer, informar os 2 digitos verificadores deste cpf.

ALGORITMOS E ESTRUTURAS DE DADOS

Sub rotinas.

As subrotinas são divisões dos algoritmos, que são utilizadas para:

- Facilitar a manutenção dos algoritmos
- Dividir as tarefas.
- Dar maior organização aos algoritmos
- Para reaproveitar os Algoritmos.

Existem 2 tipos Basicos de Subrotinas:

Função:

È a subrotina que executa seus procedimentos, e ao final retorna um valor.

Procedimento:

È a subrotina que não faz retorno nenhum. Apenas executa os procedimentos a ela associados.

Para que possamos trabalhar com as subrotinas serão necessárias algumas alterações nos nossos algoritmos.

Argumentos

Um argumento é um tipo especial de variável utilizada em um algoritmo. Ele é especial porque dentro dele estará um valor predeterminado pelo usuário que esta utilizando o algoritmo.
Isto quer dizer que o argumento é um valor que será colocado dentro de uma variável pelo usuário do algoritmo, que será utilizado pelo algoritmo como uma variável comum.
Em tempo de processamento, isto quer dizer que os argumento serão passados como parâmetros na linha de comando da chamada do programa.
A única diferença do argumento para uma variável comum é que ele já vem com um valor armazenado desde o inicio do algoritmo, enquanto que as outras variáveis necessitam ser inicializadas antes de serem usadas.
No decorrer do algoritmo os argumentos poderão ser utilizados e alterados exatamente como as variáveis comuns, e inclusive deverão ser declarados.

Uma outra definição de Argumento seria que eles são variáveis destinadas a fazer a comunicação entre dois algoritmos.

Até agora temos trabalhamos com algoritmos isolados, mais na maioria das vezes teremos que trabalhas com vários algoritmos, cada um desempenhando uma função especifica dentro do que iremos chamar de sistema

Desta forma teremos algoritmos que irão se preocupar somente com a inclusão dos elementos em um vetor, enquanto que outro irá se preocupar apenas com a consulta dos valores já incluídos.

De alguma forma, estes algoritmos terão que trocar informações, e a melhor forma disto ser realizado é através de argumento.

Comando Retorne

O comando **RETORNE** é responsável pelo retorno de informações de um algoritmo para outro.

Da mesma forma que um algoritmo poderá se comunicar com outro através de argumentos, um algoritmo responsável por exemplo por um calculo, poderá devolver uma resposta para outro algoritmo através do comando **RETORNE**.

Estes algoritmos são chamados Funções.

Muitas vezes necessitamos repetidamente de um cálculo, que iremos utilizar em vários algoritmos. Para evitar que tenhamos que repetir estes calculos em cada algoritmo em que ele é necessário, nós o transformamos em uma subrotina, e fazemos com que ele retorne a resposta ao nosso calculo de acordo com os **ARGUMENTOS** que passamos para ele.

No inicio desta apostila, vocês foram apresentado a uma serie de funções, que se destinam por exemplo, ao calculo da raiz quadrada, logaritmos, etc.

A chamada de uma função se faz da seguinte forma:

X : = SQR(4)

No exemplo acima a variável **X** depois de executada a função terá o valor 2, pois a função **SQR** calcula a raiz quadrada do argumento que for fornecido para ela. Neste caso o argumento é o 4.

Então, no exemplo acima, o quatro será o argumento que está sendo passado para a função, e o resultado do calculo da raiz quadrada será colocado em X devido ao uso do comando **RETORNE**.

ALGORITMOS E ESTRUTURAS DE DADOS

Listas:

É uma estrutura de dados linear que permite representar um conjunto de informações de forma a preservar uma relação de ordem entre eles. Esta estrutura deve também suportar os 4 tipos de atualizações :

- Inclusão
- Exclusão
- Alteração
- Atravessamento.

Representação:

Exemplo

LISTA=(A,B,C,D,E)
A=>B=>C=>D=>E

Para representar as lista dentro de nossos algoritmos nós vamos utilizar a estrutura do tipo matriz que foi vista anteriormente.

As listas podem ser alocadas segundo duas relações:

- Relação de Contigüidade
- Relação de Encadeamento.

Relação de Contigüidade:

É a forma mais simples de se relacionar os elementos de uma lista. Nela um elemento aponta sempre para o seu sucessor seqüencial.
Desta forma a nossa lista estará sempre na sua ordem física, ou seja os elementos estarão dispostos fisicamente dentro da lista de maneira que a ordem entre os elementos seja mantida.

Dentro de uma lista nós devemos permitir 4 operações básicas:

- Inclusão
- Exclusão
- Alteração
- Atravessamento

Atravessamento

A operação de atravessamentos consiste em percorrer toda a lista, do seu começo até o seu final, visitando todos os seus elementos.
O atravessamento começa pelo primeiro elemento da lista, mostra este elemento e vai sempre mostrado os próximos elementos até chegar no ultimo(N)

```
INICIO
ARGUMENTOS
        LISTA: MATRIZ[N] DE CHAR ; {DECLARAÇÃO DA LISTA}
        N:INTEIRO {NUMERO DE ELEMENTOS DE LISTA}
VARIÁVEIS
        CONT:INTEIRO;                   {VARIAVEL DE CONTROLE DO LOOP}
PROCEDA ATRAVESSAMENTO_CONTI.
        SE  N < 1
                MOSTRE ('LISTA VAZIA!!!!')
        SENÃO
                CONT : = 0
                ENQUANTO CONT < N
                        CONT : = CONT + 1
                        MOSTRE (LISTA [CONT])
                FIM_ENQUANTO
        FIM_SE
FIM
```

Inclusão em Listas Contíguas

A inclusão é a operação que visa a colocação de um novo elemento dentro da lista de maneira que a ordem entre os elementos seja mantida.
Se a inclusão ocorre no final de lista , o seu procedimento é fácil. Porem se a inclusão acontecer no meio da lista, será necessário que todos os elementos após aquele que foi incluído sejam deslocados uma casa para baixo.

Note que para que possamos fazer uma inclusão nó devemos declarar a lista com 1 elemento a mais do que ele já possui, para reservar lugar para o novo elemento que será incluído.

Passo para Inclusão:

1. Criar um espaço em branco no final da lista
2. Aceitar a letra que será incluída (**ELEMENTO_NOVO**)
3. Achar a posição de inclusão da letra na lista (**POS**)
4. Fazer uma critica para ver se a letra já não existe na lista
5. Remanejar todos os elementos abaixo da posição de inclusão uma posição para baixo começando de baixo para cima.
6. Incluir o elemento na posição de inclusão.
7. N: = N + 1

```
INICIO
ARGUMENTOS:
        LISTA:MATRIZ [N+1] DE CHAR {A LISTA AONDE SERÁ FEITA A INCLUSÃO}
        N:INTEIRO{NUMERO DE ELEMENTOS DA LISTA}
VARIAVEIS
        POS:INTEIRO {PARA ACHAR A POSIÇÃO DE INCLUSÃO}
        ELEMENTO_NOVO {O NOVO ELEMENTO A SER INCLUIDO}
        CONT :INTEIRO {VARIAVEL DE CONTROLE DO LOOP}
PROCEDA INCLUSAO_CONTI
        MOSTRE("DIGITE A LETRA PARA INCLUSÃO")
        LEIA (ELEMENTO_NOVO)
        POS: = 1
        ENQUANTO POS < = N E LISTA[POS]< ELEMENTO_NOVO
                POS: = POS+1
        FIM_ENQUANTO
        SE LISTA[POS] = ELEMENTO_NOVO
                MOSTRE ("LETRA JÁ EXISTE NA LISTA!!!!")
        SENÃO
                CONT: = N
                ENQUANTO CONT > = POS
                        LISTA [ CONT+1 ] : = LISTA [ CONT ]
                        CONT : = CONT - 1
                FIM_ENQUANTO
                LISTA [ POS ] : = ELEMENTO_NOVO
                N : = N + 1
        FIM-SE
FIM.
```

Alteração em Listas Contíguas

A alteração é o processo aonde se altera o conteúdo de um elemento que já exista na lista , para um novo valor especificado pelo usuário.

Passos par Alteração:

1. Aceitar a Letra que vai ser alterada (**ELEMENTO**)
2. Aceitar a nova Letra (**ELEMENTO_NOVO**)
3. Achar a posição de alteração(**POS**)
4. Fazer uma critica para ver se a letra existe na lista
5. Fazer uma critica para ver se a alteração não vai quebrar a ordem da lista
6. Fazer a alteração.

Na hora de fazer uma alteração deve-se tomar cuidado para que o novo valor do elemento não altere a ordem da lista.
Para se certificar que o novo valor do elemento da lista , esta compatível com a ordem da lista ,ou seja se a alteração deste valor não vai comprometer a ordem da lista devemos fazer alguns testes.

Para se fazer estes testes existem 4 situações:

Situação 1

	1
1	**NOVO**

Situação 2

	1
1	**NOVO**
2	**C**
3	**D**

Situação 3

	1
1	**A**
2	**B**
3	**NOVO**

Situação 4

	1
1	**A**
2	**NOVO**
3	**C**

- Na situação 1, o que nós temos é que a quantidade de elementos da lista é igual a 1 , neste caso qualquer valor será permitido para .
- Na situação 2 o elemento que desejamos alterar esta no começo da lista, e a lista tem mais de um elemento. Neste caso , o novo elemento deverá ser menor que o elemento na próxima posição da lista para que a alteração possa ser feita.
- Na situação 3 o elemento que se deseja alterar é o ultimo elemento da lista, e a lista tem mais de um elemento. Neste caso , o novo elemento deverá ser maior que o elemento que esta na posição anterior da lista.
- Na Situação 4 , que é o mais comum, o elemento que se deseja alterar está entre dois elementos de uma lista , e esta lista possui mais do que 2 elementos. Neste caso , o novo elemento deverá ser:
  - Maior que o elemento na posição anterior da lista , e
  - Menor que o elemento na posição posterior da lista.

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTOS
        LISTA : MATRIZ [ N ] DE CHAR   {LISTA PROPRIAMENTE DITA}
        N:INTEIRO                      {QUANTIDADE DE ELEMENTOS NA LISTA}
VARIAVEIS
        POS:INTEIRO
        ELEMENTO:CHAR
        NOVO_ELEMENTO:CHAR
        ALTERA:LOGICO
PROCEDA ALTERACAO_CONTI
        MOSTRE ("DIGITE A LETRA PARA ALTERAÇÃO")
        LEIA (ELEMENTO)
        MOSTRE ("DIGITE A NOVA LETRA")
        LEIA (NOVO_ELEMENTO)
        POS: = 1
        ENQUANTO POS < = N E LISTA [POS]<> ELEMENTO
                POS: = POS+1
        FIM_ENQUANTO
        SE POS > N
                MOSTRE ("ALTERAÇÃO INVALIDA")
        SENÃO
                ALTERA: = FALSO
                CASO
                N = 1     {SITUAÇÃO 1}
                        ALTERA: = VERDADEIRO
                POS = 1  {SITUAÇÃO 2}
                        SE NOVO_ELEMENTO < LISTA[POS + 1]
                                ALTERA: = VERDADEIRO
                        FIM_SE
                POS = N   {SITUAÇÃO 3}
                        SE NOVO_ELEMENTO > LISTA[POS – 1]
                                ALTERA: = VERDADEIRO
                        FIM_SE
                SENÃO  {SITUAÇÃO 4}
                        SE NOVO_ELEMENTO > LISTA[POS –1] E
                           NOVO_ELEMENTO < LISTA[POS + 1
                                ALTERA:=VERDADEIRO
                        FIM_SE
                FIM CASO
                SE ALTERA=VERDADEIRO
                        LISTA  [POS] : = NOVO_ELEMENTO
                SENÃO
                        MOSTRE ("ALTERAÇÃO INVALIDA")
                FIM_SE
        FIM_SE
FIM
```

Exclusão em Listas Contíguas.

A exclusão é o processo de eliminação de um elemento já existente na lista. Para excluir um elemento em uma lista contígua deve-se antecipar todos os elementos abaixo do elemento a ser excluído em uma posição. Isto será feito por dois motivos, primeiro para excluir fisicamente o elemento da lista , e segundo para não deixar um espaço em branco na lista.

Passos para Exclusão

1. Aceitar a Letra que vai ser excluída (**ELEMENTO**)
2. Achar a posição de exclusão(**POS**)
3. Verificar se a letra esta na lista
4. Remanejar todos os elementos abaixo da posição de exclusão (**POS**) uma posição para cima começando de cima para baixo.
5. N: = N – 1

```
INICIO
ARGUMENTOS
        LISTA : MATRIZ [ N ] DE CHAR {MATRIZ PARA ARMAZENAR A LISTA}
        N:INTEIRO                    {NUMERO DE ELEMENTOS NA LISTA}
VARIAVEIS
        POS:INTEIRO                  {POSIÇÃO DO ELEMENTO A SER EXCLUIDO}
        CONT :INTEIRO                {VARIAVEL DE CONTROLE DO LOOP}
        ELEMENTO:CHAR                {LETRA QUE VAI SER EXCLUIDA}
PROCEDA EXCLUSÃO_CONTI
        MOSTRE("DIGITE A LETRA PARA EXCLUSÃO")
        LEIA (ELEMENTO)
        POS: = 1
        ENQUANTO POS < = N E LISTA [POS]<> ELEMENTO
                POS: = POS+1
        FIM_ENQUANTO
        SE POS > N
                MOSTRE ("EXCLUSÃO INVALIDA")
        SENÃO
                CONT: = POS
                ENQUANTO CONT <  N
                        LISTA [CONT] : = LISTA [CONT+1]
                        CONT : = CONT + 1
                FIM_ENQUANTO
                LISTA [N] : = NULO
                N: = N - 1
        FIM_SE
FIM.
```

Lista Encadeadas

Uma lista linear é dita encadeada quando um elemento aponta sempre para o próximo elemento, mesmo que eles não sejam contíguos. Ou seja , em cada elemento da lista nós teremos um ponteiro, que ira apontar para o próximo elemento da lista, mesmo que este elemento não seja o próximo fisicamente na lista .
Para que possamos entender as listas encadeadas será preciso entender os conceitos de ordem física e ordem lógica de uma lista .

Ordem Física

É a ordem seqüencial na qual os elementos estão colocados dentro da lista. Ou seja é a ordem de inclusão dos elementos dentro da lista.

Ordem Lógica

É a ordem conseguida através dos ponteiros da lista. O ponteiro é um numero que esta em cada elemento da lista e que indica qual é a próxima posição física que deve ser visitada para que a ordem da lista possa ser mantida .

No caso das listas contíguas , o próximo elemento sempre era a próxima posição física da lista , mas no caso das listas encadeadas , isto não ocorre, porque o próximo elemento da lista será determinado por seu ponteiro, e não será necessariamente a próxima posição física da lista .

Como se pode perceber, no caso das lista contíguas, estudadas anteriormente, a ordem física da lista corresponde a ordem lógica, o que não acontece nas lista encadeadas.

Para que possamos trabalhar com lista encadeadas devemos ainda conhecer um outro conceito, o **descritor**.
O **descritor** é um registro que contem informações sobre a lista. No nosso caso , o descritor terá uma única informação: a posição física do inicio da lista .O descritor será passado como argumento para todos os algoritmos de listas encadeadas.

Antes ainda de começarmos a desenvolver os algoritmos existe outro ponto a ser esclarecido:
Com a criação de um ponteiro para cada elemento da lista, nós teremos que trabalhar com dois dados de tipo diferente em uma mesma estrutura. Já que o nosso elemento é do tipo CHAR , e o ponteiro é um numero **INTEIRO**. Para solucionar este problema , nós iremos lançar mão de um recurso estudado anteriormente, o **REGISTRO**.

ALGORITMOS E ESTRUTURAS DE DADOS

O registro é a única estrutura de dados que nos permite trabalhar com diferentes tipos de dados dentro de uma mesma variavel. Como nós temos utilizado a matriz para representar as nossa listas, nada mais lógico que utilizarmos um matriz formada por registros para podermos representar uma lista encadeada.
Uma matriz de registros é uma estrutura singular, aonde cada linha da minha matriz corresponde a um registro pré-definido que contem informações de tipos diferentes, permitindo desta forma, que se possa armazenar uma lista encadeada dentro da estrutura de matrizes

Desta forma, para declarar uma lista encadeada vamos fazer da seguinte forma:

Primeiro criamos o Registro para armazenar os dados da lista:

```
REG_LISTA:REGISTRO
        LETRA:CHAR
        PROXIMO:INTEIRO
FIM_REGISTRO
```

Depois declaramos a lista pripriamento dita:

```
LISTA:MATRIZ [ N ] DE REG_LISTA
```

Além de tudo isto , ainda existe um outro conceito a ser discutido, o valor **NULO**.
O **NULO** é um valor especial que irá retornar todas as vezes que algum algoritmo tentar acessar um elemento além do final da lista. O nulo é um valor que indica que não existem mais posições a serem pesquisadas na lista .

Atravessamento

Atravessar uma lista significa percorre-la , em ordem, de forma que cada um de seus elementos seja visitado uma única vez.
Como foi feito na Lista contígua, o atravessamento começa pelo incio da lista e vai mostrando todos os seus elementos até o final. A diferença aqui , é que o atravessamento não é feito pela ordem física , mas sim pela ordem lógica.
O atravessamento deve começar pelo **Descritor**, e ir mostrado os elementos um por um , sempre indo para o proximo, até que o proximo seja nulo.

```
INICIO
ARGUMENTOS
        LISTA : MATRIZ [ N ]  DE REG_LISTA  {DECLARAÇÃO DA LISTA }
        N :INTEIRO                          {NUMERO DE ELEMENTOS DA LISTA}
        DESCRITOR: INTEIRO                  {POSIÇÃO FISICA DO INICIO DA LISTA}
VARIAVEIS
        REG_LISTA : REGISTRO
                LETRA:CHAR                  {DECLARAÇÃO DO REGISTRO DA LISTA}
                PROXIMO : INTEIRO
        FIM_REGISTRO
        CONT:INTEIRO                        {VARIAVEL DE CONTROLE DO LOOP}
PROCEDA ATRAVESSAMENTO_ENCA
        SE DESCRITOR = NULO
                MOSTRE (“LISTA VAZIA!!!!!”)
        SENÃO
                CONT: = DESCRITOR             {INICIO DA LISTA}
                ENQUANTO   CONT <> NULO
                        MOSTRE (LISTA [CONT].LETRA)
                        CONT : =  LISTA [CONT].PROXIMO
                FIM_ENQUANTO
        FIM-SE
FIM.
```

Busca do Espaço Vazio.

Nas listas encadeadas poderá ocorrer uma situação que não era possível nas lista contíguas:
Na lista encadeada o algoritmo de exclusão poderá deixar um elemento vazio no meio da lista, e neste caso , nós poderíamos aproveitar este espaço para fazer novas inclusões na lista , evitando assim que a lista fique cheia de espaços em branco no transcorrer de diversas atualizações.
Para tanto, será necessário a criaão de um algoritmo capaz de identificar e reutilizar estes espaços em branco. Nós vamos chama-lo de **ELEMENTO_VAZIO**.
Este algoritmo deverá percorrer a lista pela ordem física, e procurar pelo primeiro espaço vazio que encontrar. Caso o algoritmo não localize nenhum espaço vazio, ele deverá "**CRIAR**" este espaço, simplesmente aumentando o numero de elementos da lista(N).

```
INICIO
ARGUMENTOS
        LISTA : MATRIZ [N]  DE REG_LISTA      {DECLARAÇÃO DA LISTA}
        N:INTEIRO                             {NUMERO DE ELEMENTOS DA LISTA}
VARIAVEIS
        REG_LISTA : REGISTRO
                LETRA: CHAR                   {DECLARAÇÃO DO REGISTRO DA LISTA}
                PROXIMO : INTEIRO
        FIM_REGISTRO
        CONT:INTEIRO                          {VARIAVEL DE CONTROLE DO LOOP}
PROCEDA ELEMENTO_VAZIO
        CONT: = 1
        ENQUANTO CONT < = N  E  LISTA [CONT].LETRA < > NULO
                CONT: = CONT + 1   {ATE ACABAR A LISTA OU ACHAR UM NOME NULO}
        FIM_ENQUANTO
        SE CONT > N
                N : = N + 1
                RETORNE (N)          {NÃO ACHOU ESPAÇO VAZIO E AUMENTOU A
                LISTA}
        SENÃO
                RETORNE (CONT)    {ACHOU ESPAÇO VAZIO}
        FIM_SE
FIM.
```

Inclusão em Listas Encadeadas.

O processo de inclusão de um elemento **B** , após um elemento **A** , dentro de uma lista encadeada é feito da seguinte forma :

- Procura-se um elemento vazio
- Coloca-se ali a informação do elemento **B**
- O próximo de **B** passa a ser o próximo de **A**
- O próximo de **A** passa a ser a posição física de **B**

Note que no caso da lista estiver vazia (DESCRITOR = NULO), ou se o elemento for incluído no inicio da lista (A = NULO) , então o descritor deverá ser ajustado para o valor de B , que passará a ser o inicio da lista, o e próximo de B passará a ser o descritor.

Exemplo:

No exemplo abaixo , o descritor aponta para o elemento numero 4, logo o inicio da lista é no numero 4.

O algoritmo do elemento vazio localizou um espaço vazio na posição 2, Portanto **POS = 2**.

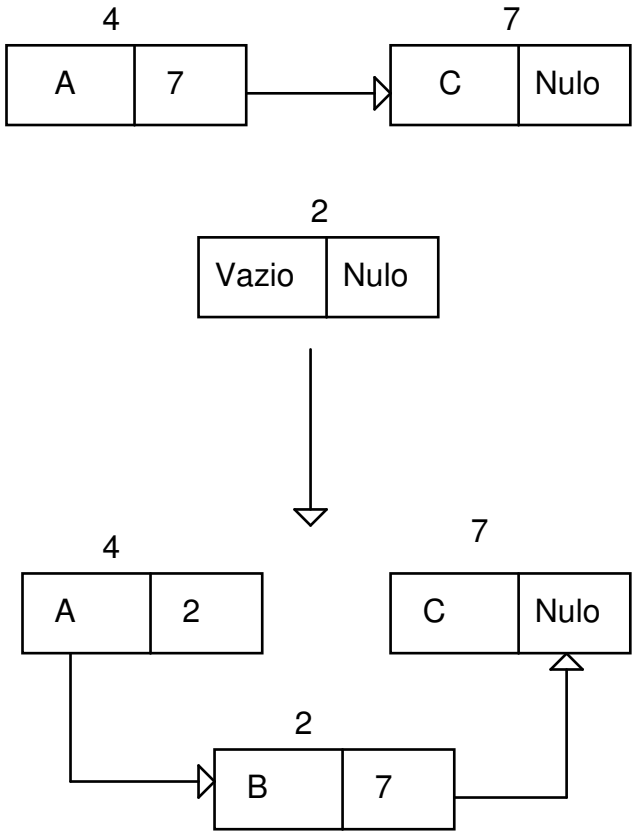
Atualmente a lista tem somente dois elementos:

- **A** na posição **4** e ,
- **C** na posição **7**.

Como o elemento a ser inclido é o B então **ANT = 4**

Para incluir o **B** no espaço vazio **2(POS)**, devemos :

- Colocar o **C** no espaço vazio **2(POS)**
- Alterar o próximo de **2(POS)** para **7(O próximo de ANT)**
- Alterar o próximo de **4(ANT)** para **2(POS)**
- O registro **7** se mantém inalterado.



ALGORITMOS E ESTRUTURAS DE DADOS

Passos para inclusão:

1. Aceitar a letra que vai ser incluida
2. Achar a posição de inclusão(**POS**)
3. Achar a posição anterior a posição de inclusão(**ANT**)
4. Verificar se a letra já não existe na lista
5. Colocar a letra na posição de inclusão(**POS**)
6. O proxino de **POS** passa a ser o próximo de **ANT**
7. O proxino de **ANT** para a se **POS**

IMPORTANTE:
O PRÓXIMO DE POS PASSA A SER O PRÓXIMO DE ANT
O PRÓXIMO DE ANT PASSA A SER A POS

Casos Especiais:

Como foi falado anteriormente existem casos especiais. Estes casos especiais acontencem quando a operação de inclusão envolve o Descritor. Isto acontece quando vamos incluir o elmento que vai ser o primeiro elemento da lista.
Neste caso então , não será possível emcontrar o elemento anterior, como esta previsto no passo 3 para insclusão(**ANT = NULO**) neste caso então devemos aplicar os seguintes passos especiais:

6. O próximo de **POS** passa ser o **DESCRITOR**
7. O **DESCRITOR** passa a ser **POS**

Note que nos casos especiais, como o valor de **ANT** não pode ser encontrado então nós substituimos **ANT** por **DESCRITOR**, que na pratia seria o **ANT** do primeiro elemento da lista.

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTOS
        LISTA : MATRIZ [ N ]  DE REG_LISTA     {DECLARAÇÃO DA LISTA }
        N:INTEIRO                              { O NUMERO DE ELEMENTOS DA LISTA}
        DESCRITOR: INTEIRO                     { O INICIO DA LISTA}
VARIAVEIS
        REG_LISTA : REGISTRO
                LETRA:CHAR                     {DECLARAÇÃO DO REGISTRO DA LISTA
                }
                PRÓXIMO : INTEIRO
        FIM_REGISTRO
        POS:INTEIRO                    {POSIÇÃO DE INCLUSÃO DO NOVO ELEMENTO
        }
        ANT:INTEIRO                    {POSIÇÃO ANTERIOR DO NOVO ELEMENTO }
        ELEMENTO_NOVO: CHAR            {A NOVA LETRA QUE VAI SER INCLUIDA}
        CONT: INTEIRO                  {VARIAVEL DE CONTROLE DE LOOP}
PROCEDA INCLUSAO_ENCA
        MOSTRE ("DIGITE A LETRA PARA INCLUSÃO")
        LEIA (NOVO_ELEMENTO)
        POS: = ELEMENTO_VAZIO (LISTA , N)
        ANT: = NULO
        CONT: = DESCRITOR
        ENQUANTO CONT < > NULO E LISTA [CONT]. LETRA < ELEMENTO_NOVO
                ANT : = CONT
                CONT: = LISTA [CONT].PROXIMO
        FIM_ENQUANTO
        SE LISTA[CONT]. LETRA = NOVO_ELEMENTO
                MOSTRE ("INCLUSÃO INVALIDA!!!!")
        SENÃO
                LISTA [POS] . LETRA : = ELEMENTO_NOVO
                SE ANT = NULO                  {CASO ESPECIAL}
                        LISTA [POS] . PROXIMO : = DESCRITOR
                        DESCRITOR : = POS
                SENAO                          {CASO NORMAL}
                        LISTA [POS]. PROXIMO : = LISTA [ANT]. PROXIMO
                        LISTA [ANT]. PROXIMO : =  POS
                FIM-SE
        FIM_SE
FIM
```

Alteração em Lista Encadeadas.

A alteração em listas encadeadas é muito parecida com a alteração em listas contíguas.
Da mesma forma que na lista contígua será necessário tomar o cuidado para que o novo valor não altere a ordem da lista .
Em essência, o algoritmo será o mesmo, apenas com algumas diferenças.
A primeiro é que na hora de se fazer as criticas, o proximo elemento deverá ser obtido com a utilização dos ponteiros. A segnda é que o elemento anterior, deverá ser conseguindo atravez de uma pesquisa sequancial pela lista, uma vez que não há outra forma de conseguinr esta informação.

Passos para alteração:

1. Aceitar a letra que vai ser alterada(**ELEMENTO**)
2. Aceitar a nova letra (**NOVO_ELEMENTO**)
3. Achar a posição de alteração(**POS**)
4. Achar a posição anterior do elemento que vai ser alterado(**ANT**)
5. Verificar se a letra a ser alterada existe na lista
6. Fazer a critica para ver se a alteração não vai quebrar a ordem da lista
7. Alterar o elemento.

Vale lembrar que no passo 6 a critica deverá ser feita de acordo com as situações vistas no algoritno de alteração em listas contíguas.

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTOS
        LISTA : MATRIZ [ N ]  DE REG_LISTA     {DECLARAÇÃO DA LISTA }
        N:INTEIRO                              { O NUMERO DE ELEMENTOS DA LISTA}
        DESCRITOR: INTEIRO                     { O INICIO DA LISTA}
VARIAVEIS
        REG_LISTA : REGISTRO
                LETRA:CHAR                     {DECLARAÇÃO DO REGISTRO DA LISTA
                }
                PRÓXIMO : INTEIRO
        FIM_REGISTRO
        POS,ANT,CONT:INTEIRO
        ELEMENTO, ELEMENTO_NOVO:CHAR
        ALTERA:LOGICO                          {FLAG DE ALTERAÇÃO}
PROCEDA ALTERACAO_ENCA
        MOSTRE ("DIGITE A LETRA PARA ALTERAÇÃO")
        LEIA (ELEMENTO)
        MOSTRE ("DIGITE A NOVA LETRA")
        LEIA (NOVO_ELEMENTO)
        ANT: = NULO
        POS: = DESCRITOR
        ENQUANTO POS <> NULO E LISTA [POS].LETRA<> ELEMENTO
                ANT: =  POS
                POS: = LISTA[POS] . PROXIMO
        FIM_ENQUANTO
        SE POS = NULO
                MOSTRE ("ALTERAÇÃO INVALIDA")
        SENÃO
                ALTERA: = FALSO
                CASO
                ANT = NULO E LISTA[POS]. PROXIMO = NULO     {SITUAÇÃO 1}
                        ALTERA: = VERDADEIRO
                ANT = NULO  {SITUAÇÃO 2}
                        SE NOVO_ELEMENTO < LISTA[LISTA[POS].PROXIMO].LETRA
                                ALTERA: = VERDADEIRO
                        FIM_SE
                LISTA[POS].PROXIMO = NULO   {SITUAÇÃO 3}
                        SE NOVO_ELEMENTO > LISTA[ANT].LETRA
                                ALTERA: = VERDADEIRO
                        FIM_SE
                SENÃO  {SITUAÇÃO 4}
                        SE NOVO_ELEMENTO > LISTA[ANT].LETRA E
                           NOVO_ELEMENTO < LISTA[LISTA[POS].PROXIMO].LETRA
                                ALTERA:=VERDADEIRO
                        FIM_SE
                FIM CASO
                SE ALTERA=VERDADEIRO
                        LISTA  [POS].LETRA : = NOVO_ELEMENTO
                SENÃO
                        MOSTRE ("ALTERAÇÃO INVALIDA")
                FIM_SE
```

```
        FIM_SE
FIM.
```

Exclusão em Listas Encadeadas.

Neste caso , a maior dificuldade reside em determinar qual o elemento que tem como próximo , o elemento que será excluído.
Essa busca tem que ser feita por varredura seqüencial , e uma vez encontrado o elemento, a exclusão é feita da seguinte forma:
O próximo do elemento anterior ao que será excluído , passa a ser o próximo do elemento que será excluído.
Depois de ajustar os ponteiros devemos mover o valor nulo para o ponteiro e para o elemento na posição em que ocorreu a exclusão.
Novamente aqui podem aparece os casos especias, que acontecem quando for solicitada a exclusão do primeiro elemento da lista(**DESCRITOR**), neste caso, o priximo do elemento excluído passa a ser o novo descritor.

Passos para Exclusão:

1. Aceitar a letra que vai ser excluída
2. Achar a posição de Exclusão(**POS**)
3. Achar a posição alteriror da letra que vai ser excluída(**ANT**)
4. Verificar se a letra a ser excluída existe na lista
5. O próximo de **ANT** passar a ser o Proxino de **POS**
6. Colocar **NULO** na posição que foi excluída.

IMPORTANTE :
O PRÓXIMO DE ANT PASSA A SER O PRÓXIMO DE POS.

Casos Especiais:

Caso não se possa localizar a letra anterior (**ANT=NULO**) isto quer dizer que o elemento que esta sendo excluído é o descritor da lista, neste caso não poderemos aplicar o passo 5 uma vez que o valor de **ANT** não existe. Neste caso devemos aplicar um passo 5 especial:

5. O **DESCRITOR** passa a ser o próximo de **POS**.

Note que este passo especial funcionar até mesmo se o elemento for o único da lista, pois neste caso o próximo é igual a NULO , e com a exclusão deste elemento o descritor passa ser ser NULO, o que indica uma lista vazia.

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTOS
        LISTA : MATRIZ [ N ]  DE REG_LISTA {DECLARAÇÃO DA LISTA }
        N : INTEIRO                          {QUANTIDADE DE ELEMENTOS DA LISTA}
        DESCRITOR: INTEIRO                   { POSIÇÃO DE INICIO DA LISTA}
VARIAVEIS
        REG_LISTA : REGISTRO
                LETRA:CHAR                   {DECLARAÇÃO DO REGISTRO DA LISTA }
                PRÓXIMO : INTEIRO
        FIM_REGISTRO
        POS:INTEIRO
        ANT:INTEIRO
        ELEMENTO: CHAR
PROCEDA EXCLUSÃO_ENCA
        MOSTRE ( " DIGITE A LETRA PARA EXCLUSÃO")
        LEIA ( ELEMENTO)
        ANT: =NULO
        POS: =DESCRITOR
        ENQUANTO POS < >  NULO E LISTA[POS].LETRA < > ELEMENTO
                ANT: = POS
                POS : = LISTA[ POS ]. PROXIMO
        FIM_ENQUANTO
        SE POS = NULO
                MOSTRE ( " EXCLUSÃO INVALIDA")
        SENÃO
                SE ANT = NULO
                        DESCRITOR : = LISTA [POS] . PROXIMO
                SENAO
                        LISTA[ ANT ].PROXIMO : = LISTA[ POS ].PROXIMO
                FIM_SE
                LISTA[ POS ].LETRA : = NULO
                LISTA[ POS ].PROXIMO : = NULO
        FIM_SE
FIM.
```

Lista Duplamente Encadeadas.

A lista duplamente encadeada tem como característica o fato de que cada elemento possui dois ponteiros. O primeiro indica o sucessor do elemento , e o segundo o seu antecessor.

No caso das listas duplamente encadeadas, o descritor terá duas informações:

- O primeiro elemento da lista
- O ultimo elemento da lista.

Para que possamos representar ambas esta informções o registro passara a se declado como um registro, da seguinte forma:

```
DESCRITOR : REGISTRO
        INICIO : INTEIRO
        FINAL : INTEIRO
FIM_REGISTRO
```

Para que possamos trabalhar com lista duplamente encadeadas, nos deveremos incluir um novo campo no registro usado para montar a matriz que servirá para armazenar a lista. Este novo campo vai conter o endereço físico do elemento anterior ao elemento atual. Da mesma forma , na lista duplamente encadeada ainda vai existir o ponteiro que indica qual é a próxima posição do elemento atual.

```
REG_LISTA:REGISTRO
        ANTERIOR:INTEIRO
        LETRA:CHAR
        PROXIMO:INTEIRO
FIM_REGISTRO
```

Note que com estas informações você poderá escolher em que ordem você deseja trabalhar com a lista, ela pode ser crescente ou decrescente.

ALGORITMOS E ESTRUTURAS DE DADOS

Atravessamento em Lista Duplamente Encadeadas.

O atravessamento em listas duplamente encadeadas é muito parecido com o atravessamento em lista encadeadas, com a diferença que você poderá escolher a ordem em qual a lista será atravessada.
Para este fim será incluído um novo argumento chamado ORDEM, que deverá indicar a ordem do atravessamento.
O atravessamento em ordem crescente começa pelo DESCRITOR.INICIO, e vai sempre pelo PROX até chegar ao final da lista. Já o atravessamento em ordem descrescente começa pelo DESCRITOR.FINAL e vai sempre pelo ANT até chegar ao inicio da Lista.
Note que de qualquer forma o final ou o inicio da lista vai ser identificado por um valor NULO nos ponteiros, pois o PROX da ultima LETRA da lista será NULO da mesma forma que o ANT da primeira letra.

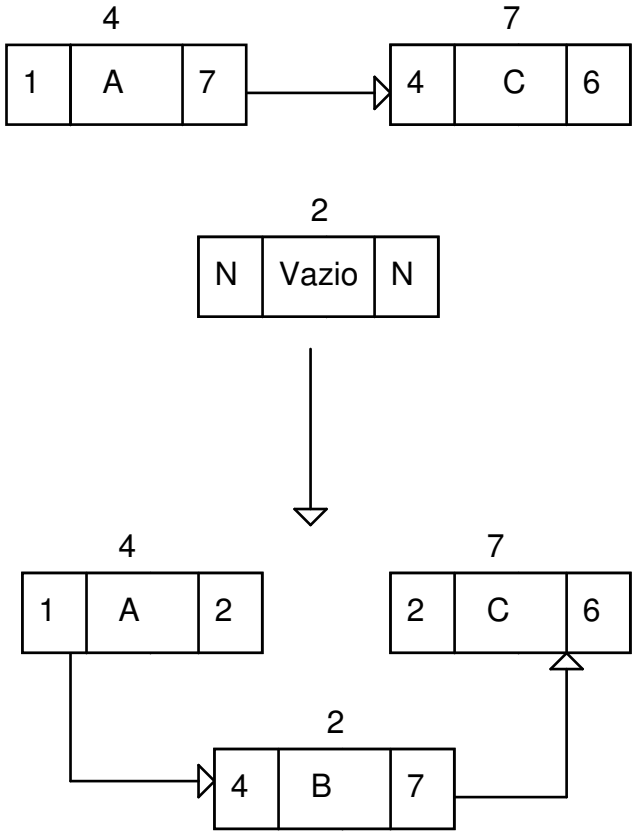
```
INICIO
ARGUMENTO
        LISTA : MATRIZ [ N ]  DE REG_LISTA    {DECLARAÇÃO DA LISTA }
        DESCRITOR : REGISTRO                  {DECLARAÇÃO DO DESCRITOR}
                INICIO: INTEIRO
                FINAL : INTEIRO
        FIM_REGISTRO
        N : INTEIRO                               {NUMERO DE ELEMENTOS DA LISTA}
        ORDEM  : CHAR                         {INDICA A ORDEM DE ATRAVESSAMENTO}
                                              {"C"=>ORDEM CRESCENTE}
                                              {"D"=>ORDEM DECRESCENTE}
VARIAVEIS
        REG_LISTA : REGISTRO
                ANTERIOR: INTEIRO
                LETRA:CHAR                  {DECLARAÇÃO DO REGISTRO DA LISTA }
                PRÓXIMO : INTEIRO
        FIM_REGISTRO
        CONT: INTEIRO                         {VARIAVEL DE CONTROLE DO LOOP}
PROCEDA ATRAVESSAMENTO_DUPLA_ENCA
        SE ORDEM = "C"
                CONT: = DESCRITOR . INICIO
        SENÃO
                CONT : = DESCRITOR . FINAL
        FIM_SE
        ENQUANTO CONT < > NULO
                MOSTRE LISTA [ CONT ].LETRA
                SE ORDEM = "C "
                        CONT :  = LISTA[ CONT ].PROXIMO
                SENAO
                        CONT : = LISTA[ CONT ].ANTERIOR
                FIM_SE
        FIM_ENQUANTO
FIM.
```

Inclusão em Listas Duplamente Encadeadas.

Para a inclusão em listas duplamente encadeadas nós vamos continuar precisando do algoritmo da busca do elemento vazio, mas não será necessário faze-lo novamente, pois poderemos usar o feito para lista encadeadas alterando apenas o registro usado para declarar a matriz.
o processo de inclusão de um elemento **A** após um elemento **B** em uma lista duplamente encadeada é feito a seguinte forma:

- Procura-se um elemento vazio**(POS)**
- Coloca-se ali a informação do elemento **B**
- O próximo de **A** passar a ser a posição física de **B**
- O próximo de **B** passa a ser o próximo de **A**
- O anterior do próximo de **A** passa a ser a posição física de **B**
- O anterior de **B** passa a ser a posição física de **A**.

Note que devemos prestar a atenção para o caso de não existir um elemento anterior ao incluído(A), ou para o caso do elemento incluído ser o primeiro da lista , ou o ultimo da lista , neste caso devemos fazer as alterações necessárias no descritor.



Passos para inclusão:

1. Aceita a letra que vai ser incluida
2. Achar a posição de inclusão(**POS**)
3. Achar a posição anterior a posição de inclusão(**ANT**)
4. Verificar se a letra já não existe na lista
5. Colocar a letra na posição de inclusão(**POS**)
6. O proxino de **POS** passa a ser o próximo de **ANT**
7. O anterior de **POS** passa a ser **ANT**
8. O próximo de **ANT** passa a ser **POS**
9. O anterior do próximo de **POS** passa a ser **POS**

IMPORTANTE:
O proxino de POS passa a ser o próximo de ANT
O anterior de POS passa a ser ANT
O próximo de ANT passa a ser POS
O anterior do próximo de POS passa a ser POS

Casos Especiais:

Da mesma forma que nas listas encadeadas nós teremos os casos especiais , só que nas listas duplamente encadeadas, nos teremos 2 casos especiais.
O primeiro acontece quando vamos incluir um elemento que por ser o menor da lista , vai passar a ser o novo **DESCRITOR.INICIO** da lista. Neste caso o valor de **ANT** será igual a **NULO**, pois não será possível achar um elemento menor que ele na lista, e deveremos ajustar as regras para alterar o valor do **DESCRITOR.INICIO**.Neste primeiro caso especial será necessário substituir o passo 6 e o passo 8 pelos seguintes passo especiais:

6. O próximo de **POS** recebe **DESCRITOR.INICIO**
8**. DESCRITOR.INICIO** recebe **POS**

O segundo caso acontece quando o elemento que vamos incluir é o maior elemento da lista, e neste caso passará a ser o **DESCRITOR.FINAL** da lista.Podemos identificar este caso especia quando o valor de Ant é igual ao valor deo DESCRITOR.FINAL, e neste caso devemos substituir o passo 9, e aplicar o seguinte passo especial:

9. **DESCRITOR.FINAL** recebe **POS**

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTOS
        LISTA : MATRIZ [ N ]  DE REG_LISTA    {DECLARAÇÃO DA LISTA }
        N:INTEIRO                                    {NUMERO DE ELEMENTOS DA LISTA}
        DESCRITOR :  REGISTRO
                INICIO:INTEIRO
                FINAL:INTEIRO
        FIM_REGISTRO
VARIAVEIS
        REG_LISTA : REGISTRO
                ANTERIOR: INTEIRO
                LETRA: CHAR                    {DECLARAÇÃO DO REGISTRO DA LISTA }
                PRÓXIMO : INTEIRO
        FIM_REGISTRO
        ELEMENTO_NOVO:CHAR
        POS, ANT,CONT: INTEIRO
PROCEDA INCLUSÃO_DUPLA_ENCA
        MOSTRE (" DIGITE A LETRA PARA INCLUSÃO")
        LEIA (ELEMENTO_NOVO)
        POS: = ELEMENTO_VAZIO (LISTA , N)
        ANT: = NULO
        CONT: = DESCRITOR.INICIO
        ENQUANTO CONT < > NULO E LISTA [CONT]. LETRA < ELEMENTO_NOVO
                ANT : = CONT
                CONT: = LISTA [CONT].PROXIMO
        FIM_ENQUANTO
        SE LISTA [CONT]. LETRA = NOVO_ELEMENTO
                MOSTRE ("INCLUSÃO INVALIDA!!!!")
        SENÃO
                LISTA [POS] . LETRA : = ELEMENTO_NOVO
                SE ANT = NULO                {PRIMEIRO CASO ESPECIAL}
                        LISTA [POS] . PROXIMO : = DESCRITOR .INICIO
                        LISTA [POS] . ANTERIOR : = ANT
                        DESCRITOR : = POS
                        LISTA [ LISTA [POS] . PROXIMO ].ANTERIOR : = POS
                SENAO
                        SE ANT = DESCRITOR . FINAL {SEGUNDO CASO ESPECIAL}
                                LISTA [ POS ] . PROXIMO : = LISTA [ ANT ] . PROX
                                LISTA [ POS ] . ANTERIOR : = ANT
                                LISTA [ ANT ] . PROXIMO : =  POS
                                DESCRITOR.FINAL : =  POS
                        SENÃO                                {CASO NORMAL}
                                LISTA [ POS ] . PROXIMO : = LISTA [ ANT ] . PROX
                                LISTA [ POS ] . ANTERIOR : = ANT
                                LISTA [ ANT ] . PROXIMO : =  POS
                                LISTA[LISTA [ POS ] .PROXIMO]. ANTERIOR : =  POS
                        FIM_SE
                FIM-SE
        FIM_SE
FIM
```

Exclusão em Listas Duplamente Encadeadas.

No caso da exclusão em listas duplamente encadeadas, não será necessário localizar o elemento anterior ao elemento que será excluído , como foi feito com as lista encadeadas. Esta informação estará disponível nos ponteiros do elemento que será excluído.
Uma vez conhecida a posição física do elemento que será excluído a exclusão em lista duplamente encadeadas funciona da seguinte forma:

- O próximo do anterior de **B** passa a ser o próximo de **B**
- O anterior do próximo de **B** passa a ser o anterior de **B**
- Move nulo para os valores de **B.**

Note que devemos levar em consideração novamente o caso do item excluído ser o ultimo ou o primeiro elemento da lista , neste caso devemos proceder as alterações necessárias no descritor da lista, que serão vistas nos casos especiais:

Passos para inclusão:

1. Aceita a letra que vai ser excluida
2. Achar a posição de Exclusão (POS)
3. Achar a posição anterior a posição de Exclusão (**ANT**)
4. Verificar se a letra existe na lista
5. O proxino de **ANT** recebe o próximo de **POS**
6. O anterior do próximo de **POS** recebe **ANT**
7. Colocar NULO na letra Anterior e Proximo

IMPORTANTE:
O proxino de ANT recebe o próximo de POS
O anterior do próximo de POS recebe ANT

Casos Especiais:

Novamente , caso a exclusão seja do primeiro ou do ultimo elemento da lista teremos os casos especiais.
Se o valor de **ANT** for igual a nulo então o elemento que esta sendo excluído é o primeiro , e deveremos utilizar o seguinte passo especial:

5. **DESCRITOR.INICIO** recebe o proximo de **POS**

Caso o valor de **POS** seja igual ao valor de **DESCRITOR.FINAL**, então o elemento que estamos exclindo é o ultimo da lista e neste caso deveremos aplicar o seguinte passo especial:

6 . O **DESCRITOR.FINAL** recebe **ANT**

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTOS
        LISTA : MATRIZ [ N ]  DE REG_LISTA    {DECLARAÇÃO DA LISTA }
        N: INTEIRO        { A QUANTIDADE DE ELEMENTOS DA LISTA}
        DESCRITOR :  REGISTRO
                INICIO :  INTEIRO
                FINAL : INTEIRO
        FIM_REGISTRO
VARIAVEIS
        REG_LISTA : REGISTRO
                ANTERIOR: INTEIRO
                ELEMENTO : CHAR          {DECLARAÇÃO DO REGISTRO DA LISTA }
                PRÓXIMO : INTEIRO
        FIM_REGISTRO
        ANT:INTEIRO
        POS:INTEIRO
        ELEMENTO: CHAR
PROCEDA EXCLUSÃO_DUPLA_ENCA
        MOSTRE ( " DIGITE A LETRA PARA EXCLUSÃO")
        LEIA ( ELEMENTO)
        ANT: =NULO
        POS: =DESCRITOR.INICIO
        ENQUANTO POS < >  NULO E LISTA[POS].LETRA < > ELEMENTO
                ANT: = POS
                POS : = LISTA[ POS ]. PROXIMO
        FIM_ENQUANTO
        SE POS = NULO
                MOSTRE ( " EXCLUSÃO INVALIDA")
        SENÃO
                SE ANT = NULO {PRIMEIRO CSO ESPECIAL }
                        DESCRITOR : = LISTA [POS] . PROXIMO
                        LISTA [ LISTA [ POS ] .PROXIMO ] . ANTERIOR : = ANT
                SENAO
                        SE POS = DESCRITOR.FINAL {SEGUNDO CASO ESPECIAL}
                                LISTA  [ ANT ] . PROXIMO : = LISTA [ POS ] . PROXIMO
                                DESCRITOR . FINAL : = ANT
                        SENÃO {CASO NORMAL}
                                LISTA  [ ANT ] . PROXIMO : = LISTA [ POS ] . PROXIMO
                                LISTA [ LISTA [ POS ] .PROXIMO ] . ANTERIOR : = ANT
                        FIM_SE
                FIM_SE
                LISTA[ POS ]. ANTERIOR : = NULO
                LISTA[ POS ].LETRA : = NULO
                LISTA[ POS ].PROXIMO : = NULO
        FIM_SE

FIM.
```

Alteração em Listas Duplamente Encadeadas

O algoritmo de alteração em listas duplamente encadeadas, é praticamente idêntico ao algoritmo de alteração em listas encadeadas, sendo necessário apenas mudar a declaração dos registros para podermos armazenar o novo ponteiro e o novo valor do descritor. Tambem não será mais necessário o processo de achar o elemento anterior ao elemento que esta sendo alterado, pois esta informação já esta armazenada nos ponteiros da lista.

Por ser um algoritmo praticamente idêntico al de listas encadeadas, não será necessário repeti-lo aqui.

Filas (Queue)

Filas são listas com acesso disciplinados do tipo **FIFO (FIRST IN FIRST OUT)**, ou seja , o primeiro a entrar na fila será o primeiro a sair.
As filas são comumente utilizadas para controle acesso a dispositivos. Normalemtne quando temos um ou poucos objetos que são necessários a vários outros objetos , teremos uma fila para controlar o acesso a este objeto. Um bom exemplo disso são as filas de impressão, para acesso a um HD, etc.
Filas são estruturas facilmente encontradas em algoritmos , e o seu processo de atualização se divide em duas partes:

- **INSERÇÃO** => Que sempre acontece no final da fila
- **REMOÇÃO** => Que sempre acontece no topo da fila.

Como a quantidade de elementos em uma fila quase sempre é pequeno (< 100) , normalmente elas são representadas através de listas, alocadas de acordo com uma relação de contigüidade.

Exemplo:

A lista representada abaixo é composta por 4 elementos, sendo que o inicio está na posição 1 e o fim na posição 4.A fila está alocada em uma lista contígua de 5 elementos.

1	**JOÃO**	INICIO
2	**MARIA**	
3	**MARCIA**	
4	**JOSE**	FINAL
5		

Inserção em Filas

O processo de inclusão se dá sempre no elemento seguinte ao ultimo elemento da fila, desde que ele não ultrapasse a dimensão máxima da fila.

Note que para as filas, a quantidade de elementos da lista se refere a quantidade máxima de elementos que poderão ser colocados na fila, ou seja , é a dimensão máxima da fila. A dimensão da fila não pode ser aumentada dimamicamente.

```
INICIO
ARGUMENTOS
        FILA : MATRIZ [ N ] DE STRING[40]   { A FILA}
        N:INTEIRO                           {NUMERO DE POSIÇÕES DA FILA}
        FINAL                       {POSIÇÃO OCUPADA PELO ULTIMO ELEMENTO DA
        FILA}
        NOME_NOVO:STRING [40] {NOME A SER INCLUIDO}
PROCEDA INSERSÃO_FILA
        SE  FINAL > = N
                MOSTRE ( " A FILA ESTA CHEIA ")
        SENÃO
                FILA [ FINAL + 1 ] : = NOME_NOVO
                FINAL : = FINAL + 1
        FIM_SE
FIM.
```

Remoção em Filas

O processo de remoção de um elemento em uma fila sempre se dá no inicio da lista, e conseqüentemente , todos os demais elementos de fila deverão ser remanejados uma posição para cima.

```
INICIO
ARGUMENTOS
        FILA : MATRIZ [ N ] DE STRING[40] {A FILA}
        N:INTEIRO          {NUMERO DE POSIÇÕES DA FILA}
        FINAL:INTEIRO
VARIAVEIS
        CONT : INTEIRO
        PRIMEIRO: STRING[40]
PROCEDA REMOÇÃO_FILA
        SE FINAL = 0
                MOSTRE ( " LISTA VAZIA ")
        SENÃO
                PRIMEIRO : = FILA [ 1 ]
                CONT: = 1
                ENQUANTO CONT < = FINAL
                        FILA [ CONT ] : = FILA [ CONT + 1 ]
                        CONT : = CONT + 1
                FIM_ENQUANTO
                FILA [ FINAL ] : = NULO
                FINAL : = FINAL - 1
                RETORNE (PRIMEIRO)
        FIM_SE
FIM
```

Pilhas (Stack)

São listas com acesso disciplinado do tipo **LIFO (LAST IN FIRST OUT)**, ou seja o ultimo a entrar será o primeiro a sair.
As pilhas normalmente são utilizadas para controlar a execusão de tarefas interdependentes, ou seja , tarefas que para serem finalizadas dependes da finalização de outras tarefas. Na informática um exemplo de utilização de pilhas é no controle de chamadas de sub rotinas.
São estruturas muito comuns em algoritmos e o seu processo de atualização se divide em duas partes:

- **INSERÇÃO =>** É a colocação de um novo elemento na pilha
- **REMOÇÃO>** É a remoção de um elemento da pilha.

Tanto a inserção como a remoção são realizados no topo da pilha.

Exemplo.

A lista ao lado representada é composta por 4 elementos, estando inserida em uma lista com 6 elementos. O topo desta lista está na posição 4.

6	
5	
4	**BRANCO**
3	**AZUL**
2	**AMARELO**
1	**VERDE**

Como a quantidade de elementos em uma fila quase sempre é pequeno (< 100) , normalmente são representadas através de listas alocadas de acordo com uma relação de contigüidade.

Inserção em Pilhas.

O processo e inserção se dá no elemento seguinte ao topo (topo +1), sempre que este não ultrapassar a dimensão máxima da lista.

```
INICIO
ARGUMENTOS
        PILHA : MATRIZ [ N ] DE STRING[40]
        N:INTEIRO        {O NUMERO DE POSIÇÕES DA PILHA}
        TOPO:INTEIRO  {POSIÇÃO DO PRIMEIRO ELEMENTO DA PILHA}
        ELEMENTO_NOVO {O NOME A SER INCLUIDO NA PILHA}
PROCEDA INSERÇÃO_PILHA
        SE TOPO > = N
                MOSTRE ( " A PILHA ESTÁ CHEIA ")
        SENÃO
                PILHA [ TOPO + 1 ] : = ELEMENTO_NOVO
                TOPO : = TOPO + 1
        FIM_SE
FIM.
```

Remoção em Pilhas.

O processo de remoção em pilha se da no topo da pilha.

```
INICIO
ARGUMENTOS
        PILHA : MATRIZ [ N ] DE STRING[40]
        N:INTEIRO        {O NUMERO DE POSIÇÕES DA PILHA}
        TOPO:INTEIRO {POSIÇÃO DO PRIMEIRO ELEMENTO DA PILHA}
VARIAVEIS
        PRIMEIRO: STRING
PROCEDA REMOÇÃO_PILHA
        SE TOPO = 0
                MOSTRE ( ' A PILHA ESTA VAZIA ")
        SENÃO
                PRIMEIRO : = PILHA [ TOPO]
                PILHA [ TOPO ]  : = NULO
                TOPO : = TOPO - 1
                RETORNE (PRIMEIRO)
        FIM_SE
FIM.
```

Recursividade:

Um Algoritmo recursivo é aquele que consiste parcialmente, ou é definido em termos de si próprio.

Tipos de Recursividade:

- Diretamente Recursivo:
  Quando um algoritmo contiver uma referencia explicita a si mesmo através de seu nome.
- Indiretamente Recursivo:
  Quando um algoritmo contiver uma chamada a um segundo algoritmo que por sua vez contem uma chamada ao primeiro

Para a criação de algoritmos recursivos devemos levar em conta os seguintes fatores:

- Os algoritmos recursivos deverão sempre ter um nome, pelo qual serão chamados
- Os algoritmos recursivos deverão sempre utilizar argumentos
- Os algoritmos recursivos sempre utilizarão o comando Retorne

O comando retorne é o principal comando dentro de um algoritmo recursivo, pois é através dele que as diversas copias dos algoritmos poderão comunicar-se entre si.

Para que um algoritmo recursivo funcione devemos estabelecer uma condição de parada, que irá disparar um comando retorne que por sua vez irá desencadear uma seqüência de retornos que irá culminar com o encerramento de todas as copias dos algoritmos que estão na pilha.

Exemplo de algoritmos recursivos:

Calculo do Fatorial

O calculo do fatorial foi um exercício que já foi feito anteriormente, de forma convencional , vamos apresentar aqui as duas formas, a convencional e a recursiva para que possamos entender melhor como a recursividade funciona.

Convencional:

```
INICIO
ARGUMENTO
        NUMERO:INTEIRO
VARIAVEIS
        FATORIAL : REAL
PROCEDA FATORIAL_CONVENCIONAL
        FATORIAL=1
        ENQUANTO NUMERO > 1
                FATORIAL : = FATORIAL * NUMERO
                NUMERO : = NUMERO + 1
        FIM_ENQUANTO
        RETORNE (FATORIAL)
FIM
```

Recursivo:

```
INICIO
ARGUMENTO
        NUMERO:INTEIRO
VARIAVEIS
        FATORIAL : REAL
PROCEDA FATORIAL_REC
        SE NUMERO = 1
                FATORIAL = 1
        SENÃO
                FATORIAL = NUMERO * FATORIAL_REC(NUMERO - 1)
        FIM_SE
        RETORNE (FATORIAL)
FIM
```

Considerações sobre recursividade:

Antes de Utilizarmos uma solução recursiva para um algoritmo devemos considerar os seguintes pontos:

- Evitar níveis de recursão muito profundos, devido a ocupação de memória.
- É preferível evitar a recursividade sempre que houver uma solução Convencional
- Nem sempre a solução recursiva é a mais rápida

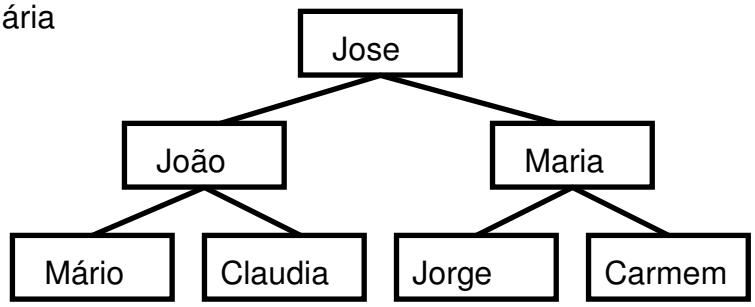
Árvores

São estruturas de dados aonde um conjunto de dados é hierarquicamente subordinado a outro.

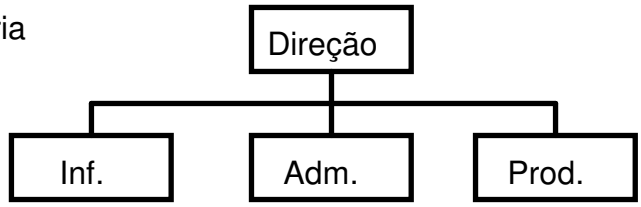
Comentários.

As árvores encontram grande aplicação no processamento de dados pois uma parcela significativa de fenômenos podem ser representados com o seu emprego.
Organogramas empresariais são exemplos de árvores, bem como os fluxogramas acadêmicos e árvores genealógicas.
A relação de hierarquia expressa através da árvore pode ser uma prioridade , um índice ou outra ascendência qualquer. Algumas estruturas de arquivo tem o seu índice organizado como sendo uma árvore.

Arvore Binária



Arvore Ternaria



Terminologia.

NÓ
É o dado a partir do qual é definida a hierarquia.

Raiz
É o nó principal, ou seja, aquele ao qual os demais nós estão subordinados.

Sub-Arvores
É aquela que se forma a partir de um determinado nó.

Nível
Numero de "**LINHAS**" que liga o nó à raiz. A raiz tem nível igual a Zero(0)

Altura
A altura de uma árvore corresponde ao seu nível máximo.

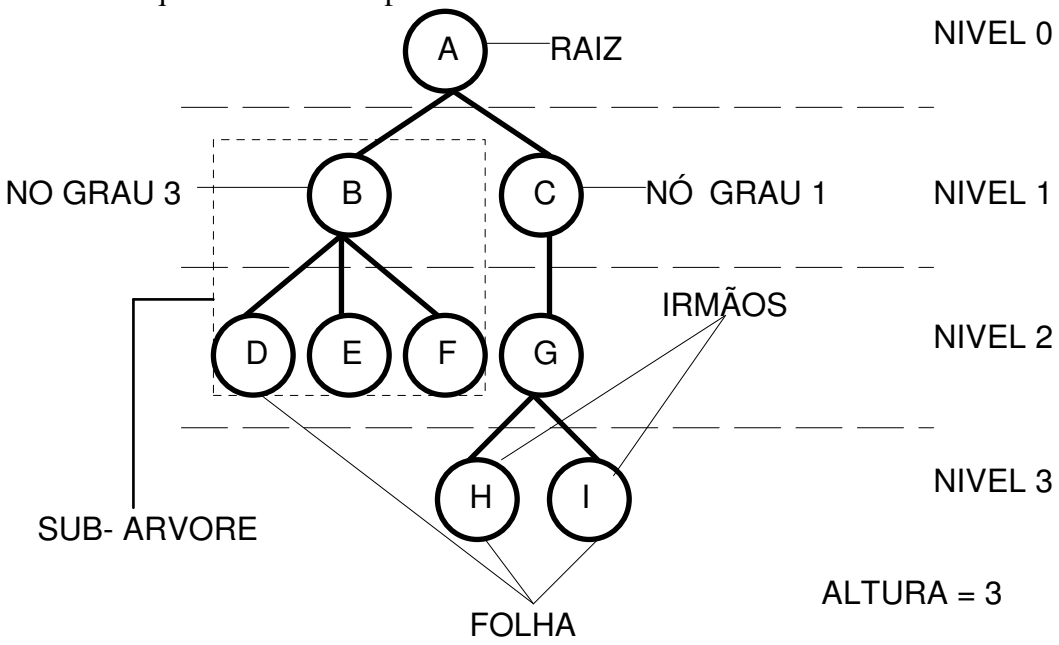
Grau
Numero de Sub - arvores que se formam a partir de um nó.

Folha
É um nó terminal , isto é , um nó com grau zero(0)

Floresta
Conjunto de árvores disjuntas.

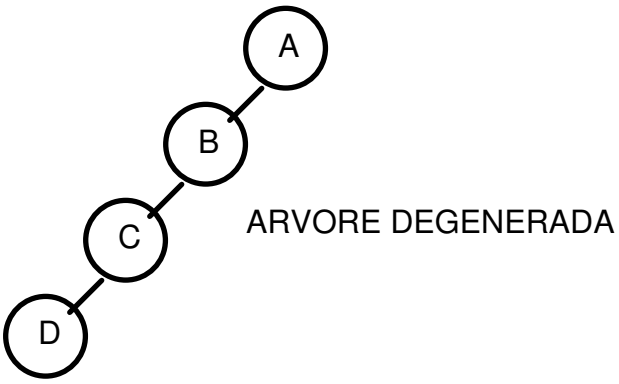
Pai
O pai de um nó é aquele ao qual o nó esta diretamente subordinado.

Irmãos
São os nós que tem o mesmo pai



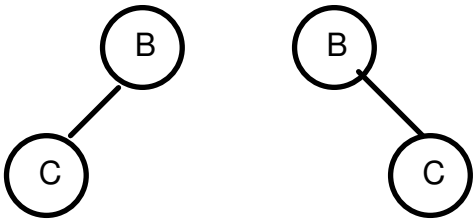
Árvore Degenerada

Na pratica se comportam como listas lineares, pois todos os seus nós estão de um mesmo lado.



Árvores binarias.

São estruturas de dados do tipo árvores aonde o grau de um nó é sempre menor ou igual a 2
Árvores binarias são o tipo de árvores mais Comumente empregado, devido a sua facilidade de alocação.
Para o caso especifico das árvores binarias , faz-se uma diferenciação entre as Sub - arvores da direita e a da esquerda. Isto é:



Se as árvores acima forem tratadas como árvores binarias, então elas seriam diferentes, caso contrario elas são idênticas ou semelhantes.

Altura de uma árvore binaria completa.

Uma árvore binaria é dita completa quando todos os nós, com exceção das folhas, tem grau igual a 2.Sendo "N" o numero total de nós da árvore, então:

ALTURA = ($\mathbf{Log_2}$ N) + 1

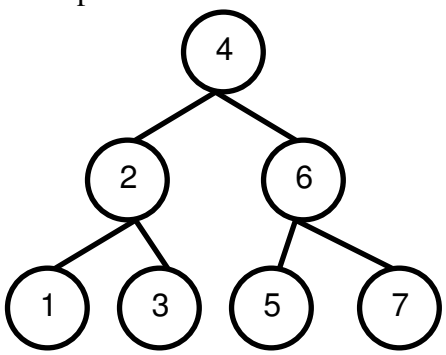
ALGORITMOS E ESTRUTURAS DE DADOS

Alocação

Via de regra a alocação de árvores se da de forma encadeada, muito semelhante as listas. Na pratica as árvores são alocadas dentro de matrizes, aonde cada registro possui 3 informações:

- O filho da esquerda do nó
- O valor do nó
- O filho da direita do nó

O descritor terá o endereço físico da raiz da árvore.
Exemplo:



ESQ	NÓ	DIR
2	4	3
4	2	5
6	6	7
-	1	-
-	3	-
-	5	-
-	7	-

DESCRITOR = 1

Árvores De Pesquisa Binária(Bst)

Ou **BINARY SEARCH TREES (BST)**, É toda árvores binária aonde as informações contidas em todos os nós respeite as seguintes regras:

- Seja maior do que todas as informações contidas em todos os nós da Sub - arvore da esquerda
- Seja menor que todas as informações contidas em todos os nós da Sub - arvores direita.
- Normalmente não se permite informações iguais em nós diferentes.

Consulta.

A consulta a uma **BST** consiste em identificar ou não a existência de um nó correspondente ao argumento de busca, para tanto procedemos da seguinte forma:

- Começamos a busca pela raiz
- Se o nó pesquisado estiver nulo, é porque o argumento não está na árvore.
- Se o argumento de busca for igual ao valor do nó , o processo de busca esta concluído.
- Se o argumento de busca for maior que o valor do nó , a busca continua pela Sub - arvore direita do nó
- Se o argumento de busca for menor do que o nó , então a busca continua pela Sub - arvore esquerda do nó.

```
INICIO
ARGUMENTOS
        ARVORE : MATRIZ [N] DE REG-ARVORE
        DESCRITOR:INTEIRO
        N:INTEIRO
        ARGUMENTO-DE-BUSCA:INTEIRO
VARIAVEIS
        REG-ARVORE: REGISTRO
                ESQUERDA:INTEIRO
                NO:INTEIRO
                DIREITA:INTEIRO
        FIM-REGISTRO
        CONT:INTEIRO
PROCEDA CONSULTA-ARVORE
        CONT : = DESCRITOR
        SE   CONT  = NULO
                MOSTRE ( "ARVORE VAZIA")
        SENÃO
                ENQUANTO CONT < > NULO
                        SE ARVORE [CONT ].NO = ARGUMENTO-DE-BUSCA
                                RETORNE CONT
                        SENÃO
                        SE ARGUMENTO-DE-BUSCA > ARVORE [CONT ].NO
                                CONT: = ARVORE [CONT ].DIREITA
                        FIM-SE
                        SE ARGUMENTO-DE-BUSCA < ARVORE [CONT ].NO
                                CONT: = ARVORE[ CONT ].ESQUERDA
                        FIM-SE
                FIM-ENQUANTO
                MOSTRE ("ARGUMENTO NÃO ENCONTRADO")
        FIM-SE
FIM.
```

Consulta Recursiva

Como nós vamos ver, as arvores binarias tem uma forte natureza recursiva, isto é , devido a sua simetria entre os nós , pode-se fazer o mesmo tratamento para diferentes nós da mesma arvore. Por este motivo podemos escrever todos os algoritmos de tratamento de arvores de forma recursiva, economizando assim linhas de código, e melhorando a performance dos algoritmos.
A consulta recursiva tem a mesma estrutura da consulta normal, percorrer a arvore até que seja encontrado o argumento de busca , ou que seja encontrado um elemento nulo, que indica que o argumento pesquisado não se encontra na arvore.

```
INICIO
ARGUMENTOS
        ARVORE: MATRIZ[N] DE REG-ARVORE
        N:INTEIRO
        DESCRITOR:INTEIRO
        ELEMENTO:INTEIRO
VARIAVEIS
        REG-ARVORE: REGISTRO
                ESQUERDA:INTEIRO
                NO:INTEIRO
                DIREITA:INTEIRO
        FIM-REGISTRO
PROCEDA CONSULTA-REC
        SE  DESCRITOR = NULO
                RETORNE "ELEMENTO NÃO ENCONTRADO"
        FIM-SE
        SE ARVORE [ DESCRITOR ].NO = ELEMENTO
                RETORNE DESCRITOR
        SENAO
                SE ARVORE [ DESCRITOR ].NO > ELEMENTO
                        CONSULTA-REC
                        (ARVORE,N,ARVORE[DESCRITOR].ESQUERDA,ELEMENTO)
                SENAO
                        CONSULTA-REC
                        (ARVORE,N,ARVORE[DESCRITOR].DIREITA,ELEMENTO)
                FIM-SE
        FIM-SE
FIM
```

Inclusão

A inclusão em uma **BST** permite anexar um novo nó mantendo todas as características da árvore. Para tanto procedemos da seguinte forma:

- Busca-se um registro vazio na matriz, atravessando a árvore pela ordem física. Nesta posição será colocado o novo elemento (pode ser usado o algoritmo de busca do espaço vazio visto anteriormente).
- Procura-se o nó que será pai do incluído (esta busca começa pela raiz)
- Caso exista elemento de igual valor na árvore o procedimento é interrompido.
- Os ponteiros do elemento incluído serão sempre nulos.
- O ponteiro de seu pai (esquerdo ou direito) deve ser alterado para o endereço físico do elemento que está sendo incluído.

Temos que tomar um cuidado especial se o elemento que vamos incluir é o primeiro da Arvore, pois neste caso teremos que atualizar o descritor

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTOS
        ARVORE: MATRIZ[N] DE REG-ARVORE
        N: INTEIRO
        DESCRITOR:INTEIRO
        ELEMENTO:INTEIRO
VARIAVEIS
        REG-ARVORE: REGISTRO
                ESQUERDA:INTEIRO
                NO:INTEIRO
                DIREITA:INTEIRO
        FIM-REGISTRO
        CONT:INTEIRO
        PAI ,FILHO , POS : INTEIRO
PROCEDA INCLUSÃO-ARVORE
        POS : = ELEMENTO_VAZIO (ARVORE,N)
        PAI : = NULO
        FILHO : = DESCRITOR
        ENQUANTO FILHO <> NULO E ARVORE [FILHO].NO < > ELEMENTO
                SE ELEMENTO > ARVORE [FILHO].NO
                        PAI : = FILHO
                        FILHO : = ARVORE[ FILHO ] .DIREITA
                SENAO
                        PAI : = FILHO
                        FILHO : =  ARVORE[ FILHO ] .ESQUERDA
                FIM-SE
        FIM-ENQUANTO
        SE ARVORE [ FILHO ].NO = ELEMENTO
                MOSTRE ( "ELEMENTO JÁ EXISTE")
        SENÃO
                ARVORE [POS].NO: = ELEMENTO
                ARVORE [POS].DIREITA : = NULO
                ARVORE [POS].ESQUERDA : = NULO
                SE PAI = NULO
                        DESCRITOR : = POS
                SENÃO
                        SE ARVORE [ PAI ].NO < ELEMENTO
                                ARVORE [ PAI ].DIREITA = POS
                        SENAO
                                ARVORE [ PAI ].ESQUERDA = POS
                        FIM SE
                FIM-SE
        FIM_SE
FIM
```

Inclusão Recursiva

O processo para inclusão recursiva tem as mesmas diretrizes da inclusão normal, ou seja , eu devo , procurar o locar de inserção do elemento , através de uma pesquisa que começa da raiz, se eu encontrar o elemento, a inclusão deve ser abortada, caso contrario, eu devo fazer a inclusão em um espaço vazio, e fazer com que o respectivo encadeamento do nó incluído com o seu pai. No algoritmo anterior, nós usamos a variável PAI para armazenar que será o pai do novo elemento, no caso do algoritmo recursivo, nós não vamos precisa desta variável, pois o valor do PAI , será armazenado naturalmente na pilha de chamadas.

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTOS
        ARVORE,N,DESCRITOR,ELEMENTO,INC
VARIAVEIS
        N,DESCRITOR,ELEMENTO : INTEIRO
        INC:LOGICO
        REG-ARVORE: REGISTRO
                ESQUERDA:INTEIRO
                NO:INTEIRO
                DIREITA:INTEIRO
        FIM-REGISTRO
        MATRIZ ARVORE[N] DE REG-ARVORE
PROCEDA INCLUSAO-REC
        SE  DESCRITOR = NULO
                INCLUIU : = VERDADEIRO
                POS : = ELEMENTO-VAZIO(ARVORE,N)
                ARVORE [ POS ].NO : = ELEMENTO
                ARVORE [ POS ].ESQUERDA : = NULO
                ARVORE [ POS ].DIREITA : = NULO
        SENAO
                SE ARVORE [ DESCRITOR ] = ELEMENTO
                        RETORNE "INCLUSAO INVALIDA")
                FIM-SE
                SE ARVORE [ DESCRITOR ].N > ELEMENTO
                        INCLUSAO-REC
                        (ARVORE,N,ARVORE[DESCRITOR].ESQUERDA,ELEMENTO,INC)
                        SE INC
                                ARVORE [ DESCRITOR ] . ESQUERDA : = POS
                                INC : = FALSO
                        FIM-SE
                SENAO
                        CONSULTA-REC
                        (ARVORE,N,ARVORE[DESCRITOR].DIREITA,ELEMENTO,INC)
                        SE INC
                                ARVORE [ DESCRITOR ].DIREITA : = DESCRITOR
                                INC: = FALSO
                        FIM-SE
                FIM-SE
        FIM-SE
        SE INC
                DESCRITOR : = POS
        FIM-SE
FIM
```

Exclusão

A exclusão permite eliminar um nó de uma árvore mantendo suas propriedades. Para tanto procedemos da seguinte forma, a exclusão será feita somente de maneira recursiva, pois a sua implementação de forma normal é muito complicada.

- Procura-se o no que vai ser excluído e o seu pai
- Se o nó que será excluído não tiver nenhum filho ele será simplesmente excluído
- Se o nó que será excluído tiver somente um filho o seu avo passará a ser o seu pai.
- Se o no a ser excluído tiver 2 filhos ele será substituído pelo nó que estiver mais a esquerda entre todos os que estão a sua direita.

Novamente devemos tomar um cuidado espcia caso se faça a exclusão da raiz da arvore, neste caso teremos que atualizar o descritor.

ALGORITMOS E ESTRUTURAS DE DADOS

```
INICIO
ARGUMENTOS
        ARVORE,N,DESCRITOR,ELEMENTO
VARIAVEIS
        REG-ARVORE: REGISTRO
                ESQUERDA:INTEIRO
                NO:INTEIRO
                DIREITA:INTEIRO
        FIM-REGISTRO
        N,DESCRITOR,ELEMENTO,FILHO,PAI,NP:INTEIRO
        MATRIZ ARVORE[N] DE REG-ARVORE
PROCEDA EXCLUSÃO-ARVORE
        FILHO : =DESCRITOR
        PAI : =DESCRITOR
        ENQUANTO ARVORE[ FILHO ].NO < > ELEMENTO
                    E ARVORE [ FILHO ].NO < > NULO
                PAI : = FILHO                          {PROCURA O NO}
                SE ELEMENTO > ARVORE  [ FILHO ].NO
                        FILHO : = ARVORE [ FILHO ].DIREITA
                SENAO
                        FILHO : = ARVORE [ FILHO ].ESQUERDA
                FIM-SE
        FIM ENQUANTO
        SE ARVORE  [ FILHO ].NO = NULO
                RETORNE "ERRO ELEMENTO INEXISTENTE"
        FIM-SE
        SE ARVORE [ FILHO ].ESQUERDA = NULO E ARVORE [ FILHO ] .DIR = NULO
                SE ARVORE [ FILHO ].NO > ARVORE [ PAI ].NO
                        ARVORE [ PAI ].DIREITA : = NULO
                SENAO                                  {CASO O NO SEJA FOLHA}
                        ARVORE [ PAI ].ESQUERDA : = NULO
                FIM-SE
        SENAO
                SE ARVORE [ FILHO ].ESQUERDA < > NULO
                  E ARVORE [ FILHO ].DIREITA < > NULO
                        NP : = ARVORE [ FILHO ] .DIREITA
                        ENQUANTO  ARVORE [ NP ].ESQUERDA < > NULO
                                NP : = ARVORE [ NP ].ESQUERDA
                        FIM_ENQUANTO
                        EXCLUSAO-ARVORE (ARVORE,N,DESCRITOR,ARVORE [NP].NO)
                        ARVORE [ FILHO ].NO : = ARVORE [NP].NO
                SENAO
                        SE ARVORE [ FILHO ].NO > ARVORE [ PAI ].NO
                                ARVORE [ PAI ].DIREITA : = ARVORE [ FILHO ].DIREITA
                        SENAO
                                ARVORE [ PAI ].ESQUERDA : = ARVORE [FILHO].ESQUERDA
                        FIM-SE
                FIM-SE
        FIM-SE
FIM
```

Atravessamento

É a operação de percorrer os nós de uma árvore de forma sistemática, visitando cada nó uma única vez. Existem 3 formas de atravessamento:

- **Atravessamento Pre-Fixado**
- **Atravessamento Pós-Fixado**
- **Atravessamento em ordem.**

Os três algoritmos de atravessamento serão feitos de maneira recursiva, devido ao fato dos algoritmos convencionais serem muito complexos

Atravessamento em ordem.

Requer que se desça pelo no raiz , sempre pela esquerda até que não seja mais possível. Então mostra-se o nó. Ai vai-se um no para direita e recomeça o processo.

```
INICIO
ARGUMENTOS
        ARVORE : MATRIZ [ N ] DE REG-ARVORE
        N:INTEIRO
        DESCRITOR:INTEIRO
VARIAVEIS
        REG-ARVORE: REGISTRO
                ESQUERDA:INTEIRO
                NO:INTEIRO
                DIREITA:INTEIRO
        FIM-REGISTRO
PROCEDA ATRAV-ARVORE
        SE DESCRITOR < > NULO
                ATRAV-ARVORE(ARVORE,N,ARVORE [ DESCRITOR ].ESQUERDA)
                MOSTRE (ARVORE [ DESCRITOR ].NO)
                ATRAV-ARVORE(ARVORE,N,ARVORE [ DESCRITOR ].DIREITA)
        FIM-SE
FIM.
```

ALGORITMOS E ESTRUTURAS DE DADOS

Atravessamento Pre-Fixado

Para se fazer o atravessamento pré - fixado, devemos primeiramente mostrar o nó , e então atravessar toda a sua Sub - arvore de esquerda, e depois toda a sua Sub - arvore da direita. Neste caso a raiz será o primeiro nó a ser mostrado , e as informações não serão apresentadas em ordem crescente.

```
INICIO
ARGUMENTOS
        ARVORE: MATRIZ [ N ] DE REG-ARVORE
        N: INTEIRO
        DESCRITOR: INTEIRO
VARIAVEIS
        REG-ARVORE: REGISTRO
                ESQUERDA:INTEIRO
                NO:INTEIRO
                DIREITA:INTEIRO
        FIM-REGISTRO
PROCEDA ATRAV-ARVORE-PRE
        SE DESCRITOR < > NULO
                MOSTRE (ARVORE [ DESCRITOR ].NO)
                ATRAV-ARVORE-PRE(ARVORE,N,ARVORE [ DESCRITOR ].ESQUERDA)
                ATRAV-ARVORE-PRE(ARVORE,N,ARVORE [ DESCRITOR ].DIREITA)
        FIM-SE
FIM.
```

ALGORITMOS E ESTRUTURAS DE DADOS

Atravessamento Pos-Fixado

Para se fazer o atravessamento pós - fixado nós devemos primeiro atravessar toda a Sub - arvore da esquerda de um nó, depois toda a Sub - arvore da direita , e só então mostrar o nó.
Neste caso , a raiz será o ultimo nó a ser mostrado, e as informações não estarão em ordem crescente.

```
INICIO
ARGUMENTOS
        ARVORE : MATRIZ [ N ] DE REG-ARVORE
        N:INTEIRO
        DESCRITOR : INTEIRO
VARIAVEIS
        REG-ARVORE: REGISTRO
                ESQUERDA:INTEIRO
                NO:INTEIRO
                DIREITA:INTEIRO
        FIM-REGISTRO
PROCEDA ATRAV-ARVORE-POS
        SE DESCRITOR < > NULO
                ATRAV-ARVORE-POS(ARVORE,N,ARVORE [ DESCRITOR ].ESQUERDA)
                ATRAV-ARVORE-POS(ARVORE,N,ARVORE [ DESCRITOR ].DIREITA)
                MOSTRE (ARVORE [ DESCRITOR ].NO)
        FIM-SE
FIM.
```

ALGORITMOS E ESTRUTURAS DE DADOS

Pesquisa

É o processo ordenado de busca a um conjunto de informações que atendam a uma condição lógico relacional.

Exemplo:

- Lista Telefônica.

Dependendo dos recursos disponíveis e da maneira com as informações estão dispostas, poderemos optar por diferentes métodos de pesquisa:

- **Pesquisa Seqüencial**
- **Pesquisa Binaria**
- **Pesquisa Por Endereçamento Direto**
- **Pesquisa por Endereçamento Indireto.**

ALGORITMOS E ESTRUTURAS DE DADOS

Pesquisa seqüencial:

É o mais simples dos métodos de pesquisa.
O conjunto de informações a ser pesquisado pode estar apresentado em qualquer ordem , ou mesmo sem ordem.
Consiste em uma varredura de todas as informações do conjunto até que :

- **Encontremos a informação desejada,**
- **Ou que não existam mais informações a serem pesquisadas.**

A pesquisa seqüencial tem as seguintes características:

- **Simples**
- **Pouco Eficaz**
- **Único método disponível em dispositivos de acesso seqüencial (fita magnética)**
- **Independe da ordem na qual as informações estão expressas**
- **Recomendado para pequenos vetores(menor ou igual a 100 elementos)**

ALGORITMO DE PESQUISA SEQUENCIAL

```
INICIO
ARGUMENTOS
        NUMEROS : MATRIZ [N] DE INTEIROS
        N:INTEIRO
        ELEMENTO_PRO:INTEIRO
VARIAVEIS:
        POS:INTEIRO
PROCEDA PESQUISA_SEQUENCIAL
        CONT : = 1
        ENQUANTO CONT < = N
                SE NUMEROS [CONT] = ELEMENTO_PRO
                        RETORNE [CONT]
                FIM_SE
                CONT : = CONT + 1
        FIM_ENQUANTO
        MOSTRE (“ARGUMENTO NÃO ENCONTRADO”)
FIM.
```

ALGORITMOS E ESTRUTURAS DE DADOS

Pesquisa binaria:

A pesquisa binaria é um dos mais eficientes métodos de pesquisa, porem exige que as informações a serem pesquisadas estejam classificadas.

Supondo que a classificação seja em ordem crescente, procedesse da seguinte forma:

- Para a pesquisa trabalha-se com dois limites, o superior e o inferior.
- O limite inferior a ser pesquisado corresponde a primeira informação do vetor
- O limite superior corresponde a ultima informação do vetor.
- A pesquisa e sempre feita para a média:
  Média = (Limite inferior + Limite superior) / 2
- Se a informação pesquisada estiver acima da média, o limite inferior passa a ser a média +1
- Se a informação buscada estiver abaixo da média, o limite superior passa a ser a média - 1

- Este processo se completa quando :
  - A informação pesquisada é encontrada,
  - Ou o limite inferior é maior que o limite inferior, neste caso o argumento não existe no conjunto pesquisado.

Características:

- **Muito Eficaz**
- **Fácil entendimento**
- **Não funciona em dispositivos de acesso seqüencial**
- **Só funciona se as informações estiverem ordenadas**

Pesquisa Binaria:

ALGORITIMO DE PESQUISA BINARIA:

```
INICIO
ARGUMENTOS
        NUMEROS:MATRIZ [N] DE INTEIROS
        N:INTEIRO
        ELEMENTO_PRO:INTEIRO
VARIAVEIS
        POS:INTEIROS
        SUP,INF,MED:INTEIROS
PROCEDA PESQUISA_BINARIA
        SUP: =N
        INF: =1
        REPITA
                MED: = TRUNCA((SUP+INF) / 2)
                CASO
                ELEMENTO_PRO > NUMEROS [MED]
                        INF: = MED + 1
                ELEMENTO_PRO < NUMEROS [MED]
                        SUP: = MED - 1
                ELEMENTO_PRO = NUMERO [MED]
                        RETORNE(MED)
                FIM_CASO
        ATE INF > SUP
        FIM_REPITA
        MOSTRE (“ARGUMENTO NÃO ENCONTRADO”)
FIM
```

Pesquisa por Endereçamento Direto:

É um dos mais fáceis métodos de pesquisa, para tanto , o argumento de pesquisa deve ser equivalente ao numero da linha que a informação ocupa dentro do vetor..

Este método de pesquisa possui dois inconvenientes:

- **O argumento de pesquisa deve ser numérico.**
- **A área de memória alocada a sua disposição é quase sempre super dimensionada.**

Características:

- **Muito Simples**
- **Pouco eficaz**
- **Só funciona em dispositivos de acesso direto.**
- **Depende da equivalência do argumento de pesquisa com a posição física do elemento.**
- **Tempo de acesso praticamente instantâneo.**
- **Muito dispersivo na ocupação de memória**

Pesquisa por Endereçamento Indireto (Hashing)

Suponha que você tenha um código formado por três números inteiros, usados como argumento de busca. Se fossemos utilizar o endereçamento direto, teríamos um vetor de 999 posições.

Suponha que destes 999 códigos possíveis, apenas 52 seriam utilizados efetivamente. Uma vez que o desperdício de memória vai ser grande , fazemos uma adaptação ao método de endereçamento direto, através da adoção de uma função que damos o nome de **função de transformação**.

O ideal é que esta função gere o menos numero de **colisões** possíveis.

Chama-se colisão a particularidade que permite que dois argumentos de busca , diferentes, terem o mesmo endereço físico a partir da função de transformação.

Uma boa função de transformação é:

Endereço físico = (resto (argumento/X)+1)

Aonde X é o primeiro numero primo imediatamente acima da quantidade de informações efetivas.

Controlando as Colisões:

Caso aconteça que dois argumentos de busca tenham o mesmo endereço físico, então o segundo argumento é colocado no próximo endereço físico livre , após a colisão, encadeando-se os ponteiros do vetor.

ALGORITMOS E ESTRUTURAS DE DADOS

Classificação :

A classificação ou ordenação de dados é uma das tarefas mais usuais no funcionamento dos sistemas. Alguns métodos de pesquisa exigem que o conjunto de dados a ser pesquisado esteja classificado de acordo com uma ordem conhecida.
Também, grande parte dos relatórios necessita de informações classificadas de acordo com uma ordem pré-determinada.

É aconselhável que a classificação fique restrita a um mínimo necessário por duas razões:

- **Absorve grande quantidade de memória secundaria(disco)**
- **É uma tarefa muito demorada**

A ordenação física de um arquivo é dividida em duas etapas:

- **Classificação Interna:**
  O arquivo é carregado em partes na memória principal, cada parte é então classificada sendo o seu resultado devolvido a memória secundaria.

- **Classificação Externa:**
  Nesta etapa os blocos ordenados são fundidos em um só arquivo, agora totalmente classificado.

Dentre os métodos mais comuns de classificação interna, estudaremos 2 famílias:

- **A classificação por troca**
- **E a por inserção**

Classificação por Inserção

Método da Inserção Direta

A característica deste método é que o vetor a ser classificado fica sempre dividido em dois segmentos.
O primeiro segmento, inicialmente contem somente o primeiro elemento da lista, estando portanto classificado.
O segundo segmento da lista , inicialmente contem todos os outros elementos do vetor.
A partir de então todos os elementos do segundo segmento são incluídos no primeiro ,um a um, de forma que o primeiro segmento esteja sempre classificado. O processo termina quando não existirem mais elemento no segundo segmento.

```
INICIO
ARGUMENTOS
        NUMEROS: MATRIZ [N] DE INTEIROS
        N:INTEIRO
VARIAVIES
        CONT,DIV:INTEIROS
PROCEDA INSERCAO_DIRETA
        DIV : = 1
        REPITA
                ELEMENTO: = NUMEROS [DIV + 1]
                CASO
                        ELEMENTO=NUMEROS[DIV]
                                DIV : = DIV + 1
                        ELEMENTO > NUMEROS[DIV]
                                DIV: = DIVI + 1
                        ELEMENTO <  LISTA[DIV]
                                CONT : = DIV
                                ENQUANTO ELEMENTO< NUMEROS[CONT] E CONT > 0
                                        NUMEROS [CONT + 1]: =NUMEROS [CONT]
                                        NUMEROS [ CONT]) : = ELEMENTO
                                        CONT  : =  CONT +1
                                FIM_ENQUANTO
                                DIV: = DIV + 1
                FIM_CASO
        ATE DIV = N
        FIM_REPITA
FIM
```

Classificação por Troca

Método da Bolha (Bubble)

Neste método cada elemento é comparado com o seu sucessor, e se o elemento for maior que o seu sucessor eles trocam de posição.

O processo de troca atravessa todo o vetor.

A classificação fica concluída quando em um atravessamento não for executada nenhuma troca.

```
INICIO
ARGUMENTOS
        NUMEROS : MATRIZ [N] DE INTEIROS
        N:INTEIRO
VARIAVEIS
        POS: INTEIRO
        AUX:INTEIRO
PROCEDA BUBBLE_SORT
        REPITA
                TROCA : = FALSO
                POS : = 1
                ENQUANTO POS <  N
                        SE NUMEROS [ POS ]  >  NUMEROS [ POS + 1 ]
                                AUX: = NUMEROS [ POS + 1 ]
                                NUMEROS [ POS + 1 ] : = NUMEROS [ POS ]
                                NUMEROS [ POS ] : = AUX
                                TROCA : = VERDADEIRO
                        FIM_SE
                        POS : = POS + 1
                FIM_ENQUANTO
        ATE TROCA = FALSO
        FIM_REPITA
FIM
```

ALGORITMOS E ESTRUTURAS DE DADOS

Método da Troca e de Partição(Quick Sort)

Foi proposto por C.A. Hoare em 1962 e é considerado como o mais rápido em relação aso métodos apresentado anteriormente.
O método adota como principio que é mais fácil e rápido classificar 2 vetores com N/2 elementos cada um , do que um único vetor com N elementos.
O Quick Sort funciona de dividindo o vetor em duas partes,e tendo um elemento central no meio do vetor. Deposi disso é movimentando para a primeira parte os elementos menores que o elemento central e para a segunda parte os elementos maiores que o elemento central.
Depois disso , o processo se repete para as partes 1 e 2 do vetor original, e assim consecutivamente até o vetor esta completamente classificado.

```
INICIO
ARGUMENTOS
        VETOR : MATRIZ [N ] DE INTEIROS
        ESQ:INTEIRO
        DIR:INTEIRO
VARIAVEIS
        I:INTEIRO
        J:INTEIRO
        TRAB:INTEIRO
        AUX:INTEIRO
PROCEDA QUICKSORT
        I : = ESQ
        J : = DIR
        TRAB : = VETOR [ TRUNCA((ESQ + DIR) / 2) ]
        REPITA
                ENQUANTO VETOR [ I ] < TRAB
                        I : = I + 1
                FIM_ENQUANTO
                ENQUANTO VETOR [ J ] < TRAB
                        J : = J - 1
                FIM_ENQUANTO
                SE  I < = J
                        AUX : = VETOR [ I ]
                        VETOR [ I ] : = VETOR [ J ]
                        VETOR [ J ] : = AUX
                        I : = I + 1
                        J : = J – 1
                FIM_SE
        ATE I  > J
        FIM_REPITA
        SE L <  J
                QUICKSORT( L , J )
        FIM_SE
        SE I < J
                QUICKSORT( I , R )
        FIM_SE
FIM.
```