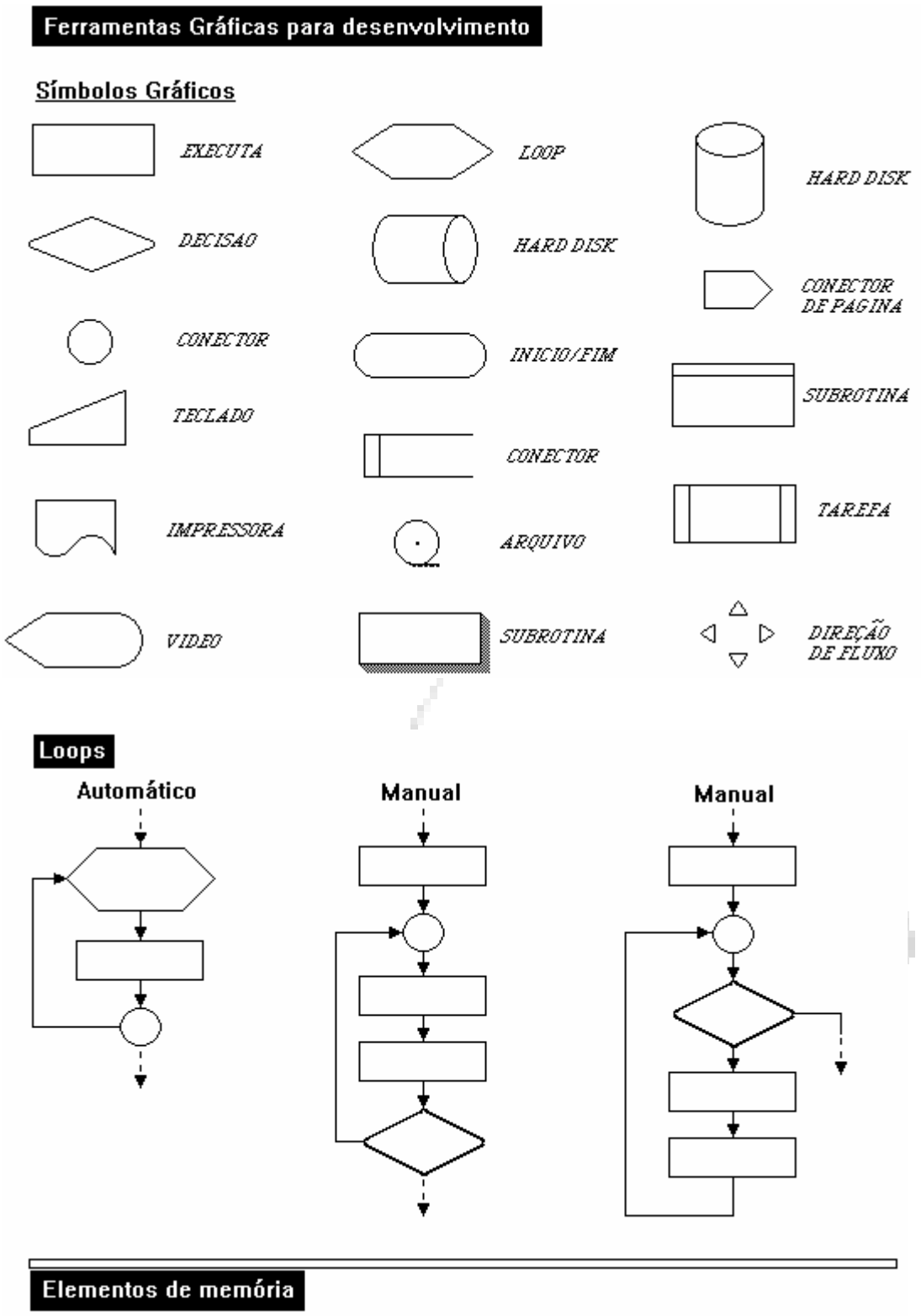
TÉCNICAS DE PROGRAMAÇÃO COM PASCAL

© Prof. Eng° Luiz Antonio Vargas Pinto
www.vargasp.net

Programar: Desenvolver uma seqüência ordenada de comandos para atingir um fim ou alguns; [O poder da programação é máximo em tarefas repetitivas]

Criar - É a capacidade de inventar novas maneiras aprimorando-as.



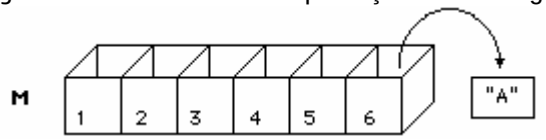
Variável: Posição da memória **RAM** Utilizada para guardar dados. Em termos de computação é o local onde armazenamos dados. Exemplo: A:=3 (Isto significa que foi guardado o valor 3 na variável A). São usadas para operação lógica ou aritmética. Devem ser declaradas na forma:

Nome_da_variável:Tipo_da_variável;

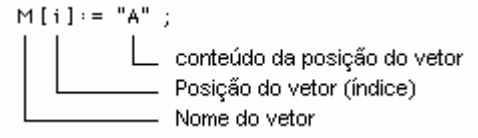
Exemplo: Se A=7 e B=4 e fizermos A=B ,então A=4 e B=4 pois apenas alteramos o destino e nunca a origem. Nessa situação, dizemos que A e B são adimensionais porquê servem para armazenar dados simples (único).

Vetores: São variáveis mais complexas que permite armazenar mais que uma informação na mesma variável, desde que estas sejam do mesmo tipo. Existem duas formas padronizadas para o tratamento de índice de armazenamento em vetor.

Exemplo: Seja o vetor M com 6 posições onde guardamos letras:



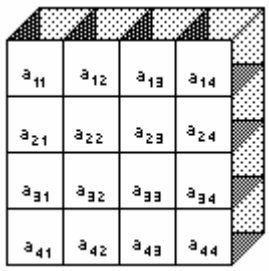
Onde o acesso é processado da seguinte forma:



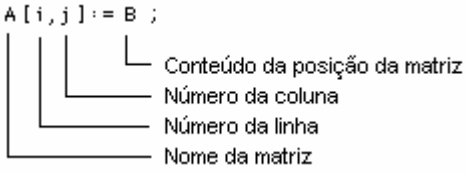

```
Exemplo: M[6]:= “A”;
     write(Linha[3]);
```

Estas são denominadas de **variáveis dimensionais** porquê requerem reserva de memória, isto é, é necessário dimensionar a quantidade máxima de memória que iremos consumir, mesmo que não a utilizemos na totalidade. Um vetor é uma matriz linha, isto é, uma linha e n colunas ao passo que a matriz possui linhas e colunas.

Exemplo: Seja a matriz A com 4 linhas e 4 colunas, 4 x 4 (lê-se 4 por 4) totalizando 16 posições onde guardamos números inteiros.

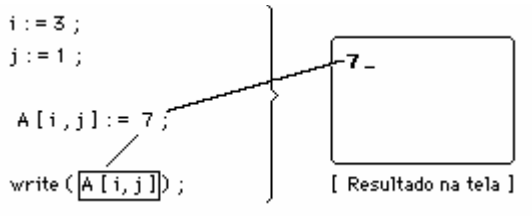


Onde o acesso é processado da seguinte forma:



Exemplo: A[2,3]:=B; {trata-se de uma variável de nome "B" cujo conteúdo é um número inteiro }
Write(Linha[7,1]);

Como o acesso é feito por meio de índices, também podemos acessar informações usando variáveis como índice:



Existem duas formas padrão de controlar índices de acesso de matrizes e vetores:

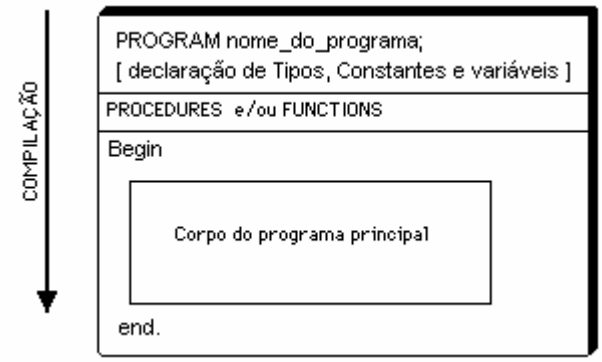
a)**Pré-Indexada**: Quando o ponteiro aponta para a última posição usada, isto é, sempre que for preciso colocar um novo valor primeiro avançamos o ponteiro.

b)**Pós-Indexada**: Quando o ponteiro aponta para uma posição livre, e no armazenamento, primeiro o dado entra e depois o ponteiro avança.

A linguagem PASCAL

Criada por **Nicklaus Wirth** em 1968, na Suíça com o propósito de instituir uma linguagem algorítmica e estruturada de acordo com a nova geração de computadores que estava em desenvolvimento.

Estrutura do Programa



Declarando tipos [Type]

Para que declarar um tipo? Para organizar melhor o programa. É usual declararmos matrizes e vetores, assim como registros com uso de tipos. Declaramos no início do programa precedido da palavra reservada Type.

Exemplo:

```
Type
  X=Array [1..7] of Byte;
Var
  S:x;
```

1-**Byte**:	valores compreendidos entre 0 e 255	[numérico-inteiro]
2-**Integer**:	valores compreendidos entre -32768 e 32767	[numérico-inteiro]
3-**Word**:	valores compreendidos entre 0 e 65.535	[numérico - inteiro]
4-**Shortint**:	valores compreendidos entre -128 e 127	[numérico - inteiro]
5-**Longint**:	valores compreendidos entre -2.147.483.648 e 2.147.483.647	[numérico - inteiro]
6-**Real**:	valores fracionários e exponenciais entre 2.9E-39 a 1.7E38	[numérico - fracionário]
7-**Char**:	ocupa 1 byte, mas só guarda caractere.	[alfanumérico]
8-**Boolean**:	lógico - True/False	[Booleano]

Declarando variáveis [Var]

Exemplo:

```
Program Teste;
Var
  X: Real;
  A,B,C: Integer;
  Nome: String [10];
  Teste: Boolean;
```

Declarando Constantes [Const]

A grande diferença entre o uso de constantes e variáveis reside no fato de que a variável mantém os dados ali guardados indefinidamente, porém podem ser alterados infinitas vezes durante o curso de um programa. Quando precisamos de valores de referencia então nos valemos de uma declaração ao compilador sobre este fato mencionando o valor de referência. São sempre precedidas da palavra reservada **Const**.

Exemplo:

```
Const
  Soma=40;
  Nome='Luiz';
```

Loop

Um dos melhores momentos em que o computador aparentemente supera a mente humana reside justamente na sua capacidade de executar tarefas, mesmo elementares, muito rápida e eficientemente. Aqui também reside talvez o maior de seu desempenho. Computadores são extremamente eficientes em tarefas repetitivas, por esta mesma razão é existem muitas opções de laços (LOOPs) nas linguagens de programação.

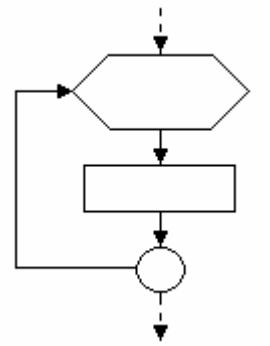
For

Características: Automático, step unitário inteiro crescente/decrescente;
Sintaxe:

For variável_contadora := início **to** fim **do**
Begin

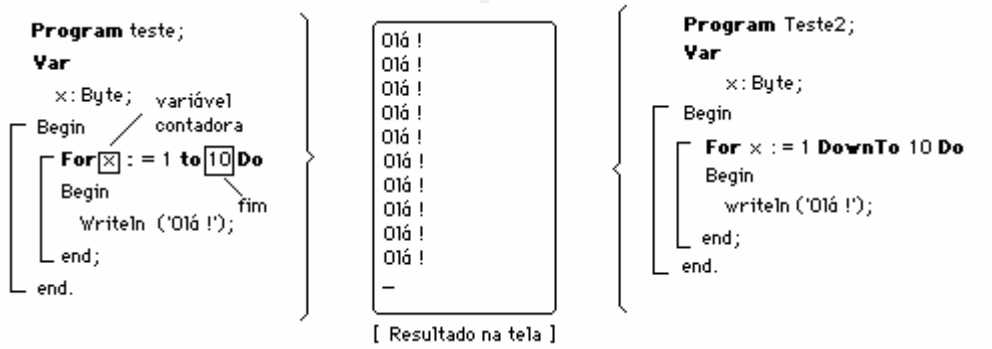
end;

Fluxo:



Descrição: O comando **For** é utilizado quando desejamos efetuar uma ou algumas operações num certo número finito (limitado) de vezes sem interrupção É o chamado Loop automático, onde a atualização da variável contadora é feito pelo próprio comando. Não é possível a interrupção do loop, a menos após o vencimento do limite final do contador. Em Pascal, o incremento é unitário positivo (crescente) ou negativo (decrescente). A variável utilizada como contador deve ter seu nome declarado pelo programador no comando **For** e deve ser declarada no início do programa como algum tipo numérico inteiro.

Exemplo:



Repeat

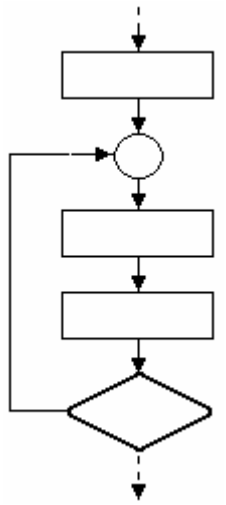
Características: Manual, Step controlado pelo programador podendo ser inteiro, lógico ou Fracionário, busca condição de saída.

Sintaxe:

Repeat

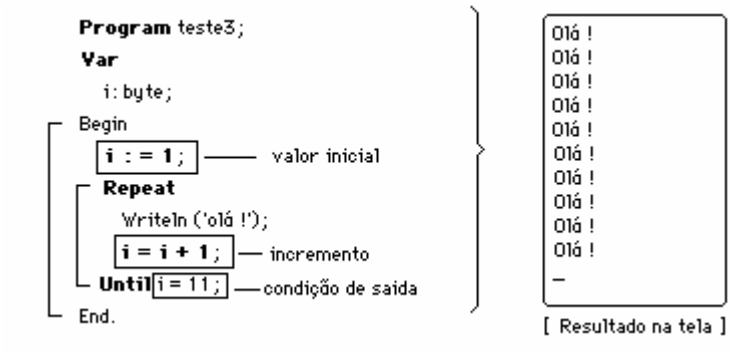
Until < condições > ;

Fluxo:

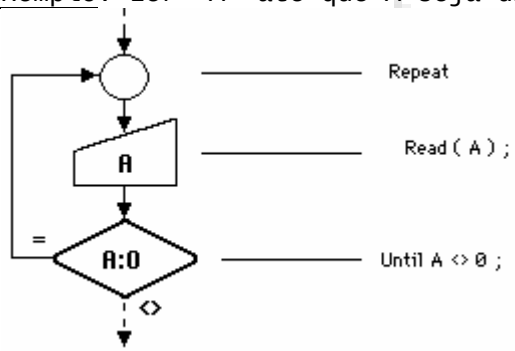


Descrição: Um dos comandos de repetição. É manual porquê o controle sobre o número de vezes que este comando executa um ou uma série de comandos está sob o controle do programador. Além disso, por essa razão podemos interromper um laço apenas determinando a condição de saída. Note que, da própria descrição do comando ele repete um número finito de comandos até encontrar uma condição de saída de laço e que esta condição é estabelecida pelo programador.

Exemplo:



Exemplo: Ler "A" até que A seja diferente de ZERO (0).



While

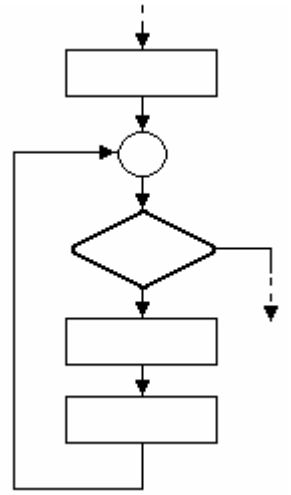
Características: Manual, Step controlado pelo programador podendo ser inteiro, lógico ou Fracionário, busca condição de permanência.

Sintaxe:

While < condições > **Do**
Begin

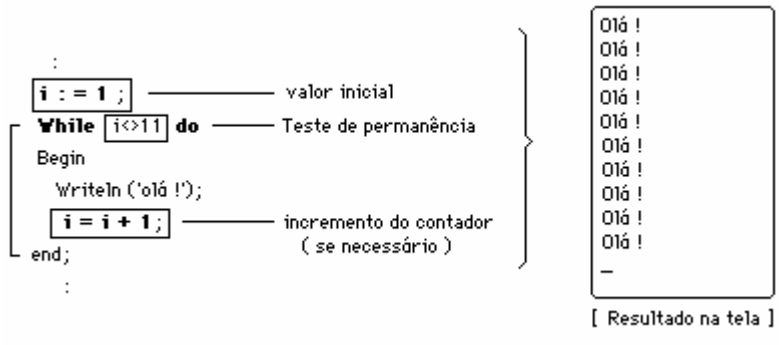
end;

Fluxo:

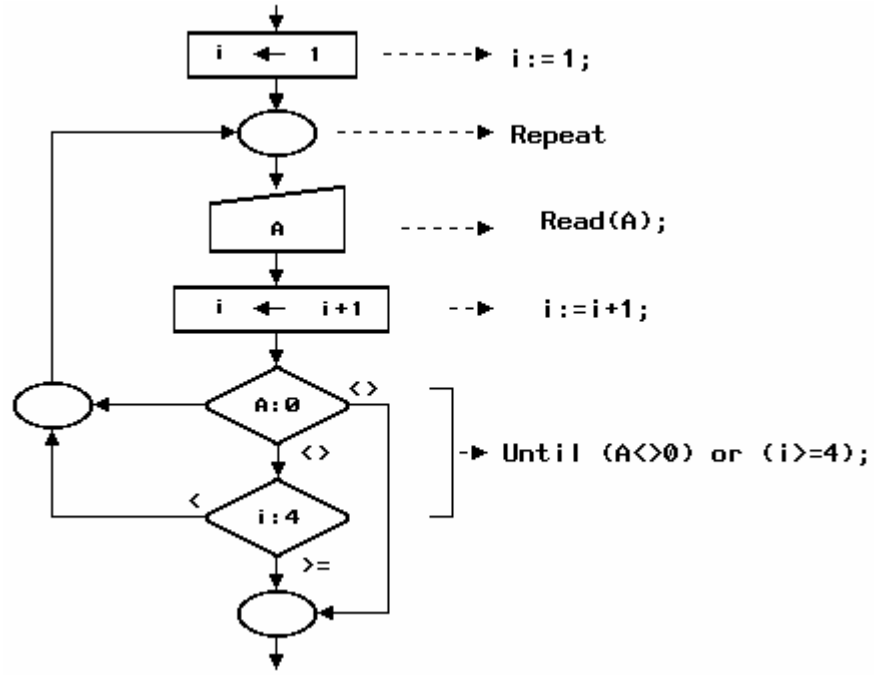


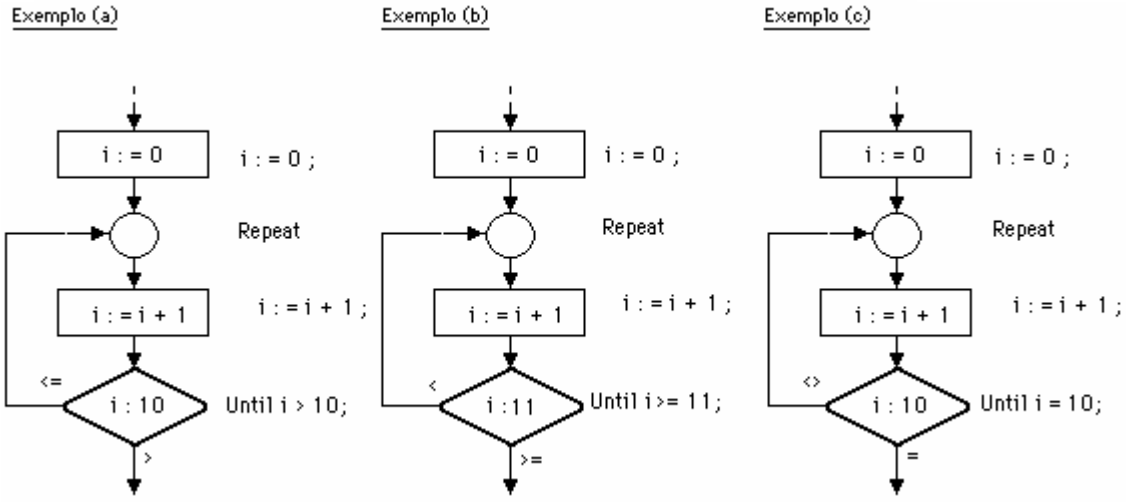
Descrição: É o outro comando de repetição manual. Semelhante ao Repeat porém com características próprias. O número de vezes que ele controla está sob o controle do operador. Sua principal característica é que o Loop é testado na sua condição de permanência no loop enquanto o Repeat procura a condição de saída do loop. Inclusive, no primeiro encontro da linha de comando, este testa inclusive se existe condição de entrada no loop.

Exemplo:



Observe que se o contador foi iniciado com valor 11, por exemplo, o loop não será executado porquê o teste permite a permanência no loop somente enquanto i for diferente de 11 o que não seria o caso (pois i=11).





Podemos utilizar qualquer comando de repetição indistintamente, salvo quando certas condições assim exigirem. É incorreto afirmarmos que um deles é mais rápido que o outro. Na verdade o procedimento que podemos adotar deve respeitar os limites de cada comando. É por essa razão que os comandos existem, e são três:

Repeat → Busca condição de encerramento do laço (loop) mas pode interromper
[Controle Manual]

While → Busca condição de permanência no laço mas pode interromper
[Controle Manual]

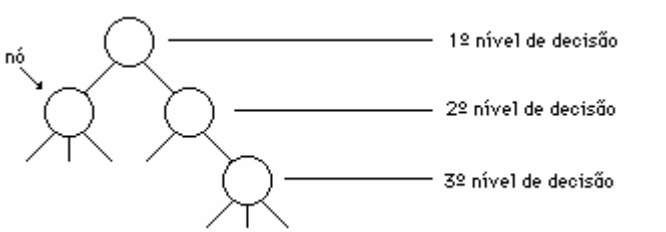
For → Repete um número pré determinado de vezes e não pode interromper
[Controle Automático]

Exemplo: Colocar o valor A sendo que este deve ser diferente de zero. Veja que o comando For seria indicado para este loop, mas neste caso ele está comprometido, pois as condições exigem um número indefinido de tentativas. Isto está de acordo com o descrito anteriormente na definição das condições.

Comandos de decisão

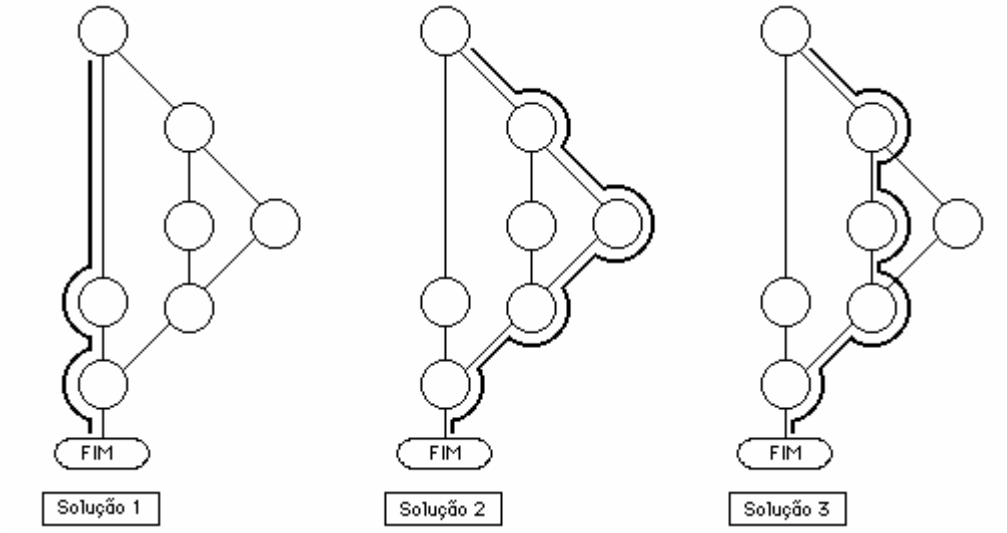
Como basicamente um programa consiste de uma seqüência lógica de "passos à serem executados", em ordem tal que essas execuções de comandos podem resultar num trabalho útil, é perfeitamente aceitável que um programa "tome" ao longo de sua execução decisões lógicas. Alguma coisa assim como uma equação do 2º Grau, o teste do Δ resulta maior ou menor que zero. Isto acarreta efeitos diferentes na seqüência de execução.

Note que o surgimento da decisão divide o programa. Aqui começam os problemas do programador: A velocidade em que a mente processa as informações é, com toda certeza, muito superior a aquele dos movimentos musculares, o que conseqüentemente faz com que pensar seja muito mais rápido que escrever. Claro que, desenvolver um programa não é um caso de velocidade, afinal, eu posso parar para pensar. Mas acontece que o aparecimento da decisão, de forma análoga a velocidade, impede o raciocínio formal. A decisão tornou-se o ponto fraco da programação. E é o que se esperaria que este fizesse. Mas não o faz. Por quê? Qual o caminho a seguir?



É possível verificar que ele é incapaz de optar. Algo assim como se não fosse possível optar por um dos ramos sem olhar os outros. Tecnicamente denominamos a esse ramo da análise o estudo do raciocínio humano. Padrão comum em enxadristas, a visão externa, além de muitos níveis a frente dessa primeira decisão: **Eurística**, que é a análise da lógica da mente humana. Caso típico do **jogo da velha**, **Torres de Hanói**, **Damas** e **Xadrez**, e jogos e sistemas onde apenas a visão do nó (situação) presente não é suficiente.

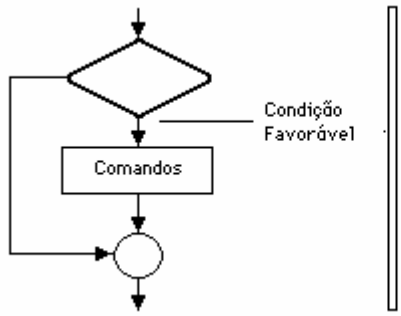
Assim considerando, é claro que seguir esse tipo de raciocínio é inviável para um algoritmo, pois enquanto o raciocínio se aprofunda na análise, o algoritmo computacional conduz todas as decisões ao fim do programa. Por essa razão é também claro que a única forma coerente de criar um fluxograma é optar por um único caminho em cada decisão ignorando o outro e seguir por este até a próxima decisão, se houver, ou até o final do programa. Após ao qual devemos retornar aos níveis deixados em aberto e recomeçar este processo até que todos os nós estejam totalmente fechados. É razoável que todo programa possui uma linha central de procedimento a qual chamamos de coluna vertebral do programa. Veja o seguinte diagrama:



Observe que existem três soluções para o fluxo de dados (as vezes muito mais). Mas observem que estes poderiam ter sido desenvolvidos por programadores diferentes. É claro que a 1ª solução é a melhor delas mas isto não significa que começar por alguma das outras duas seria errado.

Pascal por ser uma linguagem estruturada adota com freqüência estruturas em árvore binária. Isto é o mesmo que dizer que uma decisão tem somente duas alternativas como solução e a ocorrência de uma delas (condição) implica a adoção de uma das alternativas da mesma forma que a não satisfação da condição faça o fluxo seguir pela outra alternativa. Isto é a execução de um comando else. Os comandos de decisões podem ser:

a) Simples:

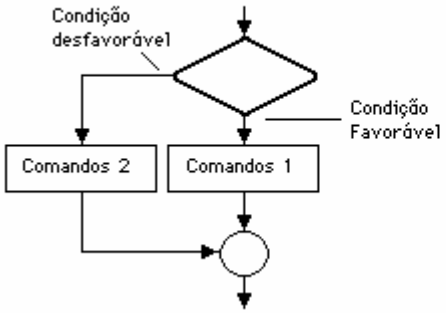


Exemplo: EXCLUSIVIDADE

```
If x = 0 Then      { Condição favorável }
Begin
  Soma := Soma + 1 ;
end;
```

Observe que a ocorrência de x=0 implica no incremento unitário da variável Soma. Note também que o incremento é um efeito de condicional em condição favorável e que neste caso a não ocorrência dessa condição favorável não acarreta nenhuma atuação sobre a variável Soma. Em outras palavras, a ocorrência pode ser tratada como **exclusividade**.

b) Composta:



Exemplo: ALTERNATIVA

```
If x = 0 Then      { Condição favorável }
Begin
  Soma := Soma - 1 ;
end else
Begin
  Soma := Soma / 2 ;
end;
```

Observe que neste caso o comando pura e simplesmente expressa um comando em estrutura de árvore binária. A condição favorável causou um "desvio" no fluxo seguindo para o comando (1). Entretanto, caso esta condição não seja satisfeita, obrigatoriamente o fluxo é desviado para o comando (2). Note que o comando (1) não é executado

Comando de atribuição

Atribuir é carregar um valor em uma ou algumas posições de memória disponíveis. Cada posição de memória que equivale a um endereço de memória armazena um (01) Byte.

Sintaxe: nome_do_destinatario **:=** objeto;

Exemplo:

```
S:=3;
```

Comando de leitura

Este comando serve para passarmos uma informação ao computador através dos dispositivos de entrada. A sua condição default (leia "defô") é feita para teclado. Em comandos de arquivo (breve) voltaremos a usar os comandos read.

Sintaxe: Read (variável_l); ou variável_1 := ReadKey;
Exemplo:

```
Read (a); { onde a é variável tipo char }
```

Esse comando, nessa condição (ainda), lê o teclado e o aguarda a tecla <Enter>. É possível passar diretamente sobre o teclado usando o parâmetro "Kbd" (keyboard) ou com o comando ReadKey.

Exemplo:

```
Read (kbd,a); { Opção válida no Turbo Pascal com uso de USES Turbo3 }
 a:=ReadKey; { Opção válida no Turbo Pascal com uso de USES CRT }
```

Não aguarda o 

e os caracteres não são ecoados no vídeo. Há ainda a restrição de que o programa deve ser notificado disto com o uso da palavra reservada "**USES**".

Exemplo:

```
Program Teste;
Uses Turbo 3; {Libera o uso do "kbd".}
```

Outra variação desse comando é:

Sintaxe: ReadLn (variável_1);

O efeito é que ao receber o dado, coloca o cursor na linha seguinte, na primeira coluna.

Observação: Os comandos **Read/ReadIn** não possuem proteção, isto é, se a variável "a" do exemplo for de algum tipo numérico qualquer, e se o usuário ao dar entrada teclar uma letra ou qualquer caractere não numérico, o comando é interrompido e o programa retorna ao DOS. E ainda mais, estes comandos apresentam problemas quando no uso de variáveis tipo String. Quando um string é lido, um flag interno sinaliza o compilador e qualquer outra leitura de outro String logo a seguir não será efetuada.

Comando de Leitura

Sintaxe: **Write** (variável_1); ou **WriteIn** (variável_1);

Este comando foi destinado a escrita. E escrita no modo computacional, significa escrever um dispositivo (Tela, impressora, saída serial e disco). No modo default (leia "defô") é para escrita na tela. Quando desejamos escrever na impressora devemos:

1º) Declarar os Uses correto:

```
Program Teste;
Uses Printer;
```

2º) Usar o parâmetro "**Lst**":

```
Write(Lst,'Eu');
```

Observações: Este comando também não possui proteção e, por exemplo, uma tentativa de escrita em um dispositivo não liberado acarreta um erro grave e o retorno ao Dos é inevitável. Tanto o **Read** como o **Write** possuem comandos adicionais contra esses "acidentes" que veremos posteriormente. O comando **Writeln** (variável_1); procede da mesma forma que o comando **Readln** quanto á tela.

Comando de posição de cursor em tela texto

Sintaxe: **GoToXY**(Coluna,Linha);
Posiciona o cursor em uma tela tipo texto com 25 linhas e 80 colunas.

Comando para limpar uma linha a partir do cursor

Sintaxe: **ClrEol**; Obs.: o cursor não sai da posição.

Comando para limpar a tela

Sintaxe: **ClrScr**; Obs.: o cursor reinicia na posição (1,1).

Funções Matemáticas

1- Operações lógicas ou aritméticas:
Pascal permite o uso de equações matemáticas. Por ser uma linguagem científica pode executar operações básicas, binárias, e lógicas.

a) Soma: **+**
b) Subtração: **-**
c) Divisão: **/**
d) Multiplicação: *****

A operação "**/**" é necessariamente do tipo Real. Isto se deve ao fato de que uma divisão nem sempre resulta em valores inteiros. Dessa forma a variável resultante em valores inteiros. Dessa forma a variável resultante deve aceitar números em ponto flutuante (Exponencial).

Exemplo:

```
S:=A / B; { S é necessariamente Real }
S:=S / 4; { Se S <> Real o compilador recusa montar o programa }
```

Um fato interessante é que Pascal permite agrupamento, respeitando a hierarquia matemática.

Exemplo:

$R = \frac{3}{4+A}$ equivale a `R:= 3 /( 4 + A);`

$S = \frac{3 \times (A+1)}{\frac{4}{B+C}}$ equivale a `S:= 3 *( A + 1 )/( 4 /( B + C ));`

Entretanto, algumas vezes é necessário obtermos apenas a parte inteira de uma divisão. Podemos usar:

e)**DIV** é divisão real
Exemplo:

$S = \frac{10}{8} \Rightarrow$ 10 \| 8 / 0 1,25 $\Rightarrow$

```
S := 10 / 8 ;    { resulta S = 1,25 }
S := 10 DIV 8 ; { resulta S = 1 }
```

e se usarmos **DIV** poderemos declarar S como: Integer, Word, LongInt.

f)**MOD** é o resto da divisão

Exemplo:

$S = \frac{10}{8} \Rightarrow \begin{array}{r|l} 10 & 8 \\ \hline -8 & 1 \\ \hline \boxed{2} & \end{array} \Rightarrow$ S := 10 **MOD** 8 ; { resulta S = 2 }

g)**SHR** Shift Right (operação binária) - deslocamento para a direita.

$X = 47_{16} = 01000111_2 \Rightarrow$ X := $47 **SHR** 2; equivale dizer que $X = 00010001_2 = 17_{10}$

h)**SHL** Shift Left (operação binária) - deslocamento para a esquerda.

Exemplo:

$X = 47_{16} = 01000111_2 \Rightarrow$ X := $47 **SHL** 2; equivale dizer que $X = 00011100_2 = 28_{10}$

i)**AND** executa a operação lógica AND e tem dois modos de operar:

i.1)**AND** binário :

Exemplo:

$X = 47_{16} = 01000111_2 \Rightarrow$ X := $47 **AND** 23; equivale dizer que:

$$\begin{array}{r} 0\,1\,0\,0\;0\,1\,1\,1 \rightarrow 47_{16} \\ \wedge\; 0\,0\,0\,1\;0\,1\,1\,1 \rightarrow 23_{10} \\ \hline 0\,0\,0\,0\,0\;1\,1\,1 \rightarrow 07_{10} \end{array}$$

i.2)**AND** Lógico:

Exemplo:

```
:
:
If (A > 0) AND (S > 7) Then write('Ok');
:
:
```

Aplicado em condicionais.

j)**OR** executa a operação lógica OR (opera de modo análogo ao **AND**)

j.1)**OR** binário:

Exemplo:

$X = 47_{16} = 01000111_2 \Rightarrow$ X := $47 **OR** 23; equivale dizer que:

$$\begin{array}{r} 0\,1\,0\,0\;0\,1\,1\,1 \rightarrow 47_{16} \\ +\; 0\,0\,0\,1\;0\,1\,1\,1 \rightarrow 23_{10} \\ \hline 0\,1\,0\,1\;0\,1\,1\,1 \rightarrow 71_{10} \end{array}$$

j.2)**OR** Lógico:

Exemplo:

```
:
:
If (A > 0) OR (S > 7) Then write('Ok');
:
:
```

k)**NOT** executa a operação lógica **NOT**

k.1)**NOT** binário:

Exemplo:

$X = 47_{16} = 71_{10} \Rightarrow X := \textbf{NOT}\ X$; equivale dizer que:

$0100\ 0111_2\ (47_{16}) \Rightarrow \textbf{NOT}\ (0100\ 0111_2) = 10111000_2 (B8_{16}) = 184_{10}$.

k.2)**NOT** Lógico:

Exemplo:

```
:
Repeat
   :
   :
Until (Not Teste); {onde Teste é uma variável Booleana}
:
```

l)**XOR** normalmente representa a operação lógica **XOR**, onde:

Exemplo:

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

de onde podemos afirmar que se duas condições são verdadeiras (ambas) ou se ambas são falsas então é sinalizado FALSO.

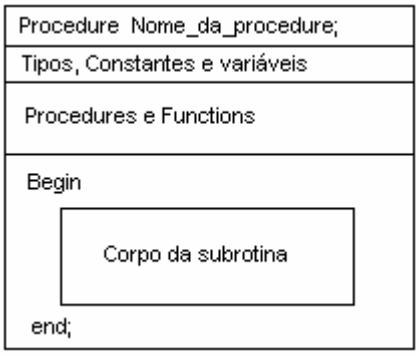
Pascal possui uma biblioteca matemática que aceita funções matemáticas tais como:

Seno ..... Sin(x);
Cosseno..... Cos(x);
Tangente ..... Tan(x);
Raiz Quadrada..... SQRT(x);
Logaritimo Neperiano..... Ln(x);
X^2 ..... SQR(x);

Subrotina (Procedure)

Definição: É um conjunto de comandos separados do programa principal e que satisfazem duas condições básicas:
1- Reduzem os códigos do programa fonte;
2- Permitem programação estruturada.

Estrutura:



Observação: O programa principal não consegue olhar dentro de suas procedures e function mas o contrário é válido, isto é, a procedure pode olhar para as variáveis do programa principal.

Variáveis Globais e Locais

Uma variável é dita Global quando é declarada no programa principal e portanto é de acesso universal por todas demais as partes componentes de um programa. Já aquela que é declarada dentro de uma Procedure ou Function é dita local, pois é acessada exclusivamente pela própria Procedure/Function ou elementos declarados dentro da mesma e não por partes do programa Principal. Muitas vezes é necessário que construamos uma ou mais procedures em um dado programa, no entanto existe casos onde a diferença entre determinadas procedures é pequena e para otimizarmos essas procedures tornando-a uma só, usamos o artifício de passar mais informações à procedure que será a única englobando as outras antigas. A essas informações extras chamamos *parâmetro*. Tecnicamente falando, existem duas formas de passagem de parâmetros: Com retorno e sem retorno.

Parâmetro

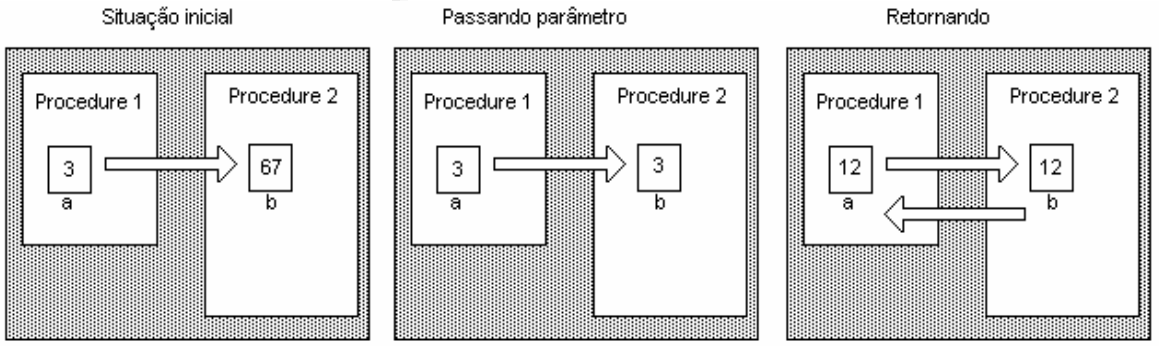
Definição: São dados adicionais usados para simplificar uma *function* ou *procedure*, agrupando mais de uma *function* e/ou *procedure* e com isto melhorando o desempenho de uma dada *function* e/ou *procedure*.

Conforme já dissemos, parâmetros em Pascal podem ser de dois tipos:

a) **Sem retorno**: As informações adicionais são passadas a *procedure* e/ou *function* e o originante permanece inalterado.

b) **Com retorno**: As informações são passadas a *procedure* e/ou *function* e o originante será alterado.

Podemos também definir parâmetro segundo o conceito computacional, ou seja: é o uso de passagem de informação entre áreas de programa. Vejamos agora através de um diagrama:



De acordo com a figura, a informação que desejamos passar é colocada na variável **a** na procedure_1. Em seguida essa informação é transferida para a Procedure_2 na área destacada como sendo a variável **b**. Assim, podemos reafirmar que após a chamada para a Procedure_2, b=a, isto é, o mesmo valor de **a**. Após a conclusão da procedure as alterações efetuadas **b** que é uma variável que pertence exclusivamente a procedure_2 pode ou não retornar a **a** no programa principal sob o controle do programador.

E, nessa condição **b** é chamada Local e **a** é chamada Global. Em Pascal não é permitido declarar variáveis com o mesmo nome em um mesmo nível. Por exemplo

duas variáveis "A" Globais. Quando o compilador monta uma procedure, este verifica se as variáveis pertencentes à procedure foram declaradas nesta. Se afirmativo a compilação prossegue e em caso contrário este verifica se a variável esta declarada no programa principal. Se lá estiver a compilação prossegue e caso contrário para e informa que não conhece a variável.

Função (Function)

Estrutura similar a procedure, diferindo apenas na forma da chamada e na colocação no programa. Enquanto a procedure simplesmente substitui uma parte do programa, a function como o próprio nome diz, tem uma função específica. O uso de funções está relacionado ao processo de cálculos sendo freqüentemente empregado em programas científicos. Da mesma forma que as procedures, as functions aceitam o uso de parâmetros.

Registro (Record)

No uso de elementos de memória sempre surgem casos tipo:

a) Elementos de armazenamento são diferentes;
b) Não é possível determinar o número máximo de elementos que necessitaremos.

O uso de vetores e matrizes permite um controle mais aprimorado, mas apresenta o inconveniente de exigir elementos iguais (mesmo tipo). Embora o uso de índice facilite a elaboração de loops e lógica de acesso, ainda não permite inserção de tipos diferentes no meio do vetor ou matriz. Entretanto, este ainda não é o ponto crítico da questão, pois este pode ser solucionável de forma satisfatória da seguinte forma:

```
Tabela1 : Array [1..3] of String[40]; {nomes}
Tabela2 : Array [1..3] of Real;    {notas}
Tabela3 : Array [1..3] of Byte;    {idades}
```

Fazendo dessa forma podemos utilizar o índice como se fosse o código do aluno e através dele obter as informações necessárias. Considere o índice como sendo uma variável do tipo byte onde índice=2. Dessa forma, se fizermos o acesso ao aluno número2 teremos todas as informações sobre ele. Até aqui, tudo está sob controle. Ocorre, que na prática, ao declararmos uma matriz ou um vetor, obrigatoriamente o compilador reserva memória para todas suas variáveis, mesmo que estas não sejam utilizadas. Isto limita o sistema e é impossível alterarmos isto "On Line". Assim devemos proceder à "Reserva" pensando no máximo possível, se pudermos. Pascal, permite a criação de uma estrutura que não somente resolve o problema de espaço mas também a criação de um tipo de armazenador único para vários diferentes: "**Record**".

Estruturas de dados

a)**Listas**: Consistem de uma relação de algum tipo de objeto ou artigos ou quaisquer outras coisas relacionadas.

exemplos: - Lista de Compras
- Lista Telefônica
- Lista de Classificação.

Em computação uma lista torna-se bastante útil quando serve para armazenar informações, quer seja na forma matricial quer seja na forma de registros.

b)**Fila**: Caracterizada por uma estrutura tipo **FIFO**:
First
In
First
Out

c)**Pilha**: Caracterizada por uma estrutura tipo **LIFO**
Last
In
First
Out

A implementação de uma lista encadeada segue princípios rígidos de estrutura de dados e programação. Como o conceito utiliza ponteiros de memória o que exige precisão nas rotinas sob a pena de perder o controle de sistema.

Ponteiro (Pointer)

O uso de ponteiros de memória além da velocidade dinâmica de memória, ocupando espaço necessário apenas ao seu uso, isto é, se o armazenamento necessitar uma expansão da área ocupada esta vai sendo ocupada na medida da necessidade, assim como se houver diminuição da memória consumida o "sistema se contrai" liberando memória. Em Pascal o tipo Pointer e suas operações são as novidades. Podemos declarar um Pointer para uso com Record e posteriormente uma lista encadeada.

Exemplo:

```
Ponta = ^Seta;
Seta = Record
            Ponteiro: Ponta;
            Name: String[40];
       end;
```

Exercícios

1º) Faça um fluxograma para um programa que recebendo 3 notas, organiza-as em ordem crescente.

2º) Faça o fluxograma para um programa que inverte a posição de um vetor, isto é, o primeiro no último e o último no primeiro e isso sucessivamente. O vetor têm 1.000 posições.

3º) Faça um fluxograma que identifica em uma lista de 2.000 números o maior, o menor e calcula a média. Números inteiros e maiores que zero. Obs:O maior número é 65.535.

4º) Faça um fluxograma para um programa que elimina os dados repetidos de dentro de um vetor com 200 posições. Sendo que esses dados são números inteiros, positivos e maiores que zero.

5º) Fazer um fluxograma para imprimir os dados de um vetor com 2.000 posições e que são maiores que zero ou múltiplos de 3.

6º) Faça um fluxograma para calcular as raízes da equação do 2º grau, mesmo complexas.

7º) Faça um programa que contém uma subrotina que recebe o valor de (a) verifica se é 0, e em caso positivo emite uma mensagem avisando o operador do erro. Considere o exercício para cálculo das raízes da equação do 2º grau.

8º) Refaça o exercício 7º usando uma Function.

Programas exemplo

1 - Fatorial

Definição: O Fatorial de um número é obtido por:

N!=N.(N-1).(N-2).(N-3)...(N-M).1

onde **N** é inteiro e positivo.

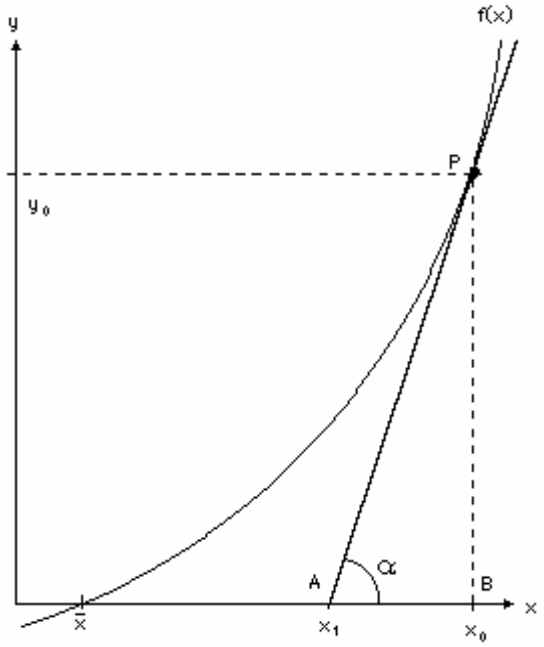
Exemplo:
3! = 3x2x1 = 6
5! = 5x4x3x2x1 = 120

Lógica: se queremos n! então procedemos (n-1) multiplicações.

```
Program Fat;
Uses CRT;
Var
  Fator:LongInt;
  N,i:Byte;
  Z:Char;
Begin
  ClrScr;
  Fator:=1;
  GoToXY(4,2);
  write('Número:');
  Readln(N);
  If N<0 Then
  Begin
    writeln('Näo existe !');
  end else
  Begin
     If N>1 Then
     Begin
       For i:=1 to N do Fator:=Fator * i;
     end;
   end;
   writeln ('Fatorial de ',N,' = ',Fator);
   z:= Readkey;
end.
```

2 - Raízes por Newton-Raphson

Este algoritmo permite determinar com excelente precisão a raiz de uma função contínua em um intervalo [a,b] desde que f(x) e f'(x) sejam não nulas nesse intervalo e que a raiz esteja nesse intervalo. Utilizamos este algoritmo pelo fato de que podemos aplicá-lo a cálculo da função raiz quadrada, que neste caso é de nosso interesse além do que este algoritmo é vastamente utilizado pela maioria dos aparelhos de cálculo e compiladores de linguagens de programação existentes. É um processo de cálculo numérico e portanto perfeitamente aplicável á computadores.



Dada a curva: $f(x) = y = \sqrt{x}$

então $x^2=a$ de onde $x^2-a = 0$, e dado um ponto genérico P_0 pertencente a curva, resultante em f(x) e passando uma reta tangente ao ponto $P_0(x_0,y_0)$ determinamos o triângulo APB retângulo, Daí:

$$Tg\,\alpha = \frac{y_0}{x_0 - x_1} \quad \text{mas} \quad Tg\,\alpha \stackrel{N}{=} f'(x) \quad \text{(Derivada de 1º ordem)}$$

E portanto deduzimos que:

$$f'(x) = \frac{y_0}{x_0 - x_1} \Rightarrow x_0 - x_1 = \frac{y_0}{f'(x_0)} \Rightarrow x_0 - x_1 = \frac{f(x_0)}{f'(x_0)} \Rightarrow \boxed{x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}}$$

ou seja, x_1 é obtido a partir de x_0 e esta mais próximo da raiz x. Este processo interativo é repetido infinitas vezes, ou tanto quanto mais próximo do valor desejado (precisão). Todo processo numérico é tratado dessa forma, isto é, nós o processamos até obtermos um valor o mais próximo possível do desejado, tantas vezes quanto for necessário. É um processo extremamente rápido, chegando ao resultado desejado em poucas interações.

```
Program SQRoot;
Uses CRT;
Var
  Erro,X,Sup,Atual,Dif,Ref,Ant:Real;
  Z:Char;
Begin
  ClrScr;
  GoToXY(5,2);
  write('Raíz:');
  Read(X);
  If X>0 Then
  Begin
    Erro:=0.00001;
    Ref:=4*X;
    Ant:=0;
    Dif:=1;
    While Dif>Erro do
    Begin
      Atual:=Ref-(Ref*Ref-X)/(2 * Ref);
```



```
      Dif:=ABS(Atual-Ant);
      Ant:=ABS(Atual);
      Ref:=Ant;
    end;
    Atual:=ABS (Atual);
    Dif:=SQRT(X)-Atual;
    Writeln ('A raíz de ',X:4:8,'= ',Atual:4:8);
    write ('A diferença do algorítimo do compilador PASCAL é de ',Dif);
 end else
 Begin
   write('Não existe raíz par de número negativo..');
 end;
 z:=Readkey;
end.
```

3- Fila encadead

```
Program fila_encadeada;
Uses CRT;
Const
   Max_Reg=3;
Type
   Ponta = ^Pointer;
   Pointer = Record
         Apt:Ponta;
         Nome:String[10];
         Code:Integer;
       end;
Var
  Prilin,Newlin,NextLin,Freelin:Ponta;
  i:Byte;
  z:Char;
Begin
  ClrScr;
  Prilin:=Nil;
  For i:=1 to Max_reg do
  Begin
    New(Newlin);
    Write('Product:..');
    Readln(Newlin^.Nome);
    write('Code:.....');
    Readln(Newlin^.Code);
    If Prilin=Nil Then
      Prilin:=Newlin
    else
      NextLin^.Apt:=Newlin;
    NextLin:=Newlin;
    NextLin^.Apt:=Nil;
  end;
  Freelin:=Prilin;
  Repeat
    write('Name of Product =');
    writeln(FreeLin^.Nome);
    write('Code of Product =');
    writeln(Freelin^.Code);
```

```
    Freelin:=Freelin^.Apt;
  Until Freelin=Nil;
  Z:=ReadKey;
end.
```