Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e Computação

Desenvolvimento de um Programa Aplicativo de Uso para Deficientes Visuais que Proporciona a Implementação de cálculo de Formas Matemáticas num Editor de Texto

Autor: Julián Mauricio Prada Sanmiguel
Orientador: Prof. Dr. Luiz César Martini

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação da UNICAMP como parte dos requisitos exigidos para obtenção do título de Mestre em **Engenharia Elétrica**.

Comissão Examinadora

Prof. Dr. Luiz César Martini – FEEC / UNICAMP
Prof. Dr. José Antônio dos Santos Borges – NCR / UFRJ
Prof. Dr. Rita de Cássia Ietto Montilha – FCM / UNICAMP

Campinas, SP – Brasil
Abril / 2010

Este exemplar corresponde a redação final da Dissertação/Tese defendida por JULIAN MAURICIO PRADA SANMIGUEL e aprovada através da comissão julgada em 26 ABR 2010

Orientador

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

P881d

Prada Sanmiguel, Julian Mauricio
Desenvolvimento de um programa aplicativo de uso para deficientes visuais que proporciona a implementação de cálculo de formas matemáticas num editor de texto / Julian Mauricio Prada Sanmiguel. -- Campinas, SP: [s.n.], 2010.

Orientador: Luiz César Martini.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Calculadoras programáveis. 2. Deficientes visuais. 3. Interpretadores (Programas de computador). 4. DELPHI (Linguagem de programação de computador). I. Martini, Luiz César. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Development of an application program to use for the visually impaired that provides the implementation of calculation of mathematical forms in a text editor
Palavras-chave em Inglês: Programmable calculator, Visually impaired, Interpreters (Computer programs), DELPHI (Computer program language)
Área de concentração: Telecomunicações e Telemática
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: José Antonio dos Santos Borgues, Rita de Cássia Ietto Montilha
Data da defesa: 26/04/2010
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Julian Maurício Prada Sanmiguel

Data da Defesa: 26 de abril de 2010

Título da Tese: "Desenvolvimento de um Programa Aplicativo de Uso para Deficientes Visuais que Proporciona a Implementação de Cálculo de Formas Matemáticas num Editor de Texto"

Prof. Dr. Luiz César Martini (Presidente): _____

Prof. Dr. José Antônio dos Santos Borges: _____

Prof. Dr. Rita de Cássia Ietto Montilha: _____

Para meus pais, Olaya e Claudia;

meu irmão, Ricardo;

e minha família;

Pelo Amor, Compreensão e Apoio infinito,

por me ajudar a que este momento chegara...

Agradecimentos

Agradeço profundamente ao Prof. Dr. Luiz Cesar Martini pela oportunidade que me deu de me especializar e pela orientação e paciência durante estes anos de estudo e pela confiança em mim depositada.

Agradeço especialmente a meu pai, e minha mãe pelo apoio incondicional sempre. Por acreditar em mim e me proporcionar as ferramentas necessárias para enfrentar todas as situações que vão acontecendo no caminho.

Também agradeço enormemente a meus amigos e família em Brasil Charlie Way, Ana Maria, Maria Isabel, Jeffrey, Nano, Jairo, Oscar, Monica, Camilo, Laura, Sandra, Sergio e todos os demais amigos que estiveram ali conmigo.

Ao Prof. Dr. Leonardo Sousa Mendes pelo apoio laboral no primeiro ano no Brasil e finalmente a CAPES e CNPq pela ajuda financeira que foi fundamental para que eu pudesse realizar meu trabalho de mestrado.

Resumo

A falta de recursos computacionais com aplicativos matemáticos voltados para deficientes visuais e outros tem impedido que os deficientes estudem e desenvolvam trabalhos nas áreas das ciências exatas. Este tema de tese visa basicamente ao desenvolvimento de uma calculadora científica programável que é ativada no próprio editor de textos de um programa gratuito conhecido por DOSVOX, disponível especificamente para uso dos portadores de necessidades especiais. A calculadora foi desenvolvida de tal forma que permite ao usuário criar algoritmos e cálculos matemáticos desde o editor de textos (EDIVOX), o qual permite aproveitar os recursos de sínteses de fala com os que este conta, o que a sua vez facilita enormemente o processo de criação, manipulação e execução destes.

Palavras-chave: Calculadora cientifica programável, Deficientes visuais, DOSVOX, EDIVOX, Editor de textos, Compilador, Interpretador, Tradutor, DELPHI

Abstract

The lack of computational resources with mathematical applications designed for the visually impaired and others has prevented them to study and develop work in the fields of exact sciences. This thesis topic points essentially to the development of a scientific programmable calculator which is activated from the text editor of a free program known as DOSVOX available specifically for use by persons with special needs. The calculator was developed in a way that allows users to create algorithms and mathematical calculations from a text editor (EDIVOX), which allows the calculator using speech synthesis resources it possesses, which in turn greatly facilitates the process creation, manipulation and implementation of these.

Keywords: Programmable scientific calculator, Visually impaired, DOSVOX, EDIVOX, Text Editor, Compiler, Interpreter, Translator, DELPHI

Sumario

Lista de Figuras

Lista de Tabelas

Capitulo 1

Introdução

1.1. Motivação

Na atualidade o avanço da tecnologia na área da computação tem gerado a presença massiva da informática na vida cotidiana das pessoas, o qual tem permitido o desenvolvimento de numerosas ferramentas para a educação, para o entretenimento, para a aquisição de informação e para as mais diversas atividades. Devido a isto, estes tipos de ferramentas converteram-se em parte indispensável da sociedade, pelo qual é muito importante permitir, facilitar e garantir a inclusão digital de todo tipo de pessoas neste médio.

A utilização da informática proporciona numerosas vantagens a quaisquer pessoas, sem importar sua condição física, devido a que esta, deixou de ser uma tecnologia voltada unicamente para programadores, analistas e pessoas com atividades com fines produtivas, para se converter numa importante fonte de informação para todo tipo de usuários. Por isso é fundamental e indispensável que estas novas tecnologias e aplicativos informáticos, possam ser voltados para suprir as necessidades especificas de qualquer perfil de usuário, dependendo das características próprias dele.

Embora que, é visto que as ferramentas e aplicativos informáticos podem ser vantajosos na solução de problemas do cotidiano, melhorando direitamente a qualidade de vida e o acesso à informação, atualmente as pessoas portadoras de necessidades especiais têm diversos obstáculos para o uso de ferramentas computacionais, já que ainda que para elas existam diferentes recursos

de *Software* no mercado, alguns destes podem ser melhor aproveitados se são superados obstáculos de acessibilidade e funcionalidade nos mesmos.

A acessibilidade é definida como o grau no qual todas as pessoas podem utilizar um objeto, visitar um lugar ou aceder a um serviço, independentemente das suas capacidades técnicas, cognitivas ou físicas. Ela é uma condição necessária para a participação social das pessoas com distintas limitações funcionais e é garante de um melhor desenvolvimento para todos.

De forma similar ao que ocorre no médio físico, um desenvolvimento de software indiferente à acessibilidade cria obstáculos desnecessários e inconvenientes para todos, já sejam, para pessoas com alguma deficiência ou não, já que isto pode dificultar o aproveitamento de todas as capacidades, recursos, e características próprias de determinado software. Por isso é importante que os aplicativos de software contemplem os diferentes requisitos das pessoas com algum grau de deficiência para assim, aproveitar todo o potencial que podem chegar a oferecer a informática, na comunicação e na aprendizagem.

1.2. Objetivos

A falta de recursos computacionais com aplicativos matemáticos orientados a pessoas com necessidades especiais, não têm permitido que pessoas com deficiência visual possam estudar e desenvolver aplicativos e algoritmos matemáticos nas áreas das ciências exatas.

Devido a esta necessidade, o objetivo principal deste trabalho consistiu no desenvolvimento de uma ferramenta computacional (MATVOX) destinada a deficientes visuais, a qual permitisse desenvolver algoritmos e aplicativos matemáticos desde um editor de textos, sem a necessidade de ferramentas externas para sua execução.

A implementação do MATVOX foi feita desta forma, devido a que uma pessoa com deficiência visual precisa ter todo na sua mão (texto, algoritmo, resultados), já que para ela não é conveniente ter numa parte os cálculos ou algoritmos matemáticos, e em outra parte os

resultados. Por este motivo decidiu-se implementar todo num editor de textos, o qual é umas das ferramentas informáticas mais utilizadas na vida cotidiana destas pessoas.

Para atingir este objetivo, o MATVOX está baseado no editor de textos (EDIVOX) de um sistema gratuito conhecido como DOSVOX, disponível especificamente para pessoas com necessidades especiais. Este editor de texto é muito utilizado no Brasil por pessoas com deficiência visual, devido a sua facilidade de uso e às características próprias da sua interface, todo isto permite que o MATVOX possa ser utilizado por um grande número de usuários no Brasil.

Por outra parte, o DOSVOX foi escolhido como plataforma do MATVOX devido a três características fundamentais:

- O DOSVOX é um software gratuito que pode ser baixado livremente da internet.
- Seu código é aberto e está disponível na internet, o que permite o desenvolvimento e o acoplamento de aplicativos novos, neste caso do MATVOX.
- O DOSVOX foi desenvolvido com todas as características e necessidades próprias dos deficientes visuais, o qual permitiu aproveitar a interface única deste sistema.

Todas estas características permitem que o MATVOX esteja ao alcance de qualquer pessoa garantindo que nada será cobrado por sua utilização e a sua vez permitindo aproveitar a interface única do DOSVOX, voltada especificamente para deficientes visuais.

Capitulo 2

2.Fundamentação Teórica

Este capitulo apresenta os fundamentos teóricos dos processadores de linguagem, já que a ferramenta desenvolvida (MATVOX) é um aplicativo que interpreta e executa uma linguagem matemática. Para isto é importante conhecer os diferentes métodos de interpretação e as etapas envolvidas neste processo. Por outra parte é feito uma comparação entre os interpretadores e os compiladores, a qual vai permitir entender porque para a execução do MATVOX é melhor interpretar é não compilar o algoritmo matemático.

Na segunda parte do capitulo é apresentada uma introdução sobre a importância das interfaces computacionais nos aplicativos informáticos para pessoas com deficiência visual e como estes permitem melhorar a comunicação Homem – Computador.

2.1. Processadores de linguagem

Uma linguagem de programação é um método padronizado para expressar instruções para um computador, ela está composta por um conjunto de regras sintáticas e semânticas as quais permitem ao programador especificar as ações que devem ser tomadas de acordo a determinadas circunstâncias.

Os processadores de linguagem são todos aqueles aplicativos informáticos nos quais um dos dados fundamentais de entrada é uma linguagem. Algumas destas ferramentas de Software são:

- Tradutores (*translators*).

- Compiladores (*compilers*).
- Interpretadores (*interpreters*).
- Pre-processadores (*preprocessors*).
- Depuradores (*debuggers*).

2.1.1. Tradutores

Um tradutor é um programa que processa um texto fonte e gera um texto objeto, e pode ser de dois tipos: compilador ou interpretador. O tradutor está escrito numa *linguagem de implementação (LI),* o texto fonte está escrito em *linguagem fonte (LF),* e o texto objeto está escrito em *linguagem objeto (LO)* (Fig. 2.1).

A linguagem fonte (LF) é a linguagem origem, o qual é transformado pelo tradutor, por exemplo: C, C++, Pascal, BASIC, etc. Também podem ser linguagens de baixo nível.

A linguagem objeto (LO) é a linguagem à qual traduz-se o texto fonte. Estas podem ser, por exemplo, outra linguagem de alto nível, linguagem de máquina, linguagem *Assembler*, etc.

A linguagem de implementação (LI) é a linguagem na qual está escrito o tradutor. Pode ser qualquer linguagem, desde uma linguagem de alto nível até uma linguagem de máquina.

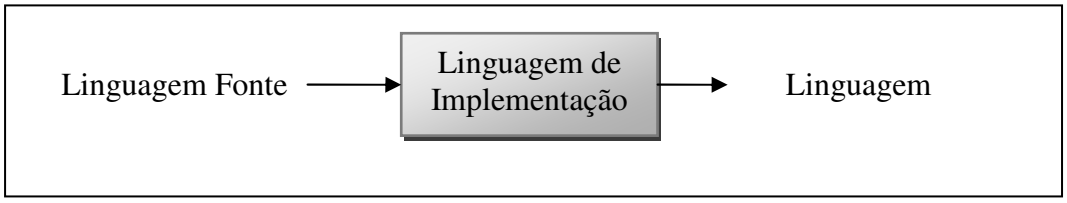


Fig. 2.1: Funcionamento de um Tradutor.

2.1.1.1. Estrutura geral de um tradutor

A construção e implementação de um tradutor é uma tarefa complexa, que pode ser reduzida aplicando uma metodologia, que consiste em dividir em módulos as distintas etapas do tradutor. A complexidade do tradutor depende das características da linguagem fonte e da linguagem objeto, e sua distância em nível. Por exemplo, é mais simples traduzir entre duas linguagens do mesmo nível, que entre uma linguagem de alto nível e uma de baixo nível.

Devido à grande variedade de sistemas computacionais desenvolvidos comercialmente, os tradutores tem que ser independentes da arquitetura do sistema sobre o qual foi desenvolvido, ou seja, eles têm que poder se executar em equipamentos computacionais distintos. Para fazer isto, o processo de tradução é feito em duas etapas denominadas *front – end (Analise)* e *back – end (síntese)*, a primeira analisa o programa fonte e a segunda é a encarregada de gerar o programa objeto, desta forma, para transportar o tradutor de uma maquina a outra, só tem que ser modificado o *back – end* (Fig. 2.2).

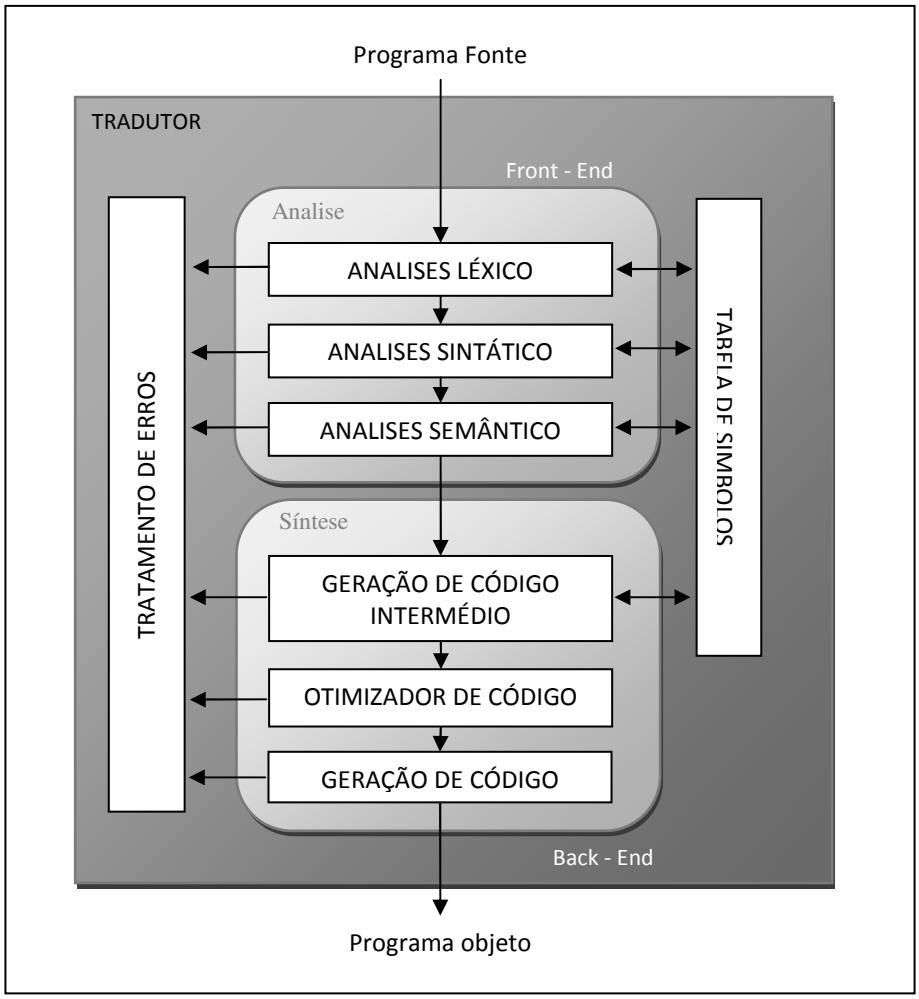


Fig. 2.2: Fases do Tradutor.

2.1.1.2. Etapa de Analise

A etapa de analise está composta por três processos: *Analise léxica, Analise sintática e Analise semântico*, os quais permitem verificar a escrita e a lógica do algoritmo processado.

Análise LÉXICA:

Um programa fonte está composto por uma serie de símbolos (letras, dígitos, caracteres especiais), os quais ao agrupar-se (*tokens ou unidades sintáticas)* representam variáveis, etiquetas, constantes, operadores, instruções, etc.

O analise léxico é a primeira etapa do tradutor, sua função é ler o programa fonte caractere a caractere, separando e identificando cada um dos *tokens* do programa fonte. Por outra parte a análise léxica também permite:

- Remover espaços em branco e saltos de linha.
- Remover os comentários do código fonte.
- Identificar erros no código.

Por exemplo, a análise na seguinte sentença em PASCAL seria (Tab. 2.1):

IF a > b + 1 THEN

TOKENS	DESCRIÇÃO
IF	Palavra reservada.
a	Identificador
>	Operador de Comparação
b	Identificador
+	Operador
1	Constante
THEN	Palavra reservada

Tab. 2.1:Exemplo do análises léxico numa expressão.

Analise SINTÁTICA:

A análise sintática é uma análise de sentenças e é mais complexa que a análise léxica. Sua função é analisar os *tokens* do programa fonte e comprovar que a estrutura das sentenças do

programa seja correta de acordo com certas regras gramaticais, em outras palavras, ela verifica que a posição dos *tokens* na sentença seja a correta, dependendo da gramática da linguagem.

Analise SEMÂNTICA:

Na análise semântica analisa se a sentença tem algum significado, porque pode existir o caso no qual uma sentença seja sintaticamente correta, mas não possa ser executada por carecer de sentido. Por exemplo, na seguinte expressão a variável B não foi definida previamente.

```
A = (B+5) * 3 ;
```

Nesse caso a expressão está sintaticamente correta, mas ela não poderá ser avaliada, já que a variável B não tem nenhum valor.

2.1.1.3. Etapa de Síntese

A etapa de síntese é a encarregada de gerar e otimizar o código objeto do algoritmo. Para isto esta etapa está dividida em três processos: Geração de código intermediário, Otimização de código e Geração de código.

Geração de código intermediário:

O código intermediário é um código abstrato independente da maquina para a qual será gerado o código objeto. O código intermediário tem que cumprir dois requisitos fundamentais: ser fácil de produzir e fácil de traduzir a linguagem objeto.

Otimizador de código:

Permite gerar um código mais compacto e eficiente, eliminando sentenças que nunca são executadas, simplificando expressões aritméticas, etc. Isto permite ter um código de máquina

mais rápido de executar. Esta fase de sínteses encontra-se em compiladores, e em alguns interpretadores (normalmente os interpretadores não otimizam o código objeto).

Geração de código:

Este processo é o encarregado de traduzir cada uma das instruções do código intermédio para posteriormente gerar o código objeto.

2.1.1.4. Tabelas de símbolos

A tabela de símbolos é uma estrutura de dados na qual está armazenada toda a informação relativa a cada identificador do código fonte, estes podem ser variáveis, funções, procedimentos etc.

Cada elemento da tabela de símbolos está composto pelo identificador e pelos seus atributos, os quais proporcionam a informação relativa a cada identificador. Alguns destes podem ser:

- **Nome do identificador.**
- **Especificação do identificador:** variável, tipo de dado, procedimento, função, etc.
- **Tipo:** por exemplo, no caso das variáveis podem ser encontrados atributos de tipo inteiro, real, String, etc.
- **Dimensão ou tamanho:** por exemplo, as dimensões de um *array.*
- etc.

Os atributos podem variar de uma linguagem a outra, dependendo das características próprias de cada linguagem e da metodologia de desenvolvimento do *tradutor.*

A tabela de símbolos é criada na fase de análise, na qual o *Analisador Léxico* proporciona a listagem de identificadores e o *Analisador Sintático* seus atributos. O *Analisador Semântico* pode também gerar um ou outro atributo.

O *Analisador Semântico* e o gerador de código obtêm da *Tabela De Símbolos* a informação necessária para realizar sua tarefa. As operações fundamentais feitas na *Tabela De Símbolos* são inserção e pesquisa de informação.

2.1.1.5. Tratamento de erros.

Os erros encontrados em cada uma das fases de analises são enviados ao modulo de tratamento de erros. Este módulo pode ser um subprograma que recebe o código do erro e o numero da linha onde este tem sido produzido, para posteriormente executar alguma ação.

No tratamento de erros é muito importante ter uma adequada sinalização do erro, já que isso permitirá ao programador localizar facilmente o foco dos problemas. Junto com o código do erro deve ser enviada uma mensagem com mínimo a seguinte informação:

- A posição onde o símbolo do erro foi encontrado.
- O símbolo de erro.
- A causa provável que pôde gerar o erro.

2.1.1.6. Compiladores e interpretadores.

Um compilador é um tradutor que mediante a compreensão da sintaxe e da gramática de uma linguagem de alto nível, produz um texto em linguagem maquina para posteriormente gerar um programa executável (Fig. 2.3).

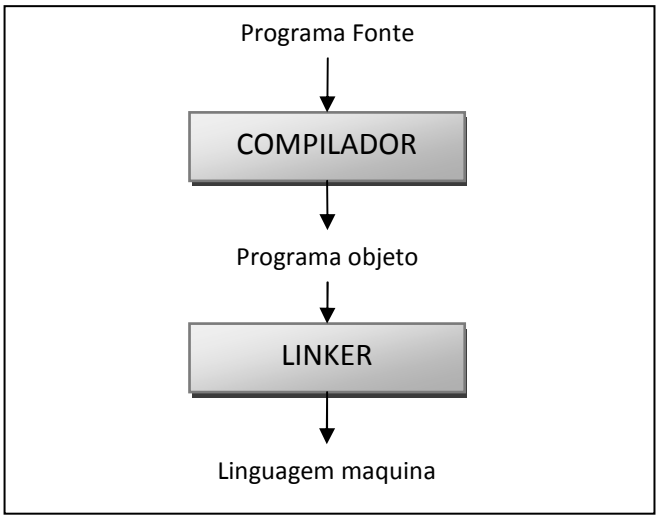


Fig. 2.3: Estrutura geral de um Compilador.

O *linker* ou ligador é um programa que extrai, do programa objeto (gerado nas primeiras fases do processo de compilação), a informação de todos os recursos necessários (bibliotecas), tirando os recursos que não precisa. Posteriormente ele liga o código objeto com suas bibliotecas produzindo assim, um arquivo executável (EXE).

Isto faz que o funcionamento de um programa compilado só possa ser visto depois de gerar e executar o arquivo executável (EXE).

Por outra parte, um interpretador é um tradutor que em vez de gerar código (linguagem maquina) para uma arquitetura em particular, simplesmente lê o código tal e como está escrito e imediatamente o converte em ações, ou seja, o executa nesse mesmo instante. Esta característica resolve o problema de transportabilidade, o que permite que uma linguagem interpretada possa ser executada sobre diversos equipamentos (Fig. 2.4).

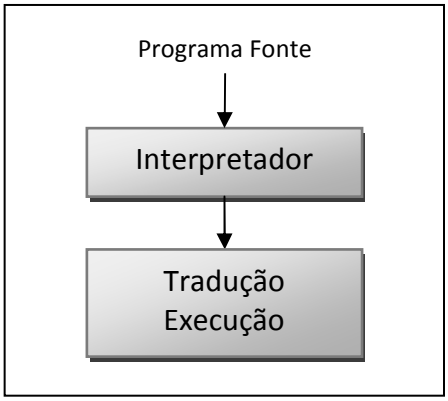


Fig. 2.4: Estrutura geral de um Interpretador.

Outras das características que fazem que os interpretadores sejam mais versáteis tem que ver com o tratamento de erros, já que alguns erros (como dividir um número entre zero), só podem ser detectados em tempo de execução, pelo qual sua solução é mais fácil nestes. No caso dos compiladores, para solucionar um erro destas características é necessário corrigir o problema e compilar novamente todo o programa, ou seja, fazer todo o processo de geração e compilação de código de novo, e posteriormente executar o programa para assim confirmar que o problema foi corrigido. Isto faz que estes tipos de erros sejam de mais fácil solução nos interpretadores.

Até pouco tempo atrás, as linguagens compiladas como *Visual Basic*, tinham uma maior preferência no mercado, já que têm um menor tempo de execução que as linguagens interpretadas, porém com o avance da tecnologia no campo dos processadores, estas diferencias de velocidade não são tão significativas, linguagens interpretadas como a *JAVA, Python* são muito usadas hoje em dia.

Vantagens dos interpretadores frente aos compiladores

- O programa pode ser executado imediatamente sem a necessidade de esperar a que este seja compilado.

- O programa pode ser interrompido com facilidade.
- O programa pode ser rapidamente modificado e executado novamente.
- São muitos apropriados na fase de desenvolvimento de um programa, já que a compilação torna difícil a execução passo a passo do programa, impedindo a edição durante o seguimento e depuração do programa.

Desvantagens dos interpretadores frente aos compiladores

- A execução é mais lenta, já que cada instrução tem que ser traduzida tantas vezes como seja executada.
- Diferente da interpretação, na compilação uma vez o programa é compilado este pode ser executado n-vezes sem a necessidade de compilá-lo de novo.

2.1.1.7. Interpretadores puros.

Os interpretadores puros são aqueles programas que analisam uma sentença e a executam, e assim sucessivamente todo o programa fonte. Estes interpretadores foram desenvolvidos na primeira geração de computadores, estes permitiam a execução de programas compridos com computadores de memória muito reduzida, já que só necessitavam conter na memória, o interpretador e a sentença a analisar e executar. O principal problema deste tipo de interpretadores é que se na metade do programa fonte é gerado algum erro, o processo tem que começar de novo. Na continuação é apresentada o diagrama geral de um interpretador puro (Fig. 2.5).

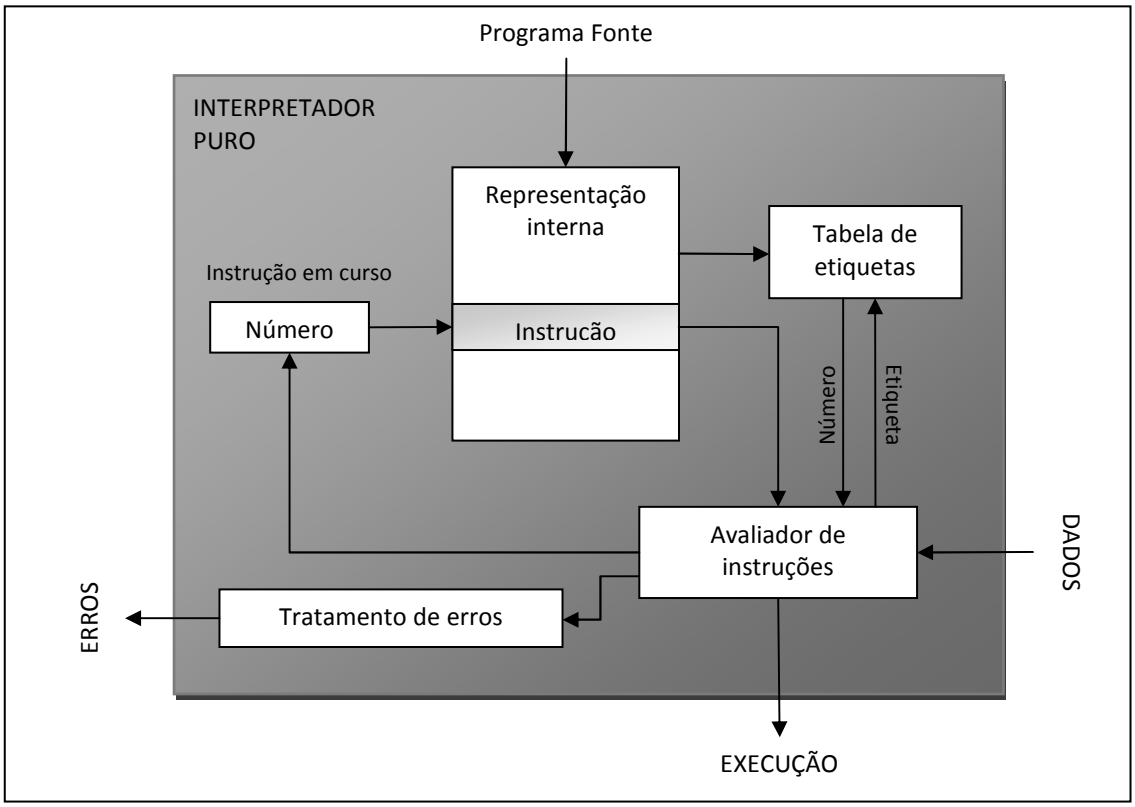


Fig. 2.5: Fases do Interpretador Puro

Representação interna: A representação interna tem todas as instruções do programa fonte, elas são armazenas e numeradas numa estrutura de tamanho fixo (*array*, arquivo binário, etc.) na memória ou em disco.

Tabela de etiquetas: Ela contém todas as etiquetas e sua posição no programa fonte, estas são utilizadas por instruções de salto e em chamadas a procedimentos e funções.

Avaliador de instruções: Ele é o encarregado de executar cada uma das instruções do programa fonte, e de enviar no caso de serem necessárias, as mensagens de erro.

2.1.1.8. Interpretadores avançados.

Os interpretadores avançados são aqueles que incorporam um passo prévio de analises de todo o programa fonte, gerando posteriormente uma linguagem intermédia a qual é executada por ele mesmo. Isto faz que os erros no léxico e na sintaxe sejam identificados na etapa de analises e, por conseguinte não passem desta. Na continuação é apresentado o diagrama geral de um interpretador avançado (Fig. 2.6).

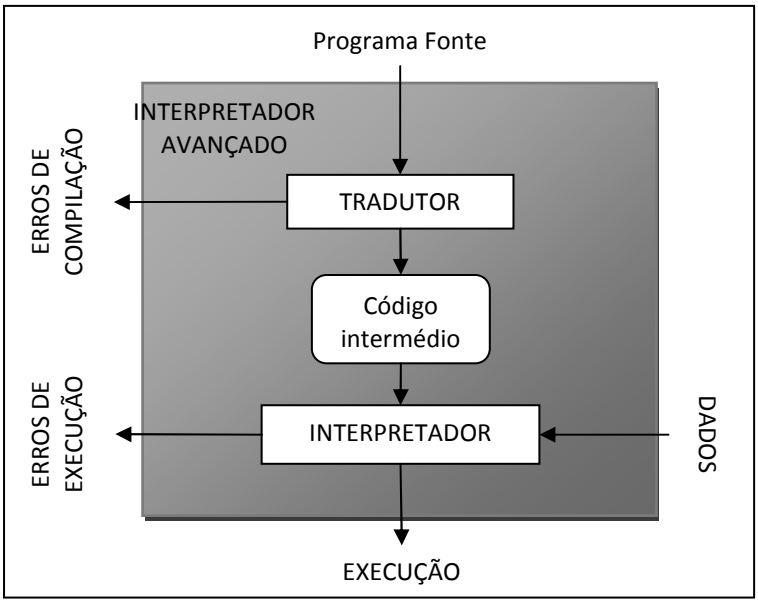


Fig. 2.6: Fases do Interpretador Avançado.

Um exemplo de interpretador avançado é o que utiliza a linguagem JAVA. Assim por exemplo, quando um programa em JAVA é compilado são gerados um ou vários arquivos (.CLASS), os quais tem o formato binário denominado *bytecode*, este formato é independente da plataforma o que permite que este seja posteriormente interpretado de acordo à plataforma na qual vai ser executado.

2.1.2. Preprocessadores

Um preprocessador é um programa que recebe um programa fonte e faz conversões léxicas nele. Ele pode remover comentários, incluir outros arquivos e executar substituições de macros. Por exemplo, no caso da linguagem em C, o preprocessador pode fazer as seguintes alterações (Fig. 2.7):

```
#include <stdio.h>
#define novo 0
int main (void)
{
    /*
    Exemplo do programa "oi mundo"
   */
    printf("Oi, Mundo!!!\n");
    return novo;
```

Fig. 2.7: Exemplo de preprocessador em C.

- Troca o texto entre os caracteres "/*" e "*/" por espaços em branco.
- Troca o texto novo por 0.
- Troca a linha *#include <stdio.h>* pelo ficheiro com esse nome, o que permite utilizar a rotina *printf().*

2.1.3. Depuradores

Os depuradores são ferramentas que permitem encontrar e corrigir os erros nos programas. Estas ferramentas normalmente estão implementadas nos compiladores de tal forma que permite ao programador comprovar e visualizar a correta execução do programa.

Um depurador:

- Permite observar o valor de qualquer variável, endereço ou expressão do programa fonte.
- Permite comprovar o código objeto gerado por cada instrução do programa fonte.
- Permite observar o programa executável em baixo nível, visualizando os valores das distintas posições de memória, registros, etc.

2.2. Leitores de Tela

Nos últimos anos, a computação tem tido uma grande transformação, já que esta deixa de ser uma tecnologia exclusiva de programadores e analistas de sistemas, para se converter numa ferramenta acessível a todo o mundo. Devido a isto, as grandes empresas que desenvolvem aplicativos de software invertem grandes quantidades de recursos para que seus sistemas sejam simples e fáceis de operar, permitindo que estes possam ser utilizados por quase qualquer perfil de usuário.

Uma das ferramentas computacionais que permitem a inclusão digital de pessoas com deficiência visual são os leitores de tela. Um leitor de tela é um aplicativo de software que identifica e interpreta o que é visualizado na tela do computador. Esta interpretação é apresentada ao usuário mediante sintetizadores de texto a voz, ícones sonoros ou uma saída *Braille.*

Os leitores de tela são uma tecnologia assistida muito útil para pessoas com algum grau de deficiência visual ou com dificuldades de aprendizagem. Normalmente estes tipos de aplicativos são combinados com outros tipos de tecnologias assistidas tais como ampliadores de tela, etc.

Atualmente o leitor de tela mais reconhecido e utilizado por pessoas com algum grau de diminuição visual e o JAWS, ele é um leitor de tela com síntese de voz, o qual permite lhe ao usuário com limitação visual interagir com todas as áreas que aparecem na tela e aceder a suas aplicações só com o uso do teclado, mediante comandos especiais.

Este aplicativo é comercializado pela empresa *Freedom Scientific,* e tem um preço aproximado de U$ 1300 (mil trezentos dólares) pelo qual não é accessível a todo o mundo. Sem embargo na atualidade existem outras ferramentas desenvolvidas especificamente para pessoas com deficiência visual e que são de livre distribuição, uma destas é o DOSVOX, a qual será apresentada mais adiante.

Capitulo 3

3.Entorno de execução do MATVOX

O objetivo principal na criação do aplicativo matemático MATVOX, é ajudar a pessoas com deficiência visual em o desenvolvimento de algoritmos matemáticos, por este motivo o MATVOX utiliza como entorno de programação um editor de textos gratuito conhecido como EDIVOX, o qual foi acrescentado com características próprias do aplicativo tais como: menus interativos, opções novas de fala, comandos de execução, entre outras.

Este editor faz parte de um software de livre distribuição conhecido como DOSVOX, o qual foi desenvolvido pelo "Núcleo de Computação Eletrônica" (NCE) da "Universidade Federal de Rio do Janeiro" (URFJ) sob orientação do Professor José Antonio Borges.

A vantagem principal na utilização de este editor de textos como plataforma de desenvolvimento é que o desenvolvedor não precisa de nenhum software externo diferente ao EDIVOX, para criar e executar seus algoritmos, já que todo o que ele precisa foi implementado no editor.

3.1. Sistema DOSVOX

O DOSVOX é um sistema computacional para microcomputadores da linha PC, o qual trabalha no sistema operacional "Microsoft Windows". Ele facilita a comunicação Homem – Maquina (HC) mediante a síntese de voz, isto faz que pessoas com algum grau de deficiência visual consigam interatuar com um computador de uma forma fácil e natural, permitindo assim,

um alto grau de independência no estudo e no trabalho. A principal diferença entre o DOSVOX e outros sistemas leitores de tela, é que a comunicação HC é mais simples e amigável, já que ele da um alto grau de importância as limitações físicas e características especiais de este perfil de usuário.

Este software comunica se com o usuário mediante a síntese de voz, mas ele não atua como um simples leitor de tela, senão que oferece um pacote com mais de 70 aplicativos falados que permitem estabelecer um dialogo amigável com o usuário, os quais a sua vez incorporam características próprias que permitem ter uma interação mais confortável e natural entre o usuário e o computador. A maior parte do diálogo entre o programa e o usuário é feito por voz humana gravada, o qual faz que o diálogo HC seja o mais confortável e o menos estressante possível.

A utilização do DOSVOX é bastante simples, já que um dos objetivos dos desenvolvedores foi reduzir ao máximo o comprometimento técnico por parte dos usuários, para assim evitar a necessidade de conhecimentos prévios do sistema.

O sistema está composto principalmente por:

- Sistema operacional que contém os elementos de interface com o usuário.
- Sistema de síntese de fala para língua portuguesa.
- Editor, leitor e impressor/formatador de textos.
- Impressor/formatador para Braille.
- Aplicações para uso geral, caderno de telefones, agenda, calculadora, preenchimento de cheques, etc.
- Diversos programas de uso geral para o cego.
- Jogos de caráter didático e lúdico.
- Utilitários de internet: FTP, acesso a WWW, um ambiente de "chat", um editor html, etc.
- Programas multimídia, como o processador multimídia (áudio midi CD), gravador de som, controlador de volumes, etc.
- Programas dirigidos à educação de crianças com deficiência visual.
- Um sistema genérico de telemarketing, dirigido a profissionais desta área.

- Ampliador de tela para pessoas com visão reduzida.
- Leitor para Windows.

3.1.1. Acionamento do DOSVOX

Uma vez que o computador é ligado e o Windows é carregado completamente, o DOSVOX pode ser executado pressionando simultaneamente as teclas Ctrl + Alt + D, feito isto é sintetizada a frase "DOSVOX - O que você deseja", a qual será ouvida sempre que o sistema necessite de alguma ação (Fig. 3.1).

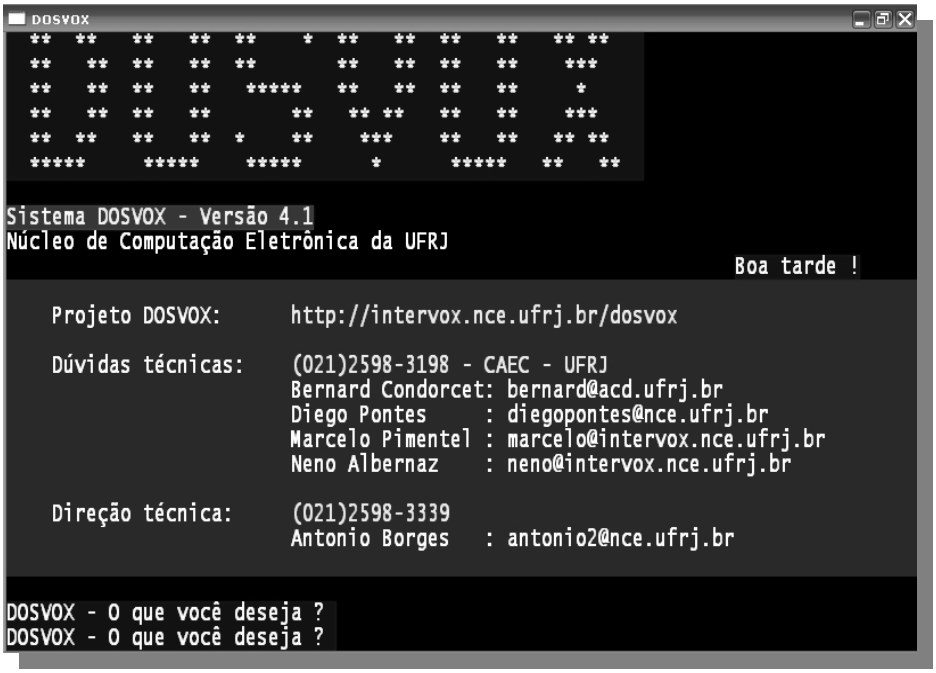


Fig. 3.1: Tela principal do DOSVOX

Para conhecer as opções do DOSVOX só é necessário utilizar a tecla F1, esta tecla está disponível na maior parte dos utilitários do sistema como uma tecla de ajuda (Fig. 3.2). Depois só

basta com pressionar uma tecla e a função correspondente é iniciada. A tecla ESC pode ser utilizada em qualquer momento para cancelar qualquer função.

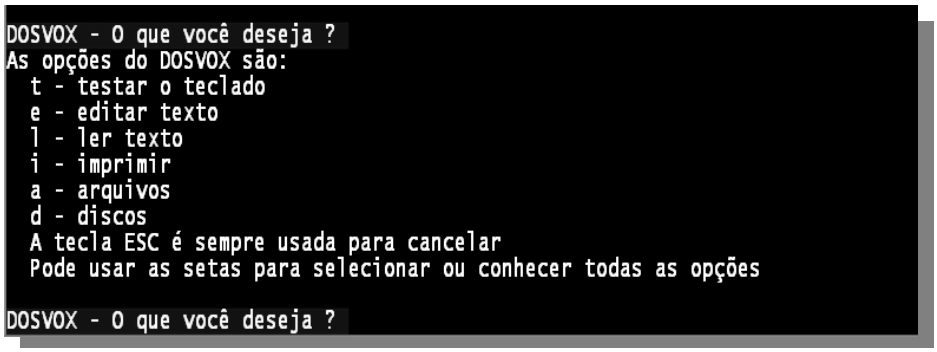


Fig. 3.2: Opções principais do DOSVOX

Opcionalmente podem ser usadas as setas e as opções são apresentadas num menu *pop-up* na tela do programa (Fig. 3.3). Para navegar nestes menus são utilizadas as setas (seta para arriba e para abaixo). Para escolher a opção desejada é utilizada a tecla ENTER.

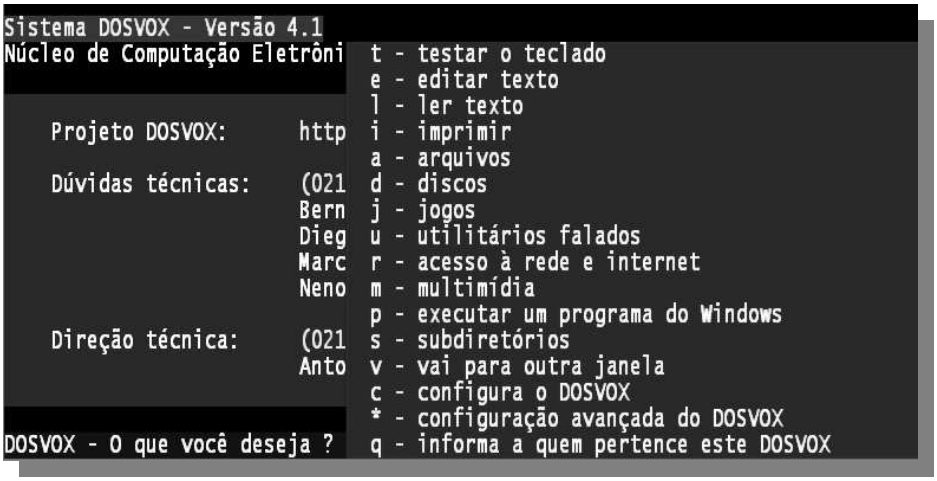


Fig. 3.3: Menu Pop-up de opções no DOSVOX

3.1.1.1. Testar teclado

A opção *Testar teclado* permite ao usuário iniciante nos microcomputadores, reconhecer a posição das diferentes teclas do teclado, facilitando desta maneira à aprendizagem dos diferentes aplicativos do sistema (Fig. 3.4).

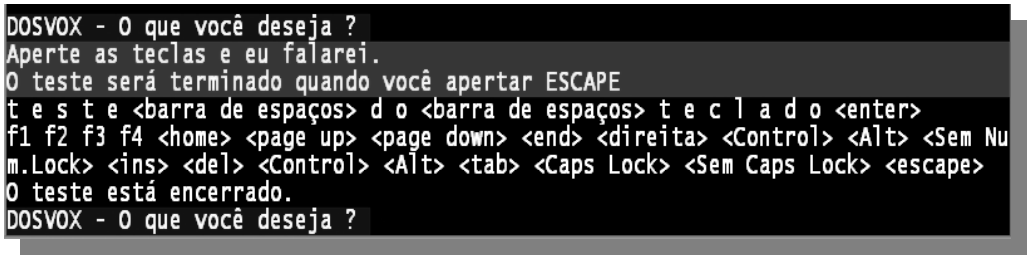


Fig. 3.4: Ferramenta para testar o teclado no DOSVOX

3.1.1.2. Manipulação de arquivos.

Quando a letra A é pressionada (após da pergunta "DOSVOX – O que você deseja?", é informado pelo sistema, o numero de arquivos no diretório, e posteriormente apresenta na tela, um menu *pop-up* com a listagem dos arquivos do diretório. Para conhecer as diferentes funções de manipulação de arquivos só é necessário pressionar a tecla F1 (Fig. 3.5).

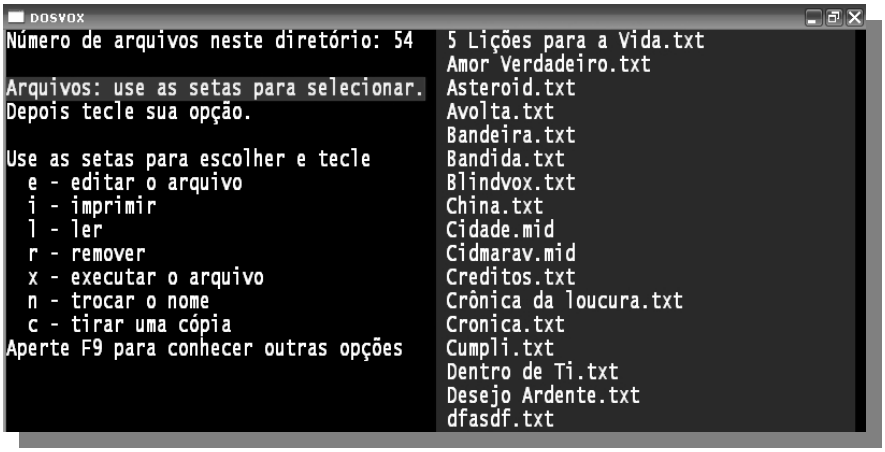


Fig. 3.5: Opções de manipulação de arquivo no DOSVOX

3.1.1.3. Jogos

O DOSVOX implementa alguns jogos os quais visam não somente ao entretenimento do usuário, senão a facilitar a aprendizagem do ambiente do DOSVOX, já que enquanto ele joga, pode ao mesmo tempo aperfeiçoar sua interação com o sistema. O menu de jogos é acionado pressionando a opção J seguido das setas do teclado (Fig. 3.6).

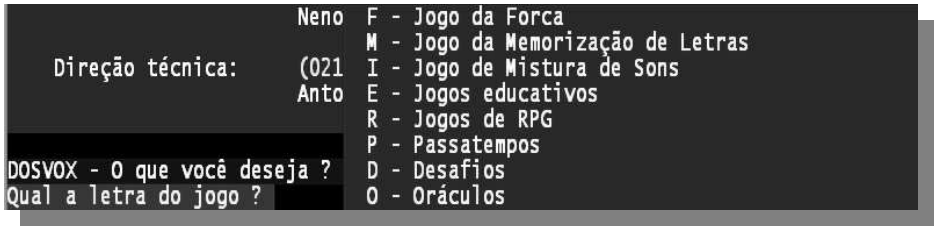


Fig. 3.6: Jogos do DOSVOX

3.1.1.4. Utilitários de uso geral

Estes utilitários permitem ao usuário realizar tarefas cotidianas proporcionando assim uma maior independência e organização. O menu de utilitários é acionado pressionando a opção U seguido das setas do teclado (Fig. 3.7).

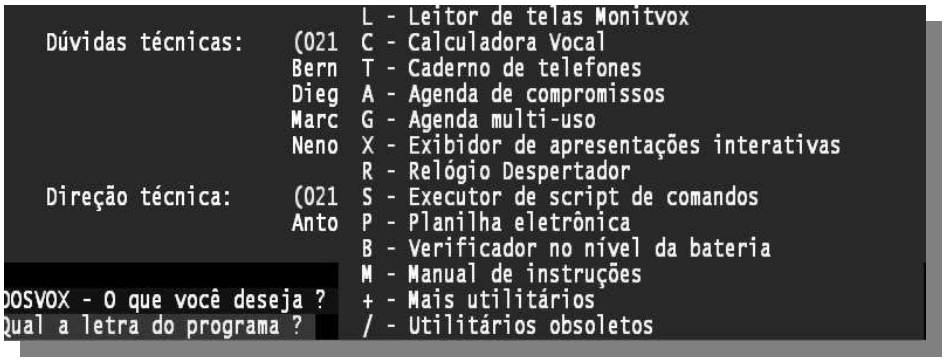


Fig. 3.7: Utilitários de uso geral do DOSVOX

Um dos utilitários encontrados neste menu é a calculadora vocal, a qual permite a execução de cálculos simples de forma completamente sonorizada (Fig. 3.8).

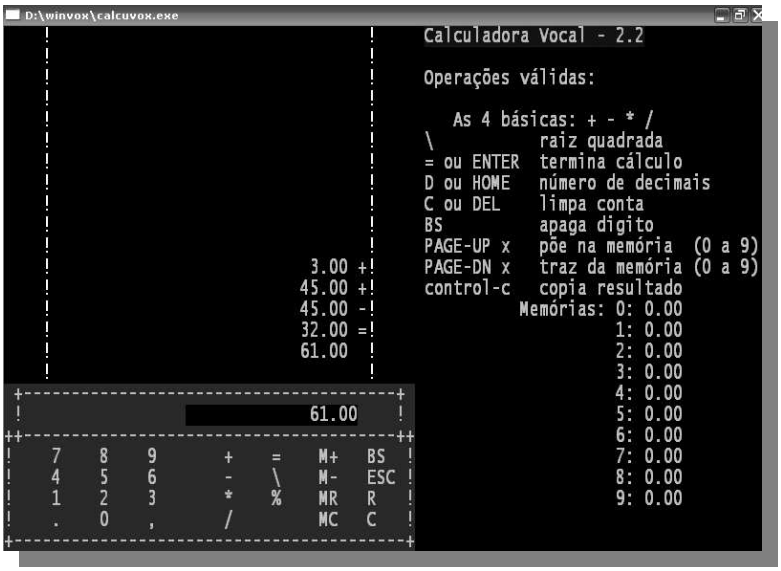


Fig. 3.8: Calculadora do DOSVOX

3.1.1.5. Multimídia

Devido a que as pessoas cegas têm uma relação muito importante com som, o DOSVOX proporciona diversos aplicativos para o processamento multimídia. Eles são apresentados acionando pressionando a opção M seguido das setas do teclado (Fig. 3.9).

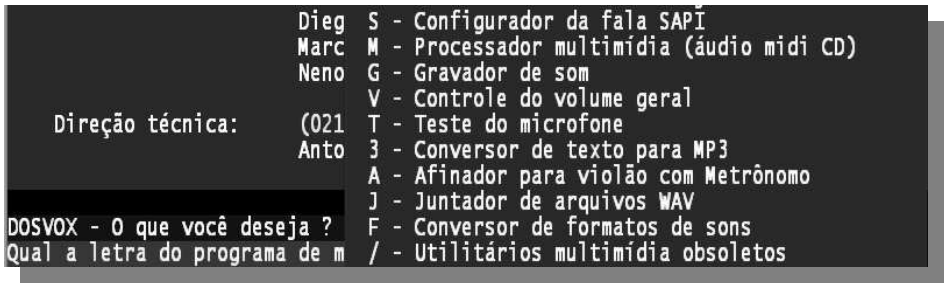


Fig. 3.9: Utilitários multimídia do DOSVOX

3.2. Editor de textos EDIVOX

O EDIVOX é um editor de textos que foi desenvolvido de acordo às necessidades especificas que possa chegar a ter um deficiente visual, este software possui comandos e funções que fazem possível que qualquer pessoa com alguma deficiência visual consiga produzir um texto de forma confortável e amigável e com características tais como: correção de trechos, correção ortográfica, edição de partes de texto, etc. as quais não são encontradas em outros métodos de escritura, como a escritura táctil e a maquina de escrever.

Na atualidade existem diversos leitores de tela, os quais permitem a deficientes visuais utilizar e interatuar com programas como “Microsoft Word” ou “Microsoft WordPad”, entre outros, sem embargo nenhum leitor destes oferece uma interface tão amigável com o usuário como EDIVOX.

Para aceder ao EDIVOX é pressionada a tecla E na tela principal do DOSVOX (Fig. 3.10).

Fig. 3.10: Tela principal do EDIVOX

Quando a tela principal do EDIVOX é apresentada, o programa vai perguntar pelo nome do arquivo, neste ponto o usuário tem duas opções: criar um arquivo novo, ou carregar um já existente. Uma vez executada alguma das duas opções o programa fica pronto para ler e editar arquivos tipo texto.

Por outra parte os comandos e controles disponíveis no EDIVOX (Copiar texto, cortar texto, etc.) são muito similares aos comandos encontrados em outros editores de texto comerciais tais como: Microsoft Word, Microsoft WordPad, etc. pelo qual é muito simples acostumar-se a utilizar este editor de texto.

3.2.1. Comandos principais no EDIVOX

O processo de criação e edição de texto no EDIVOX é muito similar ao processo realizado nos editores convencionais, sem embargo EDIVOX incorpora características sonoras, especiais para pessoas com deficiência visual. Estas características permitem ao usuário, entre outras coisas, reproduzir sonoramente todo o que é escrito no editor.

As principais ações que o usuário pode executar com as diferentes teclas do teclado são a seguintes:

Movimento do cursor:

Seta para ESQUERDA: Move o cursor um caractere para a esquerda. Ao chegar ao primeiro caractere da linha, o programa dá um bip. Por outra parte o caractere, pelo qual o cursor passou será falado.

Seta para DIREITA: Move o cursor um caractere para a direita. Ao chegar ao ultimo caractere da linha, o programa dá um bip. Por outra parte o caractere, pelo qual o cursor passou será falado.

Seta para CIMA: Move o cursor uma linha para cima. Ao chegar à primeira linha, o programa fala "*Inicio do texto*". Por outra parte o texto da linha à qual o cursor chega será falada e o cursor fica posicionado na coluna 1.

Seta para BAIXO: Move o cursor uma linha para baixo. Ao chegar a ultima linha, o programa fala "*Fim do texto*". Por outra parte o texto da linha à qual o cursor chega será falada e o cursor fica posicionado na coluna 1.

HOME: Posiciona o cursor na coluna 1 da linha.

END: Posiciona o cursor após da ultima coluna escrita da linha.

ENTER: Se o cursor está na coluna 1, ela insere uma nova linha embaixo da linha atual e fala "*Linha nova*", e o cursor fica na coluna 1. Por outra parte, se ao pressionar a tecla ENTER o

cursor está numa coluna diferente à coluna 1, a linha não é quebrada e o cursor é movido para a linha de embaixo, ficando na coluna 1.

PAGE UP: Volta 15 linhas de texto, o cursor fica na coluna 1.

PAGE DOWN: Avança 15 linhas de texto, o cursor fica na coluna 1.

Remoção de caracteres indesejáveis:

BACKSPACE: Remove o caractere a esquerda do cursor.

DEL: Remove o caractere na posição do cursor.

CTRL + Y ou F7: Remove a linha completa na qual está o cursor, e fala "*Linha removida*".

CTRL + BACKSPACE ou CTRL + H: Recupera a ultima linha apagada e fala "*Linha nova*".

CTRL + D: Apaga toda uma palavra, e soletra a palavra apagada.

CTRL + S: Apaga do inicio da linha até a posição do cursor e fala "*Apagado a esquerda*".

Leitura das linhas:

F1: Pressionando a tecla F1 várias vezes cada palavra da linha atual é falada, depois de ler a ultima palavra da linha soa um bip.

CTRL + F1: Fala a linha inteira a partir do ponto onde o cursor está. Para interromper a leitura pode ser pressionada qualquer tecla exceto as teclas CTRL, ALT, ou WINDOWS**.**

ALT + F1: Fala o resto do texto a partir do ponto onde o cursor está. Para interromper a leitura pode ser pressionada qualquer tecla exceto CTRL, ALT, ou WINDOWS.

F4: Ativa ou desativa a fala durante a digitação, as falas dos comandos continuam ativas.

Inserção de novas linhas:

ENTER: Se o cursor está na coluna 1, ela insere uma nova linha embaixo da linha atual e fala "*Linha nova*", e o cursor fica na coluna 1.

CTRL + ENTER ou CTRL + J ou CTRL + M: Insere uma linha nova em cima da linha onde o cursor está, e fala "*Linha nova*".

INSERT: Ativa o desativa a possibilidade de inserir linhas com a tecla ENTER, e fala "*Enter vai inserir linha*" ou "*Enter não vai inserir linha*" conforme seja o caso.

CTRL + Q: Atua como um ENTER, mas permite quebrar uma linha em duas partes, fala "*Linha quebrada*".

Configuração das margens:

F10: Por defeito as margens do EDIVOX estão na coluna 1 e na coluna 72, mais elas podem ser mudadas com a tecla F10, ao pressionar F10 o EDIVOX vai falar "*Digite coluna da margem esquerda:*", nesse ponto o usuário deve teclar o número da coluna e pressionar ENTER, logo vai falar "*Digite coluna da margem direita*" e o usuário deve teclar o número da coluna e depois pressionar ENTER, depois o programa vai falar "*Margem adicionada*".

Capitulo 4

4. Aplicativo Matemático MATVOX

O sistema MATVOX é um interpretador de algoritmos matemáticos de uso para deficientes visuais, ele foi criado com o propósito de permitir e lhe facilitar a pessoas com necessidades especiais o desenvolvimento de trabalhos e pesquisas nas áreas das ciências exatas.

O MATVOX foi desenhado de tal maneira que permite ao usuário criar algoritmos e cálculos matemáticos direitamente desde um editor de textos conhecido como EDIVOX, o qual foi desenvolvido especialmente para pessoas com deficiência visual, o que permite aproveitar as características e os recursos de sínteses de fala com os que este conta, facilitando em grande medida o processo de criação, manipulação e execução de algoritmos.

4.1. Interface do MATVOX

O MATVOX não utiliza uma interface própria, já que ele foi desenvolvido para ser executado direitamente desde o editor de textos (EDIVOX), pelo qual a interface utilizada por este é praticamente a mesma que a utilizada pelo EDIVOX, o qual permite ao MATVOX aproveitar todas as características implementadas no EDIVOX.

O EDIVOX tem uma interface baseada nas características próprias dos deficientes visuais, a qual é lograda graças à ergonomia de comandos e aos feedbacks sonoros com os que este conta. Este editor de texto possui grande parte dos comandos encontrados nos principais editores de texto comerciais, como o Microsoft Word, o WordPad, o Bloc de notas, etc., e a sua vez mediante os feedbacks sonoros facilita a orientação do deficiente na interface deste.

Para utilizar os diferentes comandos do editor são utilizadas combinações de teclas as quais permitem executar diferentes ações tais como: copiar textos, colar textos, configurar margens, salvar arquivos, entre muitas outras.

O *feedback* sonoro é a principal característica deste editor de textos, já que este permite ao deficiente ouvir e identificar mediante mensagens falados, cada uma das ações que estão sendo executadas no editor. Os *feedbacks* podem ser de dois tipos: mensagens pré gravadas ou voz sintetizada (Fig. 4.1).

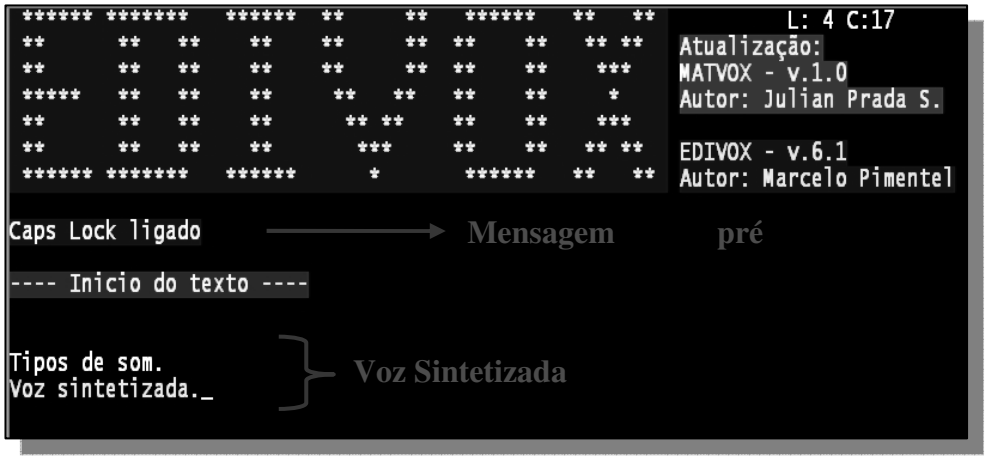


Fig. 4.1: Fala no EDIVOX

As mensagens pré gravadas são arquivos sonoros que estão armazenados no sistema, os quais são estáticos e sempre são ouvidos da mesma maneira, por exemplo: quando o usuário executa o EDIVOX, inicialmente ele sempre vai escutar a seguinte mensagem *"EDIVOX, Qual é o nome do arquivo?"*, este tipo de mensagem é estático pelo qual sempre vai ser o mesmo, porém posteriormente quando o usuário for escolher o nome do arquivo, o *feedback* gerado vai ser dinâmico, pelo qual não pode ser utilizada uma mensagem pré gravada, para isto o EDIVOX utiliza um sintetizador de voz, o qual simplesmente converte o texto em voz sintetizada (Fig. 4.2).

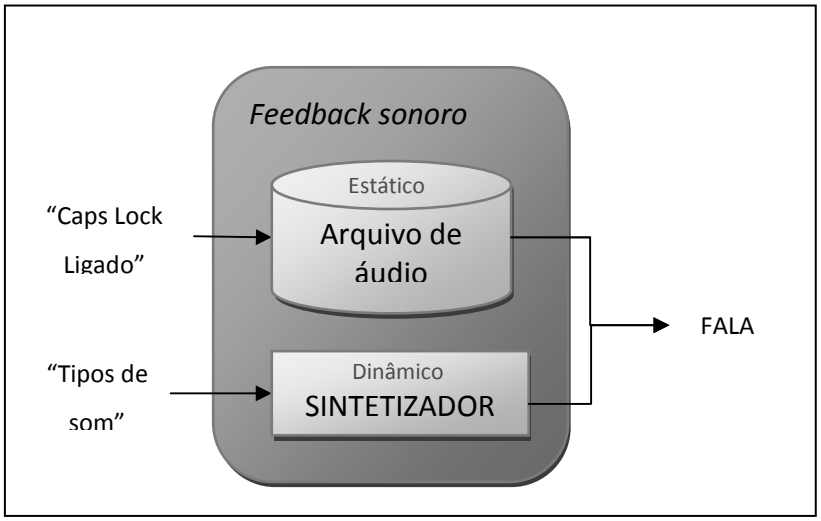


Fig. 4.2: Feedback sonoro no EDIVOX

O *feedback* dinâmico é utilizado principalmente na leitura de textos. Por exemplo, quando é escrito algum texto, as letras digitadas são faladas uma por uma e, por conseguinte são ouvidos arquivos de som pré gravados, sem embargo quando o texto é percorrido mediante as setas, o texto é lido utilizando o sintetizador de voz, pelo qual não será ouvida cada uma das letras por separado, senão que será ouvida a palavra completa.

Estes dois tipos de *feedbacks* tem suas vantagens e desvantagens, os *feedbacks* pré gravados permitem ter uma interação Homem – Computador, mais agradável e natural, sem embargo estes *feedbacks* dependem dos arquivos de som gravados previamente no sistema, o qual dificulta a inserção de novas mensagens no programa. Por outra parte os sintetizadores utilizados pelos *feedbacks* dinâmicos permitem sintetizar qualquer tipo de texto, pelo qual não é necessário ter armazenado no sistema arquivos específicos de som. Eles dependem exclusivamente do sintetizador de voz instalado no computador, o qual facilita o processo de atualização do mesmo, sem embargo atualmente som poucos os sintetizadores que conseguem sintetizar o texto com uma qualidade similar à das mensagens pré gravadas, o que faz que a interação Homem – Computador seja mais cansativo e menos natural.

Todas as características mencionadas anteriormente foram aproveitadas pelo MATVOX, e a sua vez foram desenvolvidas novas interfaces com características exclusivas deste.

4.2. Menus interativos do MATVOX

Devido à necessidade de desenvolver um aplicativo o mais intuitivo possível para o deficiente visual, o MATVOX implementa uma serie de menus interativos os quais permitem ao usuário interatuar dinamicamente com a calculadora. Neles estão definidas todas as opções da calculadora, tais como: comandos, funções, constantes, etc. as quais são acionadas mediante um controle muito simples (Fig. 4.3).

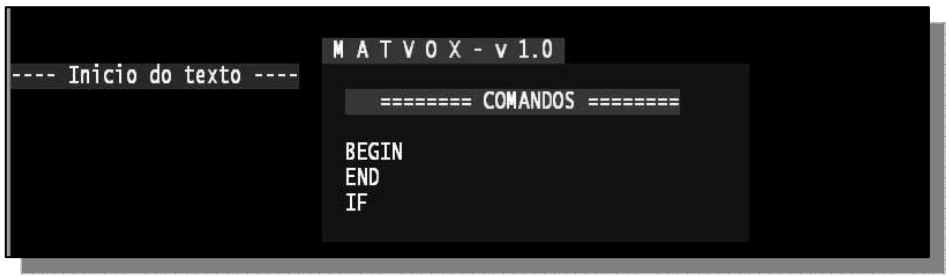


Fig. 4.3: Menus iterativos (Pop-up) do MATVOX

Estes menus facilitam em grande parte o processo de criação de algoritmos matemáticos já que eles estão interatuando e orientando em todo momento ao usuário mediante mensagens sonoras, isto permite que o usuário consiga inserir facilmente comandos e funções ao algoritmo, sem a necessidade de conhecer previamente a sintaxes destes.

Por exemplo, se o usuário vai inserir o seguinte comando: ***"\ IF { A > B } ;"*** ele não precisa conhecer previamente a sintaxes deste, simplesmente ele pode procurar e escolher no menu da calculadora o comando "*IF*", feito isto a calculadora vai perguntar pela primeira

expressão de comparação, pelo operador de comparação e pela segunda expressão de comparação e posteriormente vai inserir todo o comando no EDIVOX com a sintaxe adequada (Fig. 4.4).

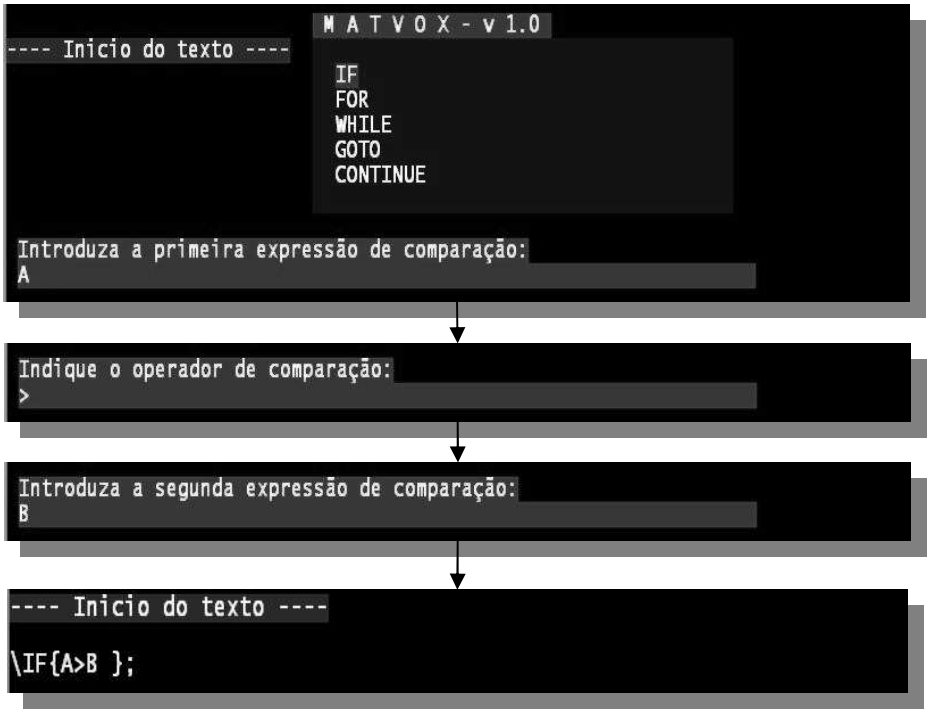


Fig. 4.4: Exemplo do uso dos menus iterativos do MATVOX, inserção iterativa de instruções

Este tipo de menus ademais de ajudar ao usuário no uso dos diferentes comandos da calculadora, evita em grande parte a geração de erros pela semântica e sintaxes destes. Por outra parte estes menus incluem mensagens de ajuda pré gravados com a descrição de cada uma das opções do menu, o que permite ao usuário identificar e conhecer a descrição de cada uma destas.

4.3. Fala detalhada de sentenças

Com o objetivo de facilitar o processo de desenvolvimento de algoritmos foi criado um modo de fala, o qual faz que o processo de leitura das linhas do algoritmo seja mais amigável e menos cansativo para o deficiente visual. Este modo de fala permite identificar facilmente possíveis erros nas linhas de comando sem a necessidade de fazer este processo mentalmente.

Por exemplo, se tem se a seguinte linha de comando: ***"\ IF { A > B } ;"*** , normalmente o EDIVOX falaria "*Barra invertida, IF, abre chave, a, maior que, b, fecha chave",* sem embargo com este modo de fala o EDIVOX vai falar *"Comando IF, o primeiro parâmetro de comparação é: A, o operador de comparação é: operador MAIOR, o segundo parâmetro de comparação é: B",* por outra parte, se o comando tiver algum erro, por exemplo se fosse: ***"\ IF { A } ;"*** o programa falará: *"Comando IF, o primeiro parâmetro de comparação é: A, o operador de comparação do comando IF não está definido ou não é válido, o segundo parâmetro de comparação do comando IF não está definido ou não é válido",* como pode se observar este modo de fala permite descrever detalhadamente cada uma das linhas de comando do algoritmo, o que permite mentalizar mais facilmente o algoritmo e assim facilitar o desenvolvimento destes.

4.4. Especificações do MATVOX

A linguagem criada no MATVOX está baseada nas características e sentenças básicas das principais linguagens de programação tais como: PASCAL, C, BASIC etc. as quais implementam características próprias, orientadas a facilitar o uso destas a pessoas com deficiência visual. Por outra parte a calculadora inclui características adicionais tais como: funções matemáticas, constantes físicas, conversões de unidades, espaços de memória para armazenar dados, entre outras, o que permite ao usuário criar algoritmos matemáticos de altas prestações. A continuação serão apresentadas as especificações funcionais da calculadora.

4.4.1. Tipo de Variáveis.

O MATVOX permite utilizar basicamente dois tipos de variáveis: variáveis numéricas e variáveis de tipo texto ou *STRING.* Estas variáveis não precisam ser declaradas, o qual facilita e simplifica o processo de criação de algoritmos.

As variáveis numéricas abarcam todos os números reais e permitem definir valores em formato de notação cientifica. Por exemplo: A = 34 E-5. Por outra parte o MATVOX implementa alguns tipos de variáveis as quais permitem agrupar dados de um mesmo tipo (dados numéricos ou dados tipo *STRING),* em uma ou duas dimensões, pelo que são muito úteis para trabalhar com grandes quantidades de dados.

Todas estas variáveis podem se definir de duas maneiras: estática ou dinamicamente. Quando são definidas de forma estática, elas ficam com um valor fixo durante todo o algoritmo, quando definem se como dinâmicas, o programa vai perguntar pelo valor da variável, o que permite criar um programa dinâmico entre o computador e o usuário.

Todos estes tipos de variáveis têm uma notação especial para sua definição a qual será explicada mais adiante.

4.4.2. Operadores Aritméticos e Relacionais

O MATVOX implementa os quatro operadores aritméticos básicos: *SOMA, RESTA, MULTIPLICAÇÃO e DIVISÃO*. Para fazer outros tipos de operações têm que ser utilizadas funções, as quais serão apresentadas mais adiante.

Os operadores relacionais permitem fazer operações de comparação entre expressões, este tipo de operações são utilizadas para avaliar condições as quais são utilizadas por alguns tipos de instruções.

Os operadores relacionais implementados são os seguintes:

- Operador Maior.
- Operador Menor.

- Operador Maior ou Igual.
- Operador Menor ou Igual.
- Operador Igual.
- Operador Diferente.

4.4.3. Instruções

O MATVOX permite programar com os principais tipos instruções que são encontradas na maioria das linguagens de programação, estas instruções podem ser classificadas nos seguintes tipos ou classes:

- Instruções de saída de dados, as quais permitem apresentar resultados na tela do EDIVOX.
- Instruções iterativas, as quais permitem realizar N numero de vezes um mesmo procedimento.
- Instruções de formatação de resultados, as quais definem a forma como são apresentados os resultados na tela do EDIVOX.
- Instruções de persistência, as quais permitem armazenar permanente dados no sistema para sua posterior utilização.
- Instruções de controle de fluxo, as quais permitem controlar a seqüência de execução do algoritmo.
- Instruções de importação e exportação de dados, as quais permitem importar ou exportar dados desde ou para um arquivo tipo texto (.TXT ou .CSV).
- Instrução para avaliar funções, a qual permite ao deficiente reconhecer funções mediante som.

Todas estas instruções serão explicadas detalhadamente nos capítulos seguintes.

4.4.4. Funções e Constantes Físicas

O MATVOX implementa a maioria das funções básicas matemáticas as quais são utilizadas em expressões e parâmetros de algumas instruções, estas funções estão classificadas da seguinte forma:

- Funções gerais, as quais abarcam funções exponenciais, de aproximação, logarítmicas, entre outras.
- Funções trigonométricas, as quais podem ser básicas, inversas ou hiperbólicas e podem ser trabalhadas em graus ou radianos.
- Conversões, as quais podem ser de comprimento, de massa, de volume, de pressão, de potência, de força, de temperatura e de ângulo.

Por outra parte foram armazenas no sistema as principais constantes utilizadas na física, tais como: *velocidade da luz, constante de Planck, numero de Avogadro, unidade de massa atômica, numero PI,* entre outras, as quais podem ser utilizadas em qualquer momento pelo programa.

4.5. Caso de Uso – Criação e execução de um algoritmo.

Nesta seção vai ser apresentada a criação e execução, passo a passo, de um algoritmo no MATVOX. Os programas no MATVOX podem ser desenvolvidos de duas formas, já seja escrevendo o algoritmo direitamente no editor de texto ou utilizando os menus interativos do programa. A continuação serão apresentadas estas duas formas de implementação.

Implementação direita no EDIVOX.

- Criar um arquivo novo (ver secção 3.2).
- Escrever o comando **\BEGIN;**

- Escrever o algoritmo no editor e finalizar com o comando **\END;**

```
\BEGIN;

  A = 5 * 30 – 4 * 7 ;
  \VarResult{ A };

\END;
```

- Posicionar o cursor em qualquer linha do programa (entre os comandos BEGIN e END)
- Pressionar as teclas Ctrl + F10
- Quando seja ouvida a mensagem "Comandos da calculadora", pressionar a tecla F2. Neste ponto o programa será executado e será ouvida a mensagem "Execução correta" no final da execução.
- Os resultados serão inseridos na linha seguinte do comando END. O texto no EDIVOX ficará da seguinte forma.

```
\BEGIN;

  A = 5 * 30 – 4 * 7 ;
  \VarResult{ A };

\END;

A = 122
```

Implementação utilizando os menus interativos do MATVOX.

- Criar um arquivo novo (ver secção 3.2).
- Pressionar as teclas Ctrl + F10
- Selecionar do menu de comandos a instrução BEGIN e pressionar a tecla ENTER.
- Pressionar as teclas Ctrl + F10

- Selecionar do menu de comandos a instrução EXPRESSION e pressionar a tecla ENTER.
- O programa vai perguntar pela variável dependente da expressão, neste ponto escreve se a letra A, e logo pressiona se a tecla ENTER.
- O programa vai perguntar pela expressão, neste ponto escreve se a expressão (5 * 30 – 4 * 7), e logo pressiona se a tecla ENTER, será ouvida a mensagem Expressão Inserida.
- Pressionar as teclas Ctrl + F10
- Selecionar do menu de comandos a instrução VarResult e pressionar a tecla ENTER.
- O programa vai perguntar pela variável que vai ser impressa, neste ponto escreve se a letra A, e logo pressiona se a tecla ENTER.
- O programa vai perguntar se vão ser impressas mais variáveis, neste ponto é pressionada a tecla ESC o a tecla N.
- Pressionar as teclas Ctrl + F10
- Selecionar do menu de comandos a instrução END e pressionar a tecla ENTER.
- Pressionar as teclas Ctrl + F10
- Quando seja ouvida a mensagem “Comandos da calculadora”, pressionar a tecla F2. Neste ponto o programa será executado e será ouvida a mensagem “Execução correta” no final da execução.
- Os resultados serão inseridos na linha seguinte do comando END.

Capitulo 5

5. Construção do MATVOX

Neste capitulo apresenta-se a parte técnica do MATVOX, onde é mostrado e explicado o processo de desenvolvimento e implementação de cada um dos componentes, que compõem o sistema. Paralelo a isto, são apresentados os diferentes recursos de software, que foram utilizados em cada uma das etapas de programação do aplicativo, esta programação é explicada de maneira geral, mediante exemplos.

5.1. Estrutura do sistema.

O MATVOX não é um aplicativo que funciona independentemente, ele está acoplado direitamente no EDIVOX, o qual é um dos aplicativos principais do DOSVOX. Na Fig. 5.1, pode se observar o diagrama de blocos do sistema que permite a execução do MATVOX.

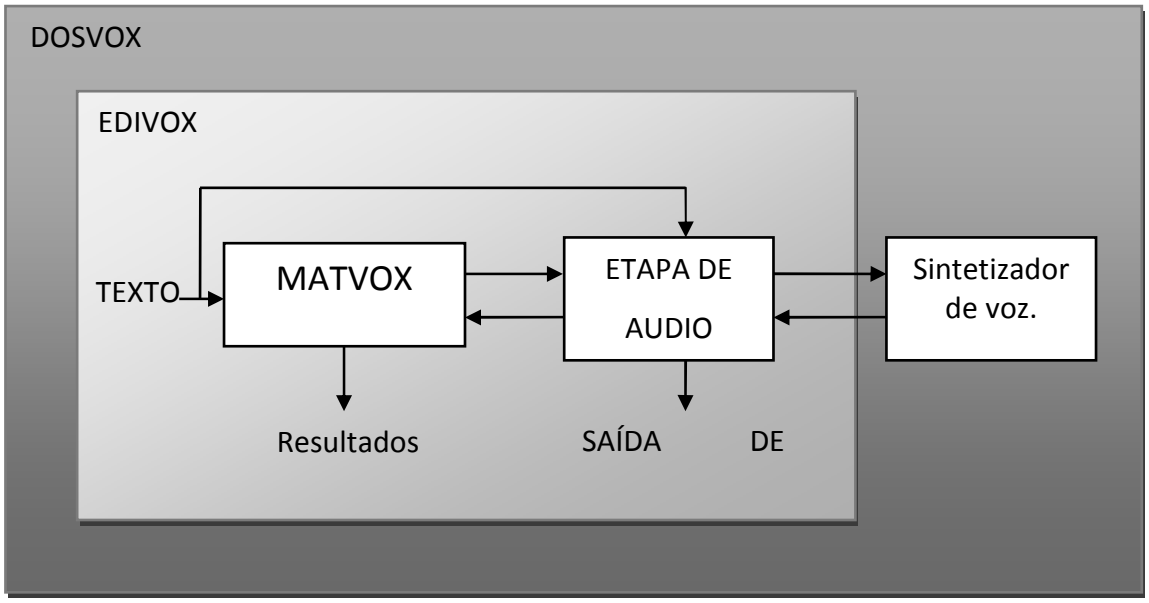


Fig. 5.1: Estrutura geral do sistema de execução do MATVOX

Este sistema está composto principalmente por quatro fases ou etapas:

- **Sistema DOSVOX:** é o sistema que atua como plataforma para a execução do editor de textos EDIVOX.

- **Editor de textos EDIVOX:** é o aplicativo no qual é executado o MATVOX, nele são inseridos os dados de tipo texto, que serão processados pelo MATVOX, e a sua vez permite apresentar os dados ou resultados gerados por este.

- **Etapa de áudio:** esta etapa é a encarregada de gerar os *feedbacks* sonoros do aplicativo.

- **Aplicativo matemático MATVOX:** é o sistema encarregado de identificar, interpretar e executar os algoritmos matemáticos inseridos no EDIVOX.

5.2. O MATVOX no EDIVOX

O MATVOX foi acoplado direitamente no EDIVOX com o propósito de não depender nem necessitar de programas ou aplicativos externos para a execução deste, e a sua vez, ter tanto o algoritmo como seu resultado na mesma janela do programa.

Um algoritmo em MATVOX pode ter a seguinte estrutura e apresentação (Fig. 5.2).

Fig. 5.2: Estrutura geral de um algoritmo no EDIVOX

Na Fig. 5.2 é apresentada a tela principal do EDIVOX. No texto inserido nela podem se identificar três partes:

- **Texto básico:** é um texto qualquer, o qual não tem relevância no MATVOX.

- **Algoritmo matemático:** está formado por instruções ou sentenças as quais são identificadas e interpretadas pelo MATVOX para executar alguma ação.

- **Resultados:** estes são gerados pelo MATVOX e dependem do algoritmo executado. Eles são inseridos automaticamente na linha seguinte do final do algoritmo.

5.3. Software de desenvolvimento

Para a construção do MATVOX foi utilizado o Software *Borland Delphi 6.0* produzido pela *Borland Software Corporation,* o qual utiliza como linguagem de programação uma versão moderna de *Pascal*, chamada *Object Pascal.* Este entorno de desenvolvimento foi utilizado devido a que o sistema DOSVOX e seus aplicativos estão desenvolvidos nele, pelo qual foi apropriado utilizar a mesma linguagem de programação para assim facilitar e evitar inconvenientes no momento de acoplar o MATVOX ao EDIVOX.

5.3.1. Arquivos FONTE do MATVOX

Uma aplicação em *Delphi* está composta por um Programa principal (**.DPR - Delphi Project)* e por Subprogramas chamados *Units* (**.PAS – Pascal)*, os quais a sua vez estão divididos em procedimentos e funções.

Para a construção e implementação do MATVOX foram modificados alguns arquivos (**.DPR e *.PAS)* do código fonte da aplicação EDIVOX e foram criadas algumas *Units* novas, as quais serão apresentadas e explicadas a continuação (Fig. 5.3).

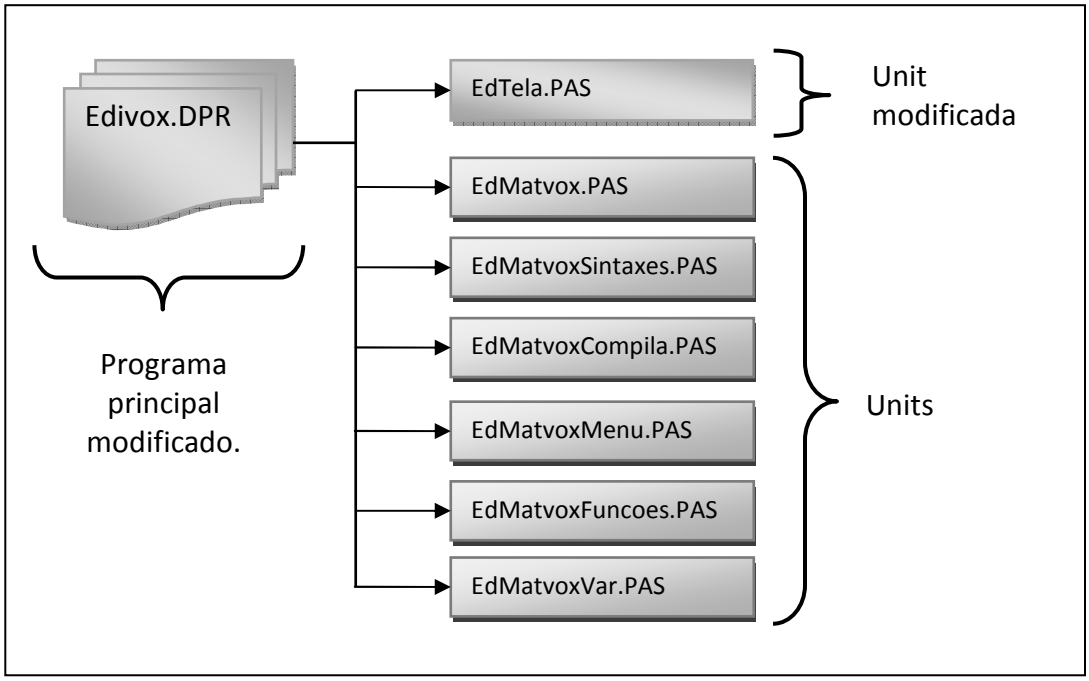


Fig. 5.3: Estrutura de arquivos FONTE do MATVOX.

Edivox.DPR: Arquivo tipo *Delphi Project*, o qual controla cada um dos Subprogramas ou *Units* do EDIVOX.

EdTela.PAS: *Unit* encarregada de controlar a escritura na tela do EDIVOX.

EdMatvox.PAS: *Unit* encarregada da impressão dos resultados do MATVOX na tela do EDIVOX.

EdMatvoxSintaxes.PAS: *Unit* encarregada do analises léxico e sintático do código gerado em MATVOX, e é a encarregada de algumas opções de fala do aplicativo.

EdMatvoxCompila.PAS: *Unit* encarregada da execução do código gerado em MATVOX e a sua vez realiza o analises semântico deste.

EdMatvoxMenu.PAS: *Unit* encarregada de gerar os menus do MATVOX.

EdMatvoxFuncoes.PAS: *Unit* encarregada de executar as diferentes funções matemáticas disponíveis no MATVOX.

EdMatvoxVar.PAS: *Unit* encarregada de definir as variáveis e procedimentos globais do MATVOX.

5.4. Procedimentos e funções principais do código do MATVOX

Na continuação são apresentados os principais procedimentos do código fonte do MATVOX. Estes são os encarregados de executar as diferentes etapas de analise, execução e apresentação de resultados do código inserido no EDIVOX. A descrição dos procedimentos e funções não apresentadas na tabela estão explicadas e comentadas no código fonte do MATVOX.

UNIT	PROCEDURE	DESCRIÇÃO
Edivox.DPR	trataPFeALT	Leitura das teclas de controle do EDIVOX
EdMatvoxMenu.PAS	ayudaComandos	Apresenta o menu de comandos.
	imprimeMenu	Leitura das teclas de controle dos menus do MATVOX.
	setaF2E	Ativa os procedimentos de analise e execução do algoritmo inserido no EDIVOX.
	setasMenuIterativo	Cria os diferentes menus do MATVOX
	fondoMenu	
	insertarFuncion	Insere as funções no EDIVOX
	leeComando	Sintetiza cada umas das opções do menu.
	insertaParametros	Crias o procedimento para

		inserir interativamente os comandos e funções no EDIVOX.
EdMatvoxSintaxes.PAS	analisaSintasis	Contem o hilo de execução do processo de analise do algoritmo.
	detectaRegion	Verifica a configuração regional do computador (ponto ou vírgula)
	buscaBEGIN	Pesquisa do inicio do algoritmo
	buscaEND	Pesquisa do final do algoritmo
	guardaPalabra	Copia uma linha do EDIVOX para seu analise.
	separaPalabra	Acondiciona a linha copiada para seu analise.
	identificaComando	Identifica o comando da linha analisada.
	identificaPatron	Identifica a estrutura da linha.
	verificaComandoCOMANDO	Valida o comando e sua estrutura.
	cicloVerificacion	Analisa cada linha do algoritmo dependendo do comando encontrado nela.
	guardaCodigo	Cria o código objeto que será executado.
	procesoLEITURA	Hilo de execução para a fala detalhada de sentenças.
	leeGrafica	Sintetiza a função do comando ReadFunction.
	leeErros	Sintetiza os erros encontrados na sentença.
	buscaComandoLEITURA	Inicializa o processo de leitura da sentença dependendo do comando encontrado nela.
EdMatvoxCompila.PAS	acondicionamentoCodigo	Acondiciona o código objeto
	defineVectorPrincipalOperacion	Seleciona o comando a executar

	expresion	Procedimento para a solução de expressões
EdMatvox.PAS	imprimeResultadosTela	Apresenta os resultados na tela do EDIVOX

Tab. 5.1: Principais procedimentos do código fonte do MATVOX.

Para fazer o acoplamento do MATVOX no EDIVOX foi modificado o procedimento ***trataPFeALT*** da ***Unit Edivox.DPR***. Na continuação é apresentado o código fonte que permitiu este acoplamento:

```
                                                    ...

// PROGRAMA CALCULADORA MATVOX – Fala detalhada de sentencas
      CTLF9:          begin
                              telaAutor;
                              try
                                    procesoLEITURA;
                              except
                              end;
                      end;
// FIM PROGRAMA CALCULADORA MATVOX

      F10:            pedeMargens;

// PROGRAMA CALCULADORA MATVOX – Menu de Comandos
      CTLF10:         begin
                              telaAutor;
                               sintSom('\COMANDOS\MENU\F62');
                              ayudaComandos;
                      end;
// FIM PROGRAMA CALCULADORA MATVOX

                                                    . . .
```

5.5. Arquitetura do MATVOX

O MATVOX é basicamente um interpretador avançado, o qual identifica e analisa um código escrito no EDIVOX para posteriormente o executar. O funcionamento deste interpretador é muito simples e seu processo de execução tem principalmente três passos ou etapas: a primeira consiste na identificação do código ou algoritmo matemático que será interpretado pelo MATVOX, a segunda permite processar, depurar e interpretar esse código para assim obter algum resultado e a terceira é a encarregada de inserir os resultados novamente no EDIVOX. A Fig. 5.4 apresenta o diagrama de blocos da estrutura básica do MATVOX com cada uma das suas etapas.

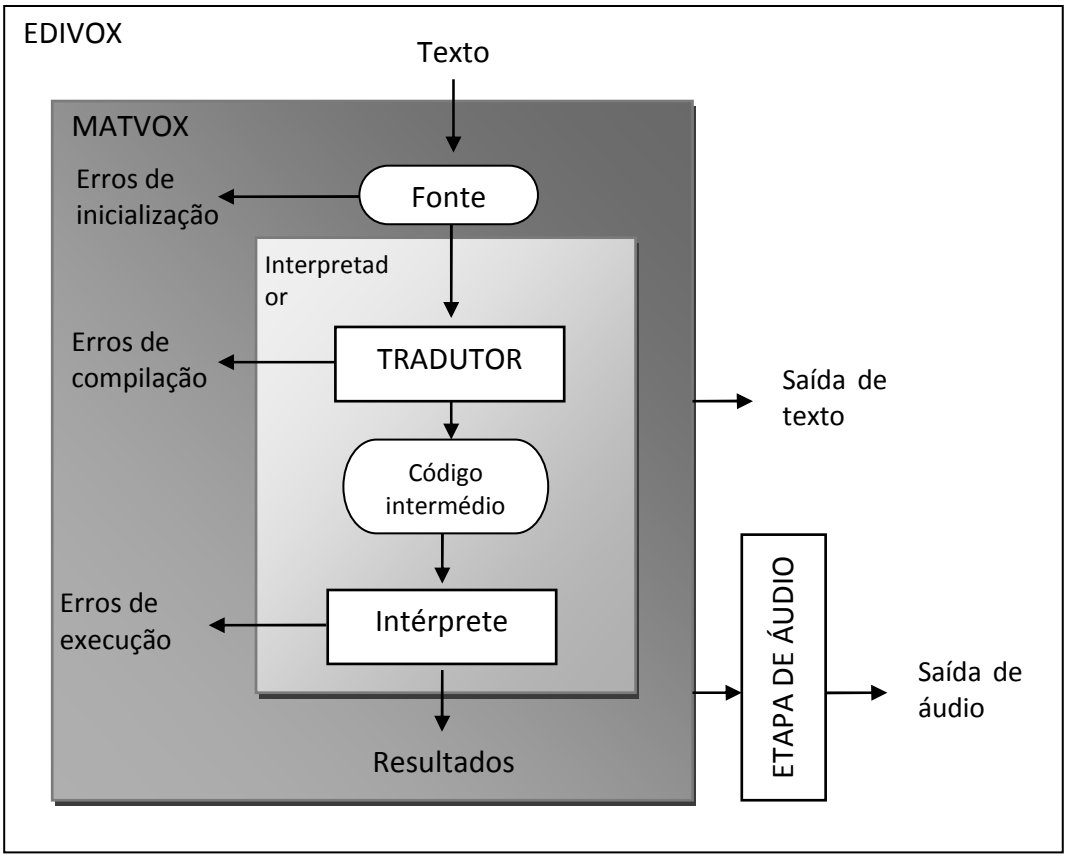


Fig. 5.4: Fases do MATVOX

5.5.1. Etapa FONTE

A etapa definida como *FONTE* é a encarregada de identificar e extrair de um texto qualquer o algoritmo que será interpretado e executado pelo MATVOX.

A Fig. 5.5 apresenta o diagrama de blocos da etapa *FONTE* a qual está dividida em dois processos: rastreamento da instrução **\BEGIN;** e rastreamento da instrução **\END;**.

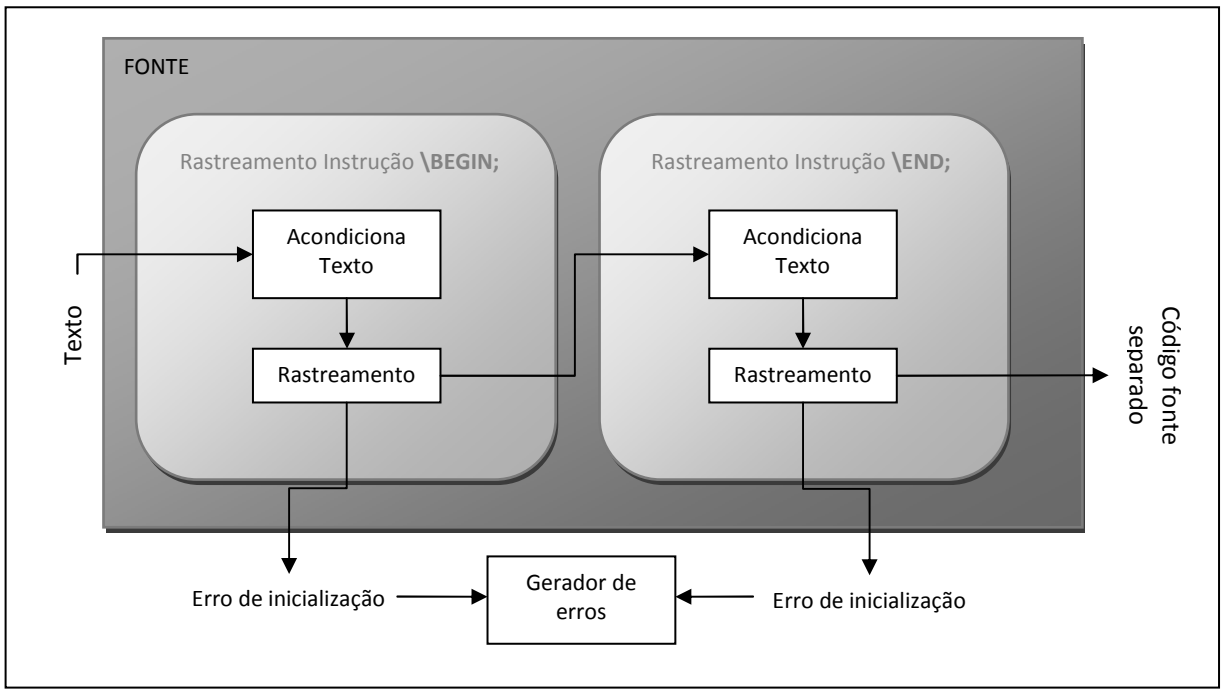


Fig. 5.5: Diagrama de blocos da etapa FONTE do MATVOX

Estes dois processos permitem identificar e separar o código que será executado do resto do texto escrito no EDIVOX. Esta etapa é executada pela *Unit EdMatvoxSintaxes.PAS* da seguinte maneira:

Rastreamento da instrução \BEGIN;

1. Faz uma copia da linha atual do cursor.
2. Acondiciona a linha copiada (apaga espaços em branco, converte todos os caracteres a maiúscula, apaga comentários e apaga etiquetas).
3. Verifica se a linha copiada depois do acondicionamento é igual a **\BEGIN;**.
4. Se a instrução **\BEGIN;** é encontrada, o seguinte processo é executado, senão o processo anterior é repetido com a linha superior.
5. Se todas as linhas superiores são analisadas, e a instrução **\BEGIN;** não é encontrada, então será gerado um erro indicando a ausência desta instrução.

Rastreamento da instrução \END;

1. Faz uma copia da linha atual do cursor.
2. Acondiciona a linha copiada (apaga espaços em branco, converte todos os caracteres a maiúscula, apaga comentários e apaga etiquetas).
3. Verifica se a linha copiada depois do acondicionamento é igual a **\END;**.
4. Se a instrução **\END;** é encontrada, o processo da etapa *FONTE* finaliza, senão o processo anterior é repetido com a linha inferior.
5. Se todas as linhas inferiores são analisadas, e a instrução **\END;** não é encontrada, então será gerado um erro indicando a ausência desta instrução.

Gerador de erros

1. Verifica se tem tido erros no processo.
2. Se tem tido erros:
    - Armazena o número da linha do erro.
    - Armazena a descrição do erro.
    - Ativa um *flag*, que indica a existência de erros.
    - O processo atual é abortado.

No exemplo da Fig. 5.6, depois de finalizar o processo da etapa *FONTE*, o programa identifica a "*Linha 4"* como o inicio do algoritmo, e a *"Linha 6"* como o final do algoritmo. O resto do texto é ignorado pelo programa.

Fig. 5.6: Exemplo da identificação de código fonte no texto.

5.5.2. Etapa do tradutor

Quando o código fonte é corretamente identificado e separado pela etapa *FONTE,* este é analisado linha por linha pelo *TRADUTOR*. O *TRADUTOR* é o programa encarregado de analisar e validar a escritura de todo o código, ele tem principalmente duas etapas: etapa de analises e etapa de sínteses (Fig. 5.7). Esta etapa é executada pela *Unit EdMatvoxSintaxes.PAS*.

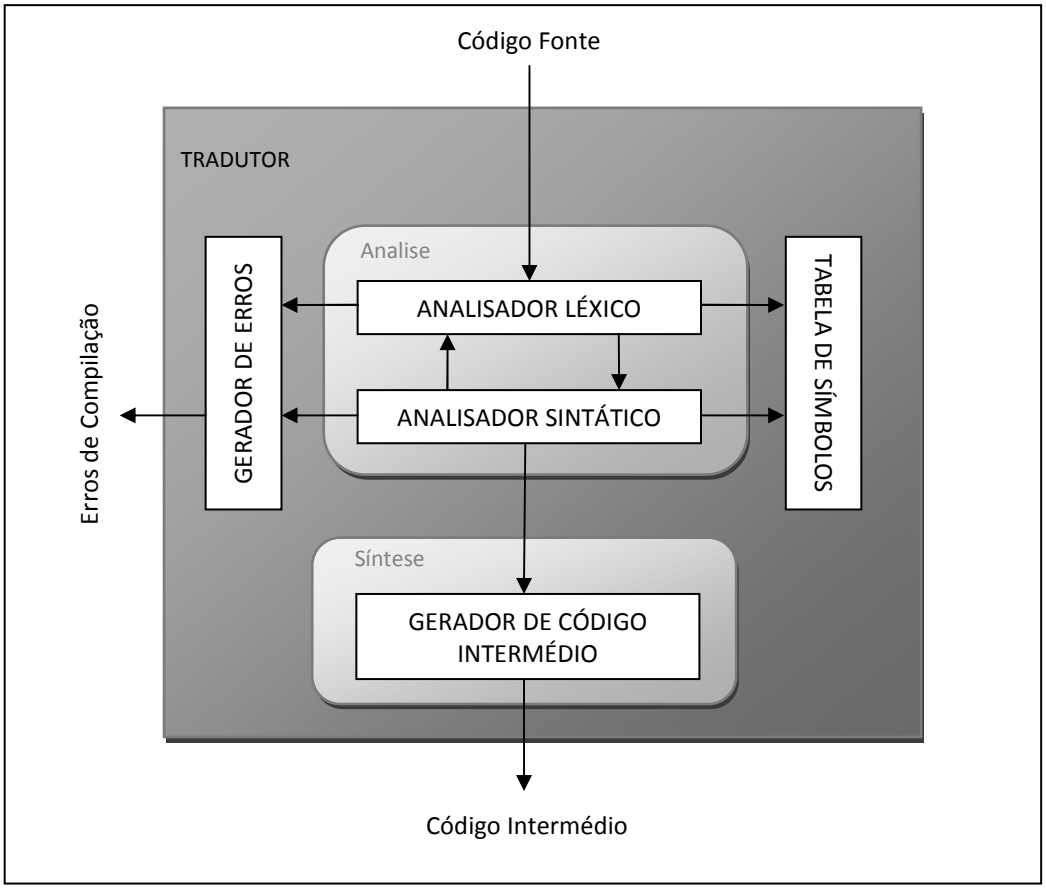


Fig. 5.7: Diagrama de blocos do tradutor do MATVOX

A etapa de analises é a encarregada de analisar o léxico e a sintaxes do código fonte (o analises semântico é feito mais adiante), nesta etapa podem ser gerados erros de compilação, os quais estão relacionados com a escritura do código, mas não com sua execução.

A segunda etapa é a encarregada de gerar um código intermédio o qual é interpretado e executado posteriormente pelo interpretador.

5.5.2.1. Analisador LÉXICO

O analisador léxico tem o propósito de identificar e agrupar em *tokens* ou *unidades sintáticas* cada uma das palavras ou caracteres do código fonte, estes *tokens* podem ser

operadores, constantes, variáveis, comandos, etiquetas etc. Para a identificação e extração de cada um dos *tokens* é executado o seguinte processo (Fig. 5.8).

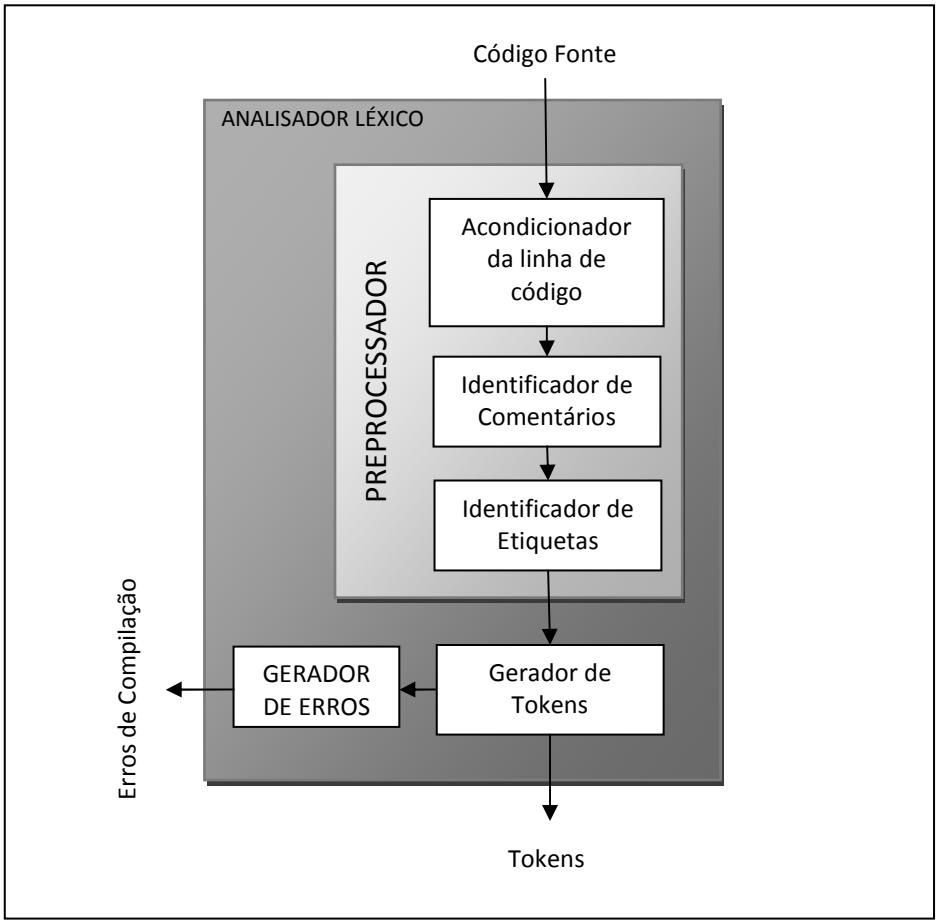


Fig. 5.8: Diagrama de blocos do analisador léxico do tradutor do MATVOX.

O processo do analises léxico é feito linha por linha começando desde a instrução **\BEGIN;** até a instrução **\END;** as quais foram previamente identificadas na etapa *FONTE.* Este analise permite validar e identificar cada uma das instruções e parâmetros do código a executar, para que assim, este seja posteriormente processado pelo analisador sintático. Este processo é executado da seguinte forma:

Etapa de préprocessamento:

O préprocessador é o encarregado de fazer algumas conversões léxicas no código do programa, as conversões são as seguintes:

1. Conversão de todos os caracteres alfabéticos a maiúsculas. Isto permite que as seguintes sentenças tenham o mesmo significado.

$$\boldsymbol{Variavel = 5} \ \textit{e} \ \boldsymbol{VARiaVel = 5}$$

2. Remoção dos espaços em branco. Isto permite que as seguintes sentenças tenham o mesmo significado.

$$\boldsymbol{Var\ iavel = 5} \ \textit{e} \ \boldsymbol{Va\ ri\ \ v\ \ e\ l = \ \ 5}$$

3. Inserção da sentença \NOP; nos saltos de linha.
4. Troca do símbolo "." (ponto) por o símbolo "," (virgula), ou vice-versa, dependo da configuração regional (símbolo decimal) do computador. Isto permite que as seguintes sentenças tenham o mesmo significado.

$$\boldsymbol{A = 3.14} \ \textit{e} \ \boldsymbol{A = 3,14}$$

5. Identificação e remoção de comentários, eles são armazenados na *Tabela de Símbolos*.
6. Identificação e remoção de etiquetas, elas são armazenadas na *Tabela de Símbolos*.

Gerador de *Tokens*

1. Identifica os diferentes *tokens* encontrados na linha, tais como: instruções, variáveis, operadores, etc.
2. Armazena os *tokens* encontrados na linha.

Gerador de erros

1. Verifica se tem tido erros no processo.
2. Se tem tido erros:
    - Armazena o número da linha do erro.
    - Armazena a descrição do erro.
    - Ativa um *flag*, que indica a existência de erros.
    - O processo atual é abortado.
3. Se não tem tido erros, o processo é repetido com a seguinte linha.

A Tab. 5.2 apresenta o algoritmo e os *tokens* identificados no programa "*Oi Mundo!!!",* mostrado anteriormente.

SENTENÇAS	TOKENS	DESCRIÇÃO
\Begin;	\	Identificador de instrução.
	Begin	Instrução.
	;	Separador de sentenças.
\Print { "Oi"+"Mundo!!!" } ;	\	Identificador de instrução.
	Print	Instrução.
	{	Separador de parâmetros.
	"	Indicador de *String*.
	Oi	Identificador.
	"	Indicador de *String*.
	+	Operador aritmético.
	"	Indicador de *String*.

	Mundo!!!	Identificador.
	"	Indicador de *String*.
	}	Separador de parâmetros.
	;	Separador de sentenças.
\End;	\	Identificador de instrução.
	End	Instrução
	;	Separador de sentenças.

Tab. 5.2: Tokens identificados no programa "Oi Mundo!!!"

No exemplo da Tab. 5.2 todos os *tokens* do algoritmo foram identificados corretamente, sem embargo quando isto não sucede o programa vai gerar um erro de compilação. No exemplo da Fig. 5.9, pode se observar um erro na compilação do algoritmo, neste caso o analisador léxico não identifica a palavra **Prrint** como um *token* válido (*token* válido: **Print**).

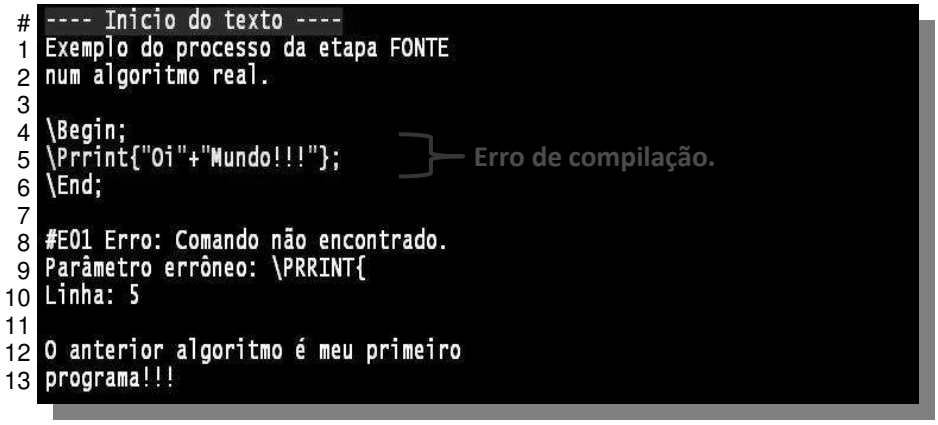


Fig. 5.9: Exemplo de erro de léxico no MATVOX, token não identificado

5.5.2.2. Analisador sintático

Quando todos os *tokens* são identificados e analisados pelo *Analisador Léxico*, o *Analisador Sintático* é executado. Ele é o encarregado de analisar e verificar a estrutura das

sentenças do algoritmo, comprovando se os *tokens* formam uma expressão válida. Para fazer isto, estão definidas umas regras especificas as quais determinam a ordem correto dos *tokens* dependendo do tipo de sentença analisada.

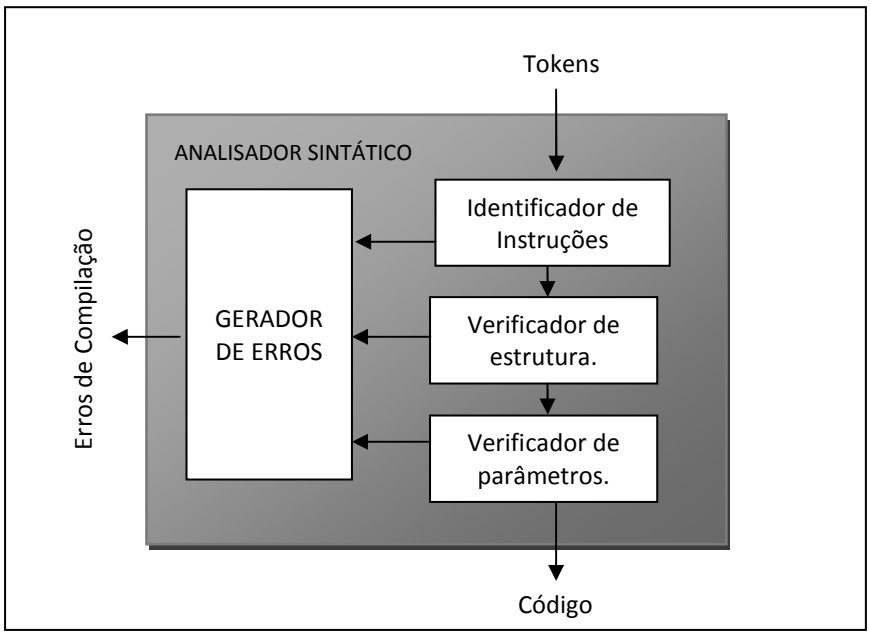


Fig. 5.10 Diagrama de blocos do analisador sintático do tradutor do MATVOX.

O processo de analises sintático é apresentado na Fig. 5.10 e tem as seguintes etapas de execução:

Identificador de instruções:

É o encarregado de identificar na sentença analisada a instrução definida nesta.

1. Procura na sentença o *token* de tipo: *instrução.*
2. Se não for encontrado nenhum *token* de tipo *instrução*, então a sentença é definida como uma *expressão,* e são inseridos os *tokens* **"\" + "Expression" + "{".**

3. Procura o *token* encontrado na *Tabela de Símbolos,* na qual estão armazenadas os identificadores de cada uma das instruções do MATVOX.

Verificador de estrutura e parâmetros:

São os encarregados de verificar a estrutura de cada sentença, ou seja, o tipo e a ordem dos *tokens* que definem a esta.

1. Avalia a sentença de acordo ao tipo de instrução encontrado pelo *identificador de instruções.* Isto permite avaliar a correta estrutura de cada uma delas, por exemplo, na seguinte sentença (Fig. 5.11) tem que ser identificados os tokens, *<Barra invertida>* , *<abre chave>, <fecha chave>, <ponto e virgula>* nessas posições.

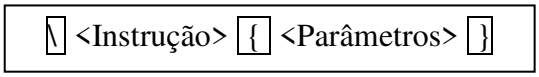


Fig. 5.11: Avaliação da estrutura de sentenças no MATVOX

2. Uma vez validada a estrutura da sentença, são avaliados o ordem e a estrutura de cada um dos parâmetros da instrução identificada, na qual, por exemplo, os parâmetros da instrução *IF* têm que estar conformados por duas expressões e um operador relacional numa ordem específica (Fig. 5.12).

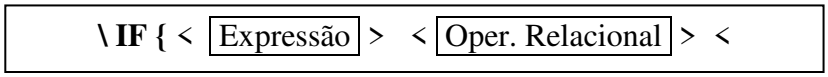


Fig. 5.12: Avaliação de parâmetros das instruções no MATVOX

Gerador de erros

1. Verifica se tem tido erros no processo.

2. Se tem tido erros:

- Armazena o número da linha do erro.
- Armazena a descrição do erro.
- Ativa um *flag*, que indica a existência de erros.
- O processo atual é abortado.

3. Se não tem tido erros, o processo é repetido com a seguinte sentença.

No exemplo da Fig. 5.13 é gerado um erro de compilação devido a que o *token* **"+"** da instrução "**Print"** não está na posição correta**.**

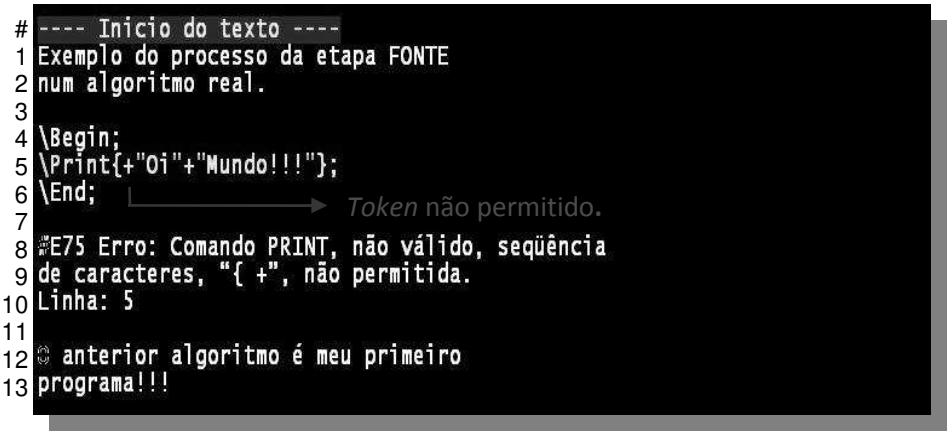


Fig. 5.13: Exemplo de erro sintático no MATVOX, token não permitido

5.5.2.3. Tabela de símbolos.

Na fase de inicialização e tradução são gerados alguns dados, os quais são armazenados na *Tabela de Símbolos*, estes dados contêm a informação necessária para as fases seguintes da execução do interpretador. Os dados gerados e armazenados são os seguintes:

- **Tabela de erros do MATVOX:** contém a listagem de todos os erros do MATVOX.
- **Palavras reservadas:** contém as palavras reservadas do sistema.

- **Linha das sentenças:** armazena a posição das sentenças no texto. Isto permite obter e apresentar informação adicional (número de linha) quando são gerados erros.
- **Etiquetas:** armazena as etiquetas com sua respectiva posição no programa. Estas são utilizadas pela instrução GOTO.
- **Comandos:** contêm todos os comandos disponíveis no MATVOX.
- **Funções:** contêm todas as funções disponíveis no MATVOX.

5.5.2.4. Gerador de código intermédio

O *Gerador de código intermédio* é o encarregado de gerar e otimizar o código que será executado pelo interpretador na seguinte etapa. Este novo código tem a seguintes diferenças com o código original:

- O código não tem espaços em branco.
- O código não tem saltos de linha.
- Todos os caracteres alfabéticos estão em maiúscula.
- Os comentários foram removidos.
- As etiquetas foram removidas.
- As expressões têm alguns *tokens* adicionais.
- Algumas instruções têm alguns parâmetros adicionais.

Todos estes câmbios têm o objetivo de facilitar a interpretação do código e a sua vez evitar possíveis erros na etapa de execução do algoritmo por parte do interpretador. A Fig. 5.14 apresenta um algoritmo com seu correspondente código otimizado.

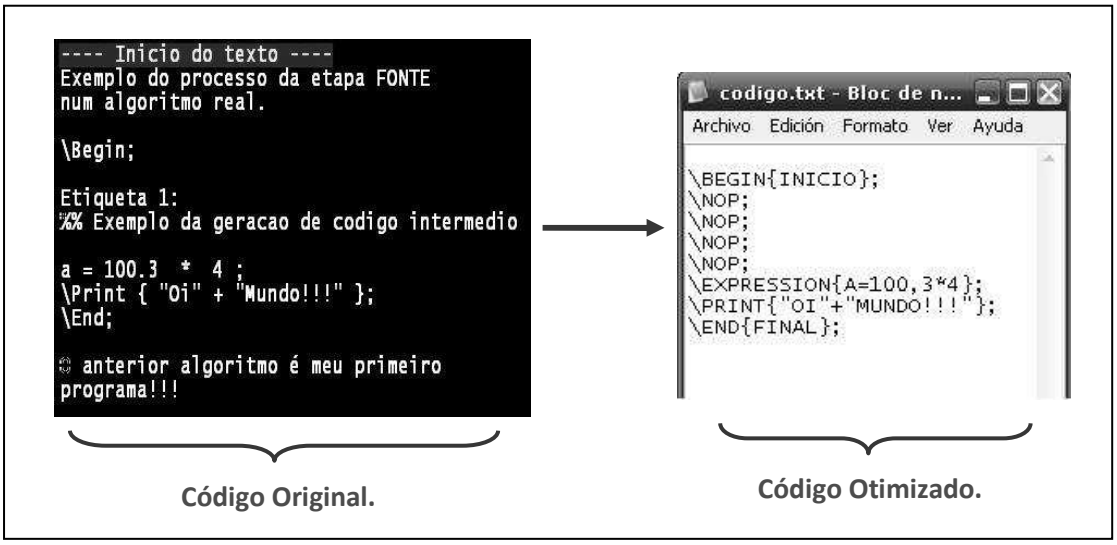


Fig. 5.14: Código intermédio gerado pelo tradutor do MATVOX.

Quando o código otimizado é gerado corretamente, significa que ele não tem nenhum erro no seu léxico, nem na sua sintaxe, pelo qual ele está pronto para sua execução. Na seguinte etapa de interpretação são possíveis que sejam gerados diferentes tipos de erros, mas estes não estão relacionados com a escritura do algoritmo, senão com sua lógica, por exemplo: "Variáveis não definidas", "Divisão por zero", etc.

5.5.3. Etapa do interpretador

O *Interpretador* tem como objetivo analisar, validar e executar o código ou algoritmo que foi gerado na etapa anterior pelo *Tradutor.* O processo de validação e execução do algoritmo por parte do *interpretador puro* é feito linha por linha, pelo qual este mesmo processo será repetido até que sejam processadas todas as sentenças do código ou até que devido a um erro de execução o programa tenha que ser abortado (Fig. 5.15). Esta etapa é executada pela *Unit EdMatvoxCompila.PAS*

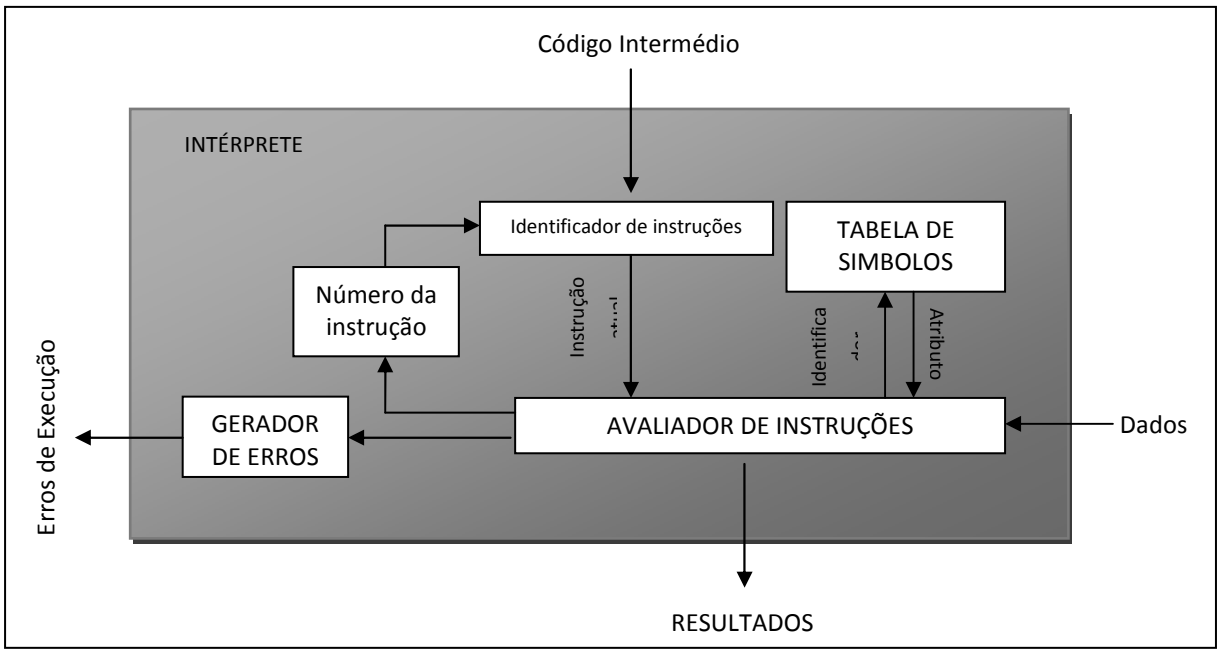


Fig. 5.15: Diagrama de Blocos do Interpretador do MATVOX.

5.5.3.1. Identificador de instruções

O *identificador de instruções* tem como função, extrair de cada sentença a instrução que tem que ser executada pelo *Avaliador de instruções*. Para realizar isto, ele executa o seguinte procedimento:

1. Faz uma copia da sentença a analisar.
2. Identifica a instrução na *Tabela de Símbolos* para sua posterior execução.
3. Envia para o *Avaliador de Instruções* os dados da instrução identificada.

5.5.3.2. Tabela de Símbolos

Na etapa de interpretação, a tabela de símbolos é acrescentada com alguns dados os quais são os seguintes:

- Constantes armazenadas no sistema (constantes físicas).
- Constantes armazenadas pelo usuário.
- Variáveis definidas no código do programa.

5.5.3.3. Avaliador de instruções

O *Avaliador de instruções* é o encarregado de executar cada uma das sentenças do código mediante uma serie de regras e processos definidos e programados no MATVOX (Fig. 5.16).

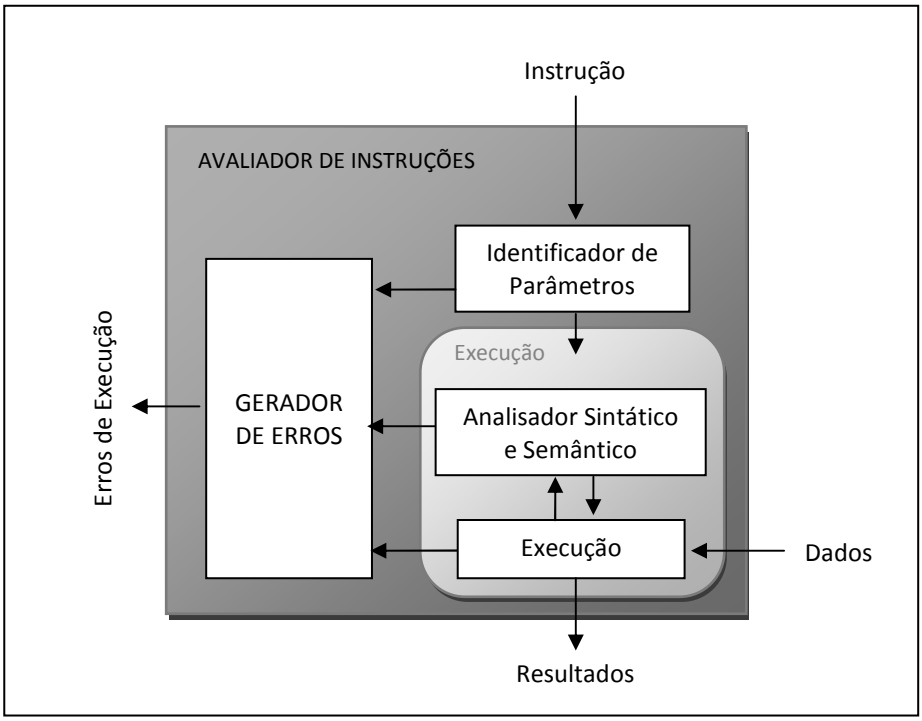


Fig. 5.16: Diagrama de blocos do avaliador de instruções do MATVOX.

Identificador de Parámetros.

O *Identificador de Parâmetros* tem a função de separar cada um dos parâmetros que compõem a sentença ou instrução, estes parâmetros são aqueles que estão no meio das chaves na

frente da instrução. Por exemplo, os parâmetros do código da Fig. 5.14 definida anteriormente, seriam separados da seguinte maneira Tab. 5.3.

SENTENÇAS	PARÂMETROS	DESCRIÇÃO
\BEGIN{INICIO};	INICIO	Indicador de inicio do algoritmo.
\NOP;	Sem parâmetros	
\NOP;	Sem parâmetros	
\NOP;	Sem parâmetros	
\NOP;	Sem parâmetros	
\EXPRESSION{A=100,3*4};	A	Variável dependente da expressão.
	=	Operador de definição.
	100,3	Constante numérica.
	*	Operador aritmético.
	4	Constante numérica.
\PRINT{"OI"+"MUNDO!!!"};	"OI"	*String*
	+	Operador de concatenação
	"MUNDO!!!"	*String*
\END{FINAL};	FINAL	Indicador de final do algoritmo.

Tab. 5.3: Exemplo de identificação de parâmetros no interpretador do MATVOX.

Analisador Sintático, Semântico e execução de instruções.

Na fase de interpretação uma das tarefas do *Analisador Sintático* consiste na criação dos arvores binários para a avaliação de expressões e execução de instruções, este processo trabalha simultaneamente com o *Analisador Semântico,* o qual é o encarregado de detectar a validez da semântica de cada uma das sentenças. Nele são detectados e verificados os erros semânticos, por exemplo: se as variáveis de uma expressão têm sido definidas previamente, se vai ser feita uma multiplicação entre dados de tipos diferentes, se vai ser feita uma divisão por zero, etc.

A etapa de *execução* é um processo que é realizado paralelamente com os *Analisadores Sintáticos e Semânticos*, no qual cada sentença é analisada e validada, e imediatamente executada pelo programa. Este processo é repetido sucessivamente até executar todo o código.

Por exemplo, na avaliação de uma expressão (Fig. 5.17) estes três processos trabalham da seguinte maneira:

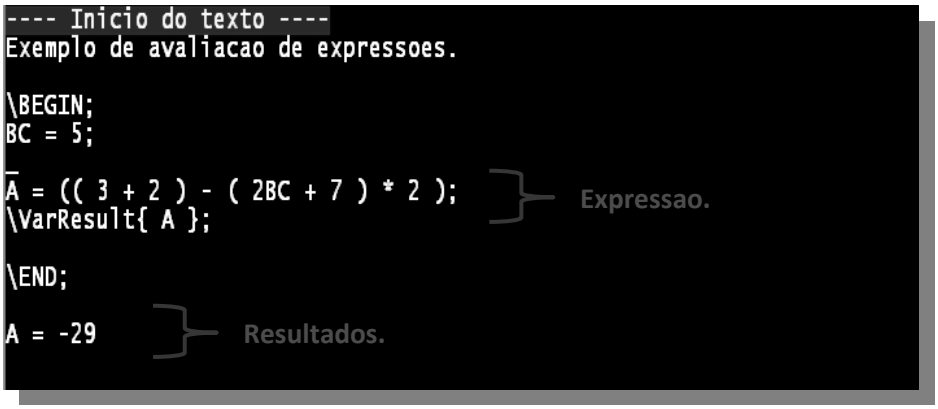


Fig. 5.17: Exemplo de avaliação de uma expressão pelo interpretador do MATVOX.

1. Separação da variável dependente.

A variável dependente (neste caso o identificador "A") é separada da expressão. O operador "=" é ignorado.

2. Separação de parâmetros e avaliação da expressão (Tab. 5.4).

FASE	PROCESSO	RESULTADO
-	Expressão	$\boldsymbol{A} = ((3+2)-(2BC+7)*2)$
1	Separação	((3 + 2) - (2 * BC + 7) * 2)
2	Substitui identificadores	((3 + 2) - (2 * 5 + 7) * 2)
3	Niveles profundidade	***3*** ***2*** ***3*** ***3*** ***2***
4	Operad. Prioridade 1	***1*** ***4***
5	Operad. Prioridade 2	***2*** ***5*** ***3***
6	Operação 1	((3 + 2) - (2 * 5 + 7) * 2)
7	Operação 2	((3 + 2) - (10 + 7) * 2)
8	Operação 3	(5 - (10 + 7) * 2)
9	Operação 4	(5 - 17 * 2)

10	Operação 5	(5 - 34)
11	Resultado	-29

Tab. 5.4: Fases de avaliação de uma expressão no MATVOX.

Fase 1: São separados todos os operadores, parênteses, constantes e identificadores da expressão.

Fase 2: se a expressão contém algum identificador (neste caso o identificador BC), este é procurado no sistema e posteriormente é substituído por seu valor correspondente. Estes identificadores podem ser:

- Variáveis definidas previamente (neste caso BC = 5).
- Constantes definidas pelo usuário no sistema.
- Constantes armazenadas no sistema.

Fase 3: É identificado o nível de profundidade de cada um dos operadores na expressão.

Fase 4 - 5: A cada operador é atribuído um valor de prioridade, este valor depende do nível de profundidade e do tipo de operador (um operador de produto vai ter maior prioridade que um operador de soma).

Fase 6 – 10: São feitas cada uma das operações de acordo às prioridades definidas nas fases 3 e 4. A *Tabela de símbolos* é acrescentada com os resultados de cada uma destas operações (informação necessária para a instrução *\StepResult{}).*

Fase 11: a variável dependente extraída previamente é armazenada na tabela de símbolos e o resultado da expressão é atribuído a ela.

Gerador de erros

1. Verifica se tem tido erros no processo.
2. Se tem tido erros:
    - Armazena o número da linha do erro.

- Armazena a descrição do erro.
- Ativa um *flag*, que indica a existência de erros.
- O processo atual é abortado.

3. Se não tem tido erros, o processo é repetido com a seguinte sentença.

5.5.4. Etapa de Áudio

A etapa de áudio é a encarregada de apresentar os resultados de forma auditiva ao usuário. Ela está composta por dois elementos principais, um para a saída de conteúdo dinâmico (sintetizador) e outro para a saída de conteúdo estático (arquivos de áudio).

Para o conteúdo dinâmico o DOSVOX utiliza o sintetizador *MBROLA* desenvolvido pela universidade Belga (*the TCTS Lab of the Faculté Polytechnique de Mons*), o qual está disponível para vários idiomas. Este sintetizador é de uso gratuito para aplicações com propósitos não comerciais.

Para o conteúdo estático o MATVOX reproduz uma serie de arquivos de som, os quais estão armazenados nas pastas de instalação do DOSVOX (pasta *Som* do EDIVOX) e os quais são reproduzidos dependendo da ação executada.

5.5.4.1. Fala de sentenças

A *Etapa de Áudio* também é a encarregada do processo de *Fala detalhada de sentenças,* explicada anteriormente e a qual é uma das características principais do MATVOX. Para este processo é realizado o seguinte procedimento:

1. Identificação e armazenagem dos diferentes parâmetros da sentença.

Neste passo, o MATVOX mediante o *Analisador Léxico* identifica, separa e armazena cada um dos parâmetros da sentença os quais podem são:

- Comentários.
- Etiqueta.

- Instrução.
- Parâmetros da instrução (variáveis, operadores, expressões, etc.).

2. Validação de cada um dos parâmetros.

Neste passo, o MATVOX analisa cada um dos parâmetros identificados no passo 1, mediante o *Analisador Sintático.* Depois do análise o MATVOX vai gerar uma seqüência de reprodução dependendo dos parâmetros e da valides destes. Por exemplo, uma seqüência pode estar formada da seguinte forma:

<Etiqueta> **Um:** <Instrução> **IF** <Expressão> **A** <Operador> **<>** <Expressao> <Mensagem de erro ><Comentário> **Exemplo**

Na anterior seqüência está composta pelos comentários *Exemplo*, a etiqueta *Um*, a instrução *IF*, a expressão de comparação *A*, o operador de comparação *Diferente* e uma mensagem de erro pela ausência da expressão de comparação 2.

3. Reprodução da seqüência.

Neste ultimo passo o MATVOX identifica o conteúdo dinâmico e o conteúdo estático e de acordo a isso reproduze a seqüência, no casso anterior a reprodução vai ser a seguinte:

- **"Os comentários são"**: conteúdo estático, arquivo de áudio..
- **"Exemplo"**: conteúdo dinâmico, sintetizador.
- **"O Label é"**: conteúdo estático, arquivo de áudio.
- **"Um"**: conteúdo dinâmico, sintetizador.
- **"Comando BEGIN"**: conteúdo estático, arquivo de áudio.
- **"A primeira expressão de comparação é":** conteúdo estático, arquivo de áudio.
- **"A":** conteúdo estático, arquivo de áudio.
- **"O operador é":** conteúdo estático, arquivo de áudio
- **"Operador diferente":** conteúdo estático, arquivo de áudio

- **"A segunda expressão de comparação não está definida ou não é válida":** conteúdo estático, arquivo de áudio.

5.5.4.2. Reprodução de funções matemáticas

Outras das características importantes do MATVOX é sua capacidade de reproduzir por meio do som, funções matemáticas, o qual permite ao deficiente visual ter uma idéia da forma de determinada função. Este processo é realizado também pela *Etapa de Áudio*, mediante a reprodução de tonos e é realizado da seguinte maneira:

1. **Avaliação da função:** este processo é realizado pela etapa de *execução* do interpretador do MATVOX, e consiste em avaliar a função matemática em determinados pontos definidos pelo usuário.

2. **Acondicionamento da amplitude:** a magnitude de cada ponto da função é acondicionada e logo é relacionada à freqüência de um tono (a amplitude é direitamente proporcional à freqüência do tono).

3. **Cálculo da pendente de cada ponto da função:** a pendente calculada estabelece a duração de cada um dos tonos gerados para a função.

4. **Reprodução de cada um dos tonos gerados:** Na Fig. 5.18 os quadrados representam os pontos de avaliação da função X, as setas de linhas continuas representam as amplitudes da função e, por conseguinte a freqüência de cada tono, e as setas de linhas descontinuas representam a pendente de cada ponto da qual dependem a duração de cada tono.

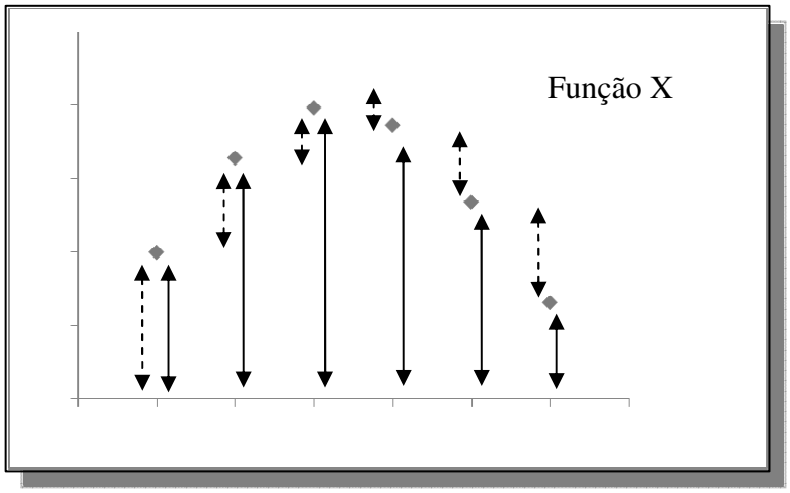


Fig. 5.18: Representação de amplitudes e pendentes de uma função.

Capitulo 6

6. Conclusões e Trabalhos Futuros

O aplicativo desenvolvido (MATVOX) implementa numa mesma aplicação (EDIVOX), todo o necessário para desenvolver e executar algoritmos. Isto permite que o deficiente visual consiga implementar cálculos matemáticos na mesma janela do editor de textos, o qual evita a necessidade de utilizar aplicativos externos a este.

Para facilitar o desenvolvimento de algoritmos por parte dos usuários, foi implementado um sistema de menus interativos na interface do MATVOX, estes permitem que o deficiente visual interatue de forma natural e amigável com o computador, o qual evita a necessidade de ter algum conhecimento prévio do programa. Estes menus foram desenvolvidos tendo em conta as características e limitações próprias de este perfil de usuário, eles contem mensagens sonoras que orientam em todo momento a usuário no processo de criação e execução de algoritmos.

Também foi criado um sistema de fala que permite entre outras coisas, verificar de forma simples a estrutura de cada uma das sentenças do algoritmo. Isto faz que o deficiente visual consiga detectar interativamente os possíveis erros na escritura deste, sem a necessidade de ter que realizar este processo mentalmente.

Por outra parte, a linguagem de programação desenvolvida tem regras de escrita pouco rigorosas com o usuário, o qual diminui enormemente a geração de erros no desenvolvimento de algoritmos. A linguagem inclui instruções e funções similares às encontradas em outras linguagens de programação comerciais, e contem características próprias orientadas especificamente para pessoas com algum grau de deficiência visual, isto faz que o MATVOX seja uma verdadeira alternativa gratuita para o estudo e desenvolvimento de trabalhos nas áreas das ciências exatas.

6.1. Sugestões para trabalhos futuros.

O aplicativo desenvolvido implementa as principais instruções e funções encontradas nas maioria de linguagens de programação, sem embargo o MATVOX ainda pode ser atualizado com diversas funcionalidades que permitam ter um aplicativo mais poderoso e eficiente. Algumas destas características são as seguintes:

- Implementação de um sistema de depuração que permita a execução do algoritmo passo a passo, o qual facilitaria o analises em tempo real do algoritmo.
- Criação de instruções que permitam a definição de procedimentos e funções, o qual permitiria o desenvolvimento de algoritmos mais estruturados.
- Criação de instruções que permitam a pesquisa e filtragem dos dados importados mediante as instruções *ImportText* e *ImportPlanivox.*
- Criação e implementação de operadores lógicos (And, Or, etc.), os quais permitam avaliar numa mesma linha diversas condições.
- Criação de funções que permitam a avaliação de operações matemáticas avançadas tais como: derivação, integração, entre outras.
- Criação de funções estadísticas.

Atualmente o MATVOX está disponível só na versão portuguesa, pelo qual seria interessante desenvolver um sistema que permitirá configurar o aplicativo para outros idiomas como o inglês e o espanhol.

Referências Bibliográficas

COMPILERS.NET. Gramáticas, Compiladores, Metacompiladores. Disponival em: http://www.compilers.net. Acesso em: 20 Mar. 2009.

CUEVA, Juan Manuel. Conceptos Basicos de Procesadores de Lenguaje. Universidad de Oviedo. ESPAÑA. Disponível em: http://www.di.uniovi.es/procesadores/Apuntes/ConceptosBasicos/10_Conceptos_Basicos_Procesadores_Lenguaje.pdf. Acesso em: 06 Mar. 2009.

DOSVOX. Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro. Disponível em: <intervox.nce.ufrj.br/dosvox>. Acesso em: 10 Mar. 2009.

EBERLIN, Samer. O Software Livre como Alternativa para a Inclusão Digital do Deficiente Visual. Universidade Estadual de Campinas. BRASIL. Disponivel em: http://www.decom.fee.unicamp.br /~samer/files/SamerEberlin-Mestrado.pdf. Acesso em: 10 Jun. 2009.

ELLOA B, Guedes da Costa; IGHOR O, do Rego Barros; FECHINE, Joseana. Matraca – Ferramenta Computacional para Auxilio a Deficientes Visuais no Uso do Computador. Universidade Federal de Campina Grande. BRASIL. Disponível em: http://www.dsc.ufcg.edu.br/~pet/atividades/Artigos/ARTIGO_MATRACA.pdf. Acesso em: 10 Apr. 2009.

FREEDOM SCIENTIFIC. JAWS - Job Access With Speech. Disponível em: <http://www.freedomscientific.com/fs_products/JAWS_HQ.asp>. Acesso em: 04 Out. 2009.

GALVEZ, Sergio; MORA MATA, Miguel Angel. Compiladores. Traductores y Compiladores con lex/yacc, jflex/cup y javacc. . Disponível em: http://www.lcc.uma.es /~galvez/ ftp/libros/Compiladores.pdf. Acesso em: 06 Mar. 2009.

GARCIA, Horacio. Reconocedor de Lenguajes con base em Gramaticas Formales. Instituto Politecnico Nacional. Mexico DF, MEXICO. Disponível em: http://itzamna.bnct.ipn.mx: 8080/dspace/bitstream/123456789/756/1/1341_2007_UPIICSA_MAESTRIA_garcia_salas_horacioalberto.pdf. Acesso em: 15 Apr. 2009.

KEMENY, John G; KURTZ, Thomas E. True BASIC Reference Manual. True BASIC Inc. Hanover, USA.

KOWALTOWSKI, Tomasz. Implementação de Linguagens de Programação. Universidade Estadual de Campinas. BRASIL.

MBROLA. The MBROLA Project - Towards a Freely Available Multilingual Speech Synthesizer. TCTS Lab of the Faculté Polytechnique de Mons (Belgium). Disponível em: <http://tcts.fpms.ac.be/synthesis/mbrola.html>. Acesso em: 04 Out. 2009.

O'BRIEN, Stephen. Turbo PASCAL 6. McGraw – Hill, Inc. BRASIL.

PIMENTEL, Marcelo. EDIVOX – Editor de Textos para Deficientes Visuais. Universidade Federal de Rio de Janeiro. BRASIL.

RYAN, Stansifer. Compiler Construction. Department of Computer Sciences Florida Institute of Technology Melbourne, Florida USA. Disponível em: http://www.cs.fit.edu/ ~ryan/cse4251/ chapter3.pdf. Acesso em: 25 Mar. 2009.

WIKIPEDIA. The Free Encyclopedia. Provided by Wikimedia Foundation. Disponível em: <http://en.wikipedia.org/wiki/Accessibility>. Acesso em: 04 Out. 2009.

WIKIPEDIA. The Free Encyclopedia. Provided by Wikimedia Foundation. Disponível em: <http://es.wikipedia.org/wiki/Compilador>. Acesso em: 06 Mar. 2009.

WIKIPEDIA. The Free Encyclopedia. Provided by Wikimedia Foundation. Disponível em: <http://es.wikipedia.org/wiki/Síntesis_de_habla>. Acesso em: 04 Out. 2009.

Apêndice A – Manual de usuário

Lista de Figuras.

Lista de Tabelas

Lista de Exemplos

1. Introdução

O MATVOX é um aplicativo voltado para deficientes visuais, que permite o desenvolvimento de algoritmos matemáticos utilizando a plataforma DOSVOX. Ele caracteriza se pela sua facilidade de uso e pelas suas características que permitem seu uso nas mais diversas áreas das ciências exatas.

As principais características do MATVOX são:

- Uso do editor de textos EDIVOX, o qual está especialmente voltado para pessoas com deficiência visual.
- Linguagem de instruções muito similar às linguagens de programação comerciais (C, BASIC, PASCAL, JAVA).
- Todo o necessário para a escrita e execução dos algoritmos está incluído no EDIVOX (com o MATVOX instalado), o qual evita a necessidade de utilizar programas externos a este.
- Inserção de resultados na mesma tela do editor onde está escrito o algoritmo.
- Léxico pouco rigoroso com a escrita do código, o qual diminuiu a geração de erros no algoritmo.
- Sistema de identificação de erros muito descritivo, o qual facilita a correção dos mesmos.
- Menus interativos que permitem uma comunicação muito amigável entre o programa e o usuário.
- Sistema de ajuda para a fácil aprendizagem da linguagem.
- Sistema inteligente de fala de sentenças, o qual permite ter um melhor entendimento do código.
- Grande variedade de instruções e funções.
- Mensagens faladas poucos cansativos para o usuário.

Para a correta execução do MATVOX é necessário ter instalado no computador o sistema DOSVOX Versão 4.1. Nas seções seguintes será explicado o procedimento passo a passo para a instalação do DOSVOX e do MATVOX.

2. Instalação do DOSVOX

Primeiro que todo o DOSVOX é um aplicativo gratuito que pode ser baixado da internet livremente. Para a instalação do aplicativo pode se realizar o seguinte procedimento.

A. Baixar o aplicativo.

Ele pode ser obtido no site http://intervox.nce.ufrj.br/dosvox/download.htm**.** Atualmente a ultima versão disponível do DOSVOX e a numero 4.1, esse módulo tem aproximadamente 112 MB e contem todos os programas do DOSVOX.

B. Instalação do aplicativo.

Para instalar o aplicativo basta com executar o arquivo *dv41-setup.exe* e a versão completa do DOSVOX será instalada.

Em Windows Vista, XP ou NT é necessário ter privilégios de administrador para instalar completamente o DOSVOX. Se a instalação for realizada numa conta sem privilégios, não será possível ativar alguns itens do DOSVOX.

3. Instalação do MATVOX

Como requisito para instalar e utilizar o MATVOX, o DOSVOX tem que ter sido instalado previamente no sistema, já que o MATVOX é executado sobre esta aplicação. Para instalar corretamente o MATVOX é necessário:

A. Obter o arquivo de instalação *mt21-setup.zip*.

B. Extrair seu conteúdo.

O arquivo *mt21-setup.zip* vai ter os seguintes arquivos (Fig. 3.1):

- Arquivo **EDIVOX.EXE**: é o arquivo executável do editor de textos do DOSVOX. O executável incluído neste arquivo tem a ferramenta MATVOX incluída nele.
- Pasta **SOM:** contem todos os arquivos sonoros e de configuração, necessários para a correta execução do MATVOX.

Fig. 3.1: Arquivos de instalação do MATVOX.

Estes arquivos têm que ser extraídos na pasta de instalação do DOSVOX, geralmente está pasta fica no endereço C:\winvox, sem embargo, se ela está em outro endereço, o importante é que os arquivos extraídos fiquem copiados na pasta de instalação do DOSVOX. Durante o

processo de copiado, o Windows vai perguntar se deseja substituir o arquivo EDIVOX.EXE, neste ponto tem que ser escolhido que SIM (Fig. 3.2).

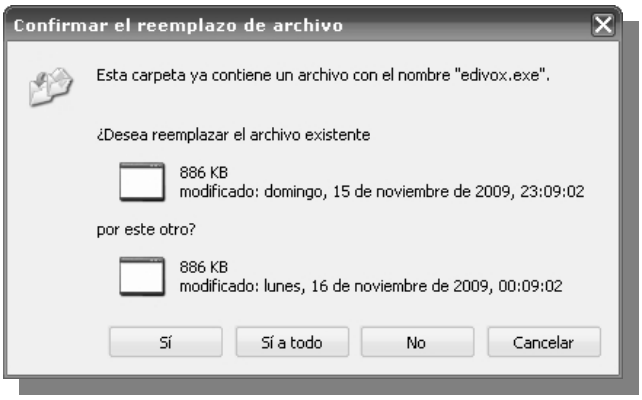


Fig. 3.2: Instalação do MATVOX, substituição de arquivos.

C. Verificação da instalação.

A verificação da instalação do MATVOX será explicada mais adiante.

4. Inicialização do DOSVOX

Existem duas formas de executar o programa, a primeira consiste em configurar o programa para que inicie automaticamente com a inicialização do sistema operativo (Microsoft Windows), e a segunda é através do pressionamento simultâneo das teclas Ctrl + Alt + D .

Quando o DOSVOX é inicializado ou usuário ouvira a frase de diálogo "*Sistema DOSVOX, versão 4.1, Núcleo de computação da URFJ, bom dia"*, e posteriormente a frase *"DOSVOX, O que você deseja?"* (Fig. 4.1). Esta frase será ouvida na tela principal do DOSVOX quando o programa necessite que o usuário insira alguma informação que permita executar alguma ação. A partir desse momento o programa fica pronto para que o usuário possa comandar de forma interativa o computador.

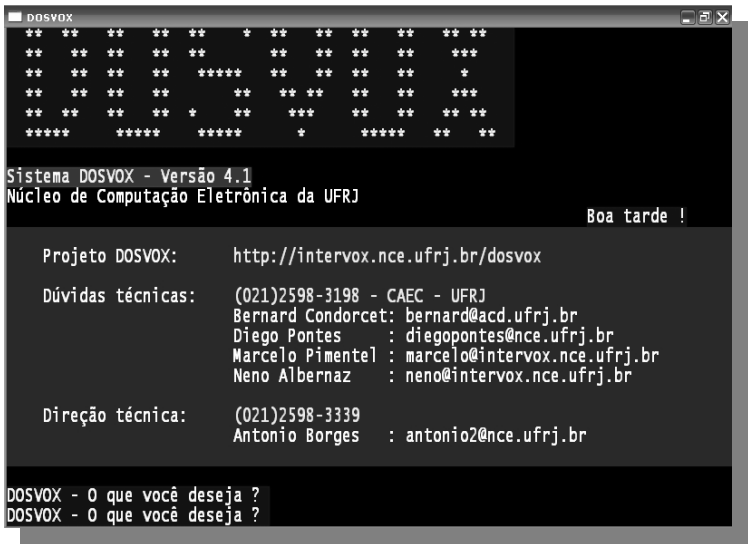


Fig. 4.1: Tela inicial do DOSVOX

4.1. Ajuda e execução de aplicativos no DOSVOX

Para obter ajuda no DOSVOX é utilizada a tecla F1, quando ela é pressionada, são apresentados os comandos disponíveis do aplicativo (Fig. 4.2).

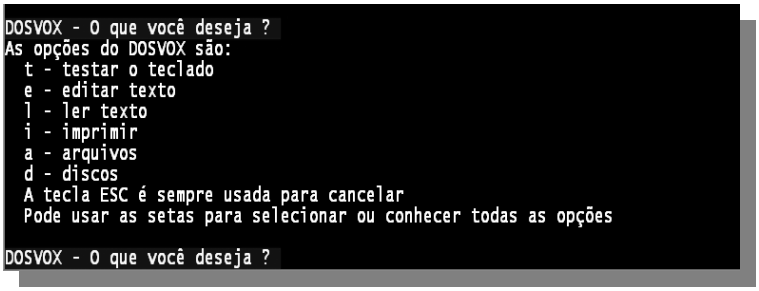


Fig. 4.2: Lista de opções apresentadas quando é utilizada a tecla F1 no DOSVOX

Por outra parte as setas PARA CIMA e PARA BAIXO são utilizadas para apresentar e selecionar cada um dos aplicativos disponíveis no DOSVOX. A tecla ENTER é usada para executar e a tecla ESC para cancelar (Fig. 4.3).

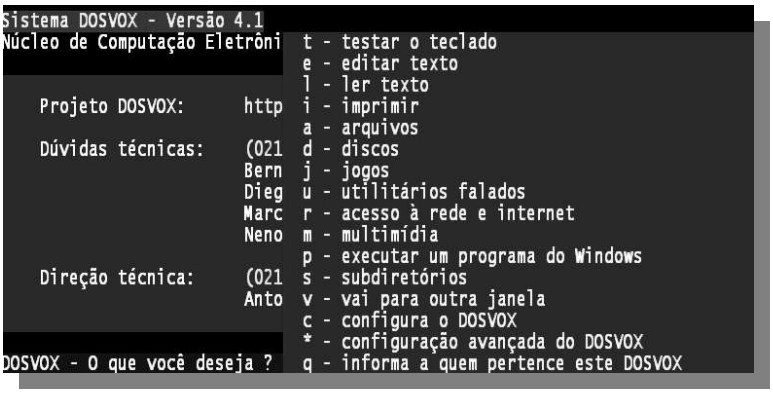


Fig. 4.3: Menu de aplicativos do DOSVOX.

Outra forma de executar aplicativos é pressionando a tecla de atalho correspondente, direitamente na tela do DOSVOX (ou seja, na linha: "*DOSVOX – O que você deseja?*"), estas teclas de atalho podem ser vistas na Fig. 4.3. Por exemplo, para editar um texto seria pressionada a tecla E, a qual ativará a ferramenta (EDIVOX) destinada para realizar está ação.

4.2. Saída do DOSVOX

O DOSVOX pode ser fechado pressionando a tecla F ou a tecla ESC, feito isto será ouvida uma mensagem de despedida "*Confirma o fim do DOSVOX, sim ou não*", neste ponto o usuário tem as seguintes opções:

Teclas S ou ENTER para finalizar o DOSVOX.

Teclas N ou ESC para retornar à tela principal do DOSVOX.

Se o usuário escolher a opção de finalizar o DOSVOX apresentara as seguintes opções:

Tecla D para desligar o computador.

Tecla L para finalizar a sessão (logoff) no Windows.

Tecla W para retornar a Windows.

Tecla R para reiniciar o Windows.

5.Execução do EDIVOX

Para executar o EDIVOX primeiro é necessário estar na janela principal do DOSVOX onde é feita a pergunta "O que você deseja?", e pressionar a tecla E, o qual imediatamente apresentará a janela principal do EDIVOX. Outra forma de aceder ao EDIVOX é mediante os menus interativos do DOSVOX, para isso, basta só com pressionar na janela principal do DOSVOX, as setas PARA CIMA ou PARA BAIXO e a continuação por médio das mesmas setas, procurar e selecionar a opção "*E – Editar texto*", uma vez encontrada a opção, é pressiona a tecla ENTER o qual executará o EDIVOX (Fig. 5.1).

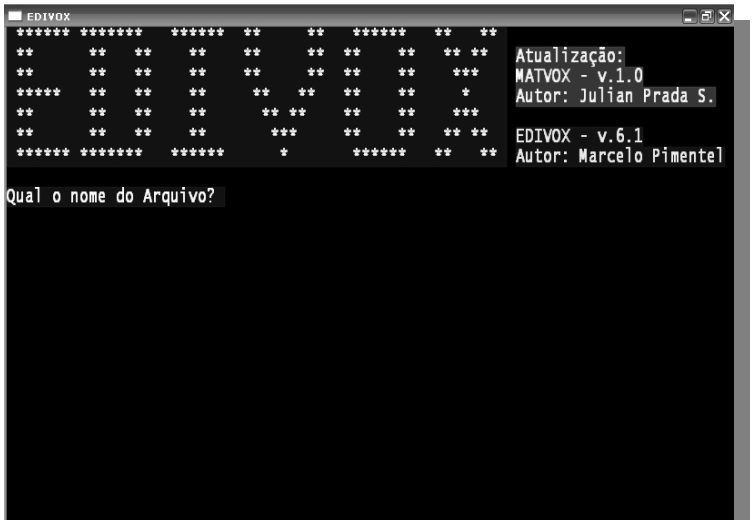


Fig. 5.1: Tela principal do EDIVOX.

5.1. Criar ou carregar arquivos

Quando o EDIVOX é executado (Fig. 5.1), ele vai falar "*EDIVOX – Qual o nome do arquivo?*", nesse momento o usuário tem a opção de criar um arquivo novo (arquivo de texto com extensão TXT) ou carregar um já existente. Para criar um arquivo novo, tem que ser inserido um

nome de até 256 caracteres e finalizar com a tecla ENTER, por outra parte para carregar um arquivo já existente basta com escrever o nome do arquivo ou procurar ele mediante as setas PARA CIMA ou PARA BAIXO (Fig. 5.2), e a continuação, pressionar a tecla ENTER para entrar na janela de edição.

Fig. 5.2: Carga de arquivos no EDIVOX.

O EDIVOX apresenta a mensagem "*Arquivo novo*", quando o arquivo é novo e "*Arquivo carregado*" quando é um arquivo já existente.

5.2. Criação e edição de texto

O processo de criação e edição de texto no EDIVOX é muito similar ao processo realizado nos editores convencionais, sem embargo EDIVOX incorpora características sonoras, especiais para pessoas com deficiência visual. Estas características permitem ao usuário, entre outras coisas, reproduzir sonoramente todo o que é escrito no editor.

As principais ações que o usuário pode executar com as diferentes teclas do teclado são a seguintes:

Movimento do cursor:

Seta para ESQUERDA: Move o cursor um caractere para a esquerda. Ao chegar ao primeiro caractere da linha, o programa dá um bip. Por outra parte o caractere, pelo qual o cursor passou será falado.

Seta para DIREITA: Move o cursor um caractere para a direita. Ao chegar ao ultimo caractere da linha, o programa dá um bip. Por outra parte o caractere, pelo qual o cursor passou será falado.

Seta para CIMA: Move o cursor uma linha para cima. Ao chegar à primeira linha, o programa fala "*Inicio do texto*". Por outra parte o texto da linha à qual o cursor chega será falada e o cursor fica posicionado na coluna 1.

Seta para BAIXO: Move o cursor uma linha para baixo. Ao chegar à ultima linha, o programa fala "*Fim do texto*". Por outra parte o texto da linha à qual o cursor chega será falada e o cursor fica posicionado na coluna 1.

HOME: Posiciona o cursor na coluna 1 da linha.

END: Posiciona o cursor após da ultima coluna escrita da linha.

ENTER: Se o cursor está na coluna 1, ela insere uma nova linha embaixo da linha atual e fala "*Linha nova*", e o cursor fica na coluna 1. Por outra parte, se ao pressionar a tecla ENTER o cursor está numa coluna diferente à coluna 1, a linha não é quebrada e o cursor é movido para a linha de embaixo, ficando na coluna 1.

PAGE UP: Volta 15 linhas de texto, o cursor fica na coluna 1.

PAGE DOWN: Avança 15 linhas de texto, o cursor fica na coluna 1.

Remoção de caracteres indesejáveis:

BACKSPACE: Remove o caractere a esquerda do cursor.

DEL: Remove o caractere na posição do cursor.

CTRL + Y ou F7: Remove a linha completa na qual está o cursor, e fala "*Linha removida*".

CTRL + BACKSPACE ou CTRL + H: Recupera a ultima linha apagada e fala "*Linha nova*".

CTRL + D: Apaga toda uma palavra, e soletra a palavra apagada.

CTRL + S: Apaga do inicio da linha até a posição do cursor e fala "*Apagado a esquerda*".

Leitura das linhas:

F1: Pressionando a tecla F1 várias vezes cada palavra da linha atual é falada, depois de ler a ultima palavra da linha soa um bip.

CTRL + F1: Fala a linha inteira a partir do ponto onde o cursor está. Para interromper a leitura pode ser pressionada qualquer tecla exceto as teclas CTRL, ALT, ou WINDOWS.

ALT + F1: Fala o resto do texto a partir do ponto onde o cursor está. Para interromper a leitura pode ser pressionada qualquer tecla exceto CTRL, ALT, ou WINDOWS.

F4: Ativa ou desativa a fala durante a digitação, as falas dos comandos continuam ativas.

Inserção de novas linhas:

ENTER: Se o cursor está na coluna 1, ela insere uma nova linha embaixo da linha atual e fala "*Linha nova*", e o cursor fica na coluna 1.

CTRL + ENTER ou CTRL + J ou CTRL + M: Insere uma linha nova em cima da linha onde o cursor está, e fala "*Linha nova*".

INSERT: Ativa o desativa a possibilidade de inserir linhas com a tecla ENTER, e fala "*Enter vai inserir linha*" ou "*Enter não vai inserir linha*" conforme seja o caso.

CTRL + Q: Atua como um ENTER, mas permite quebrar uma linha em duas partes, fala "*Linha quebrada*".

Configuração das margens:

F10: Por defeito as margens do EDIVOX estão na coluna 1 e na coluna 72, mais elas podem ser mudadas com a tecla F10, ao pressionar F10 o EDIVOX vai falar "*Digite coluna da margem esquerda:*", nesse ponto o usuário deve teclar o número da coluna e pressionar ENTER, logo vai falar "*Digite coluna da margem direita*" e o usuário deve teclar o número da coluna e depois pressionar ENTER, depois o programa vai falar "*Margem adicionada*".

5.3. Arquivamento e saída do EDIVOX

Para sair do EDIVOX o usuário só tem que pressionar a tecla ESC, depois o programa vai perguntar pela confirmação da saída e vai a perguntar também se o usuário quer salvar o arquivo.

Algumas ações que podem ser executadas são as seguintes:

F2: Permite salvar o arquivo em qualquer momento e fala "Arquivo gravado".

CTRL + X: Grava o arquivo e termina o EDIVOX sem perguntar.

CTRL + F2: Permite salvar o arquivo com outro nome.

6.Programação com MATVOX

O MATVOX está acoplado ao EDIVOX, o qual quer dizer que para criar e executar algoritmos matemáticos não é necessária a utilização de programas ou aplicativos externos a ele. Quando o MATVOX é instalado, ele é automaticamente aderido ao EDIVOX e fica pronto para rodar.

A correta instalação do MATVOX pode ser verificada de duas maneiras, a primeira consiste em verificar na esquina superior direita da tela do EDIVOX a existência da seguinte imagem (Fig. 6.1).

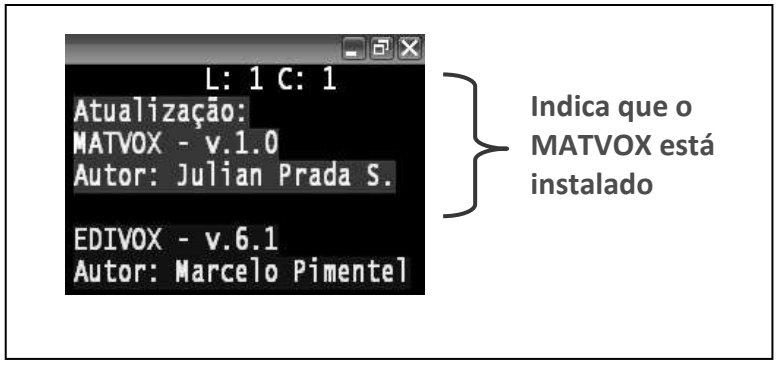


Fig. 6.1: Verificação de instalação do MATVOX.

A segunda forma consiste em pressionar as teclas CTRL + F12, o qual permitira ouvir a seguinte frase "*EDIVOX 6.1*" e logo "*Atualização Calculadora Programável MATVOX Versão 1.0*", isto indica que o MATVOX está corretamente instalado, se ele não estiver só seria ouvido "*EDIVOX 6.1*".

Neste capitulo serão apresentadas as regras, características e instruções gerais do MATVOX, tais como, variáveis, operadores, estruturas de programação, funções matemáticas, etc. as quais são muito similares a algumas das estruturas de outras linguagens de programação como PASCAL, BASIC, JAVA, C/C++ entre outros.

6.1. Criação e execução do programa “Oi Mundo!!!” no MATVOX

No MATVOX um programa ou algoritmo tem basicamente duas partes, uma parte de código e outra parte de resultados, os quais são inseridos imediatamente depois da ultima linha de programação.

A continuação é apresentado um programa básico, o qual imprime na secção de resultados o texto “Oi Mundo!!!”, o código fonte é o seguinte:

Código:

```
\BEGIN;
  \Print{"OI"+"MUNDO!!!"};
\END;
```

Resultados:

```
OI MUNDO!!!
```

Para executar um algoritmo no EDIVOX tem se que realizar o seguinte procedimento:

A. Escrever o código desejado. Fig. 6.2.

Fig. 6.2: Passo 1 na execução de um algoritmo no EDIVOX, escritura do código.

B. Posicionar o cursor: para executar o programa o cursor tem que ser posicionado em qualquer linha do código. A linha na qual está o cursor fica de cor amarelo (Fig. 6.3).

Fig. 6.3: Passo 2 na execução de um algoritmo no EDIVOX, posicionamento do cursor

C. Pressionar simultaneamente as teclas CTRL + F10: ao pressionar estas teclas, será apresentado um menu *Pop-up* do MATVOX na tela do EDIVOX, e será ouvida a mensagem *"Comandos da calculadora"* (Fig. 6.4).

Fig. 6.4: Passo 3 na execução de um algoritmo no EDIVOX, execução do menu.

D. Pressionar a tecla F2: ao pressionar esta tecla, o programa será executado e os resultados serão inseridos no EDIVOX. Também será ouvido um indicador sonoro ao final da execução, dependendo do resultado desta, neste caso será ouvida a mensagem, *"Execução correta"* (Fig. 6.5).

Fig. 6.5: Passo 4 na execução de um algoritmo no EDIVOX, algoritmo executado.

E. Tecla ESC: está tecla pode ser utilizada durante a execução do algoritmo, a qual vai abortar imediatamente a execução do mesmo e será ouvida a mensagem *"Execução abortada."*.

6.1.1. Sinais sonoras na execução de algoritmos.

O MATVOX implementa uma serie de mensagens e sinais sonoras que permitem ao usuário identificar em todo momento o estado atual da execução do algoritmo.

Quando um programa tarda um tempo considerável na sua execução o usuário pode perceber o som "*beep*, *beep*, *beep*, *beep*....", o qual indica que a execução do algoritmo não tem finalizado.

Se o usuário em determinado momento desejar silenciar este som, mas não parar a execução do programa, então só tem que pressionar a tecla ENTER. Para ativar novamente o som, tem que pressionar a tecla ENTER de novo.

O Exemplo 6.1 tarda um tempo considerável na sua execução pelo qual o usuário vai poder identificar está sinal de som.

```
\ BEGIN;                                    %% Inicio do programa
  \ FOR { i=0:999999 };
  \ CONTINUE;
```

```
\ END;                                          %% Fim do programa
```

Exemplo 6.1:Exemplo para testar os sons de execução do MATVOX.

Quando finaliza a execução de um algoritmo podem ser gerados dois tipos de mensagem:

***Execução correta*:** significa que o programa foi executado corretamente.

***Erro na execução do algoritmo*:** significa que o algoritmo tem algum problema na sua programação.

***Execução Abortada*:** quando o usuário aborta o programa.

6.1.2. Mensagens de erro

Quando o algoritmo ou programa tem algum problema na sua escrita ou nos seus parâmetros vai ser gerado um erro, no momento em que seja executado. Esta mensagem de erro está composta por três linhas ou partes (Exemplo 6.2).

```
\BEGIN;
  \Print { A };
\END;

#E77 Erro: Comando PRINT, não válido, variável não encontrada.
Parâmetro errôneo: A
Linha: 2
```

Exemplo 6.2: Exemplo da mensagem de erro no MATVOX.

A primeira linha, ***#E77 Erro:** Comando PRINT, não válido, variável não encontrada*, apresenta o número e a descrição do erro, neste caso está indicando que existe uma variável que não foi definida no programa.

A segunda linha, ***Parâmetro errôneo: A***, apresenta o parâmetro (variável, identificador, operador, caractere, etc.) que está errado. Neste caso indica que a variável A não foi definida.

A terceira linha, ***Linha: 2*** apresenta o número da linha na qual encontra se o erro. Neste caso a variável A, definida na instrução *Print* está na linha 2.

Simultaneamente às mensagens escritas, também são ouvidas algumas mensagens sonoras, as quais neste caso são as seguintes:

"Erro, comando Print, não válido, variável não encontrada"
"Parâmetro errôneo, A"
"Linha 2"
"Erro na execução do algoritmo"

6.2. Menus Pop – Up

O MATVOX implementa uma serie de menus interativos, com os quais o usuário pode criar de forma fácil e amigável algoritmos no EDIVOX. Estes permitem inserir instruções, funções, constantes, etc. de forma iterativa no programa.

Os menus são ativados desde a tela principal do EDIVOX mediante as teclas CTRL + F10.

Para navegar entre os diferentes menus são utilizadas as seguintes teclas:

Setas para a DIREITA e para a ESQUERDA: permite navegar entre os diferentes menus disponíveis no MATVOX.

Setas PARA ARRIBA e PARA ABAIXO: permite navegar entre as diferentes opções de cada um dos menus do MATVOX.

Tecla F1: permite ouvir informação de ajuda ou de descrição, em cada uma das opções de cada menu.

Tecla ESC: permite sair do menu e voltar à tela principal do EDIVOX.

Tecla ENTER: permite escolher e inserir os dados da opção desejada.

Quando é selecionada e escolhida alguma opção do menu, o programa vai fazer de forma interativa uma serie de perguntas dependendo da instrução, variável, função ou constante selecionada. Por exemplo, se é escolhida a instrução *VarResult*, o programa vai perguntar o seguinte:

- Pergunta: "*Introduza o nome da variável que deseja imprimir*"

Nesta parte o usuário tem que escrever o nome da variável e pressionar a tecla ENTER. Se ele desejar ouvir a pergunta novamente, só tem que pressionar a tecla F1. Para finalizar o menu sem inserir nenhum dado é pressionada a tecla ESC. Depois de fazer isto o programa vai perguntar o seguinte:

- Pergunta: "Deseja imprimir mais variáveis, se ou não? (S/N)"

Neste tipo de perguntas o usuário tem que escolher entre as diferentes opções, (neste caso tecla S para sim e a tecla N para não). Se o usuário desejar cancelar a pergunta só tem que pressionar a tecla ESC.

Quando as perguntas finalizam, o programa vai falar "*Comando Inserido!!!"* e posteriormente a instrução é inserida na tela do EDIVOX com todos seus dados, neste caso seria inserida a seguinte linha:

\ VarResult { <nome da variável> , <nome da variável > .... **};**

6.2.1. Casos especiais na inserção de instruções

Na maioria de instruções, o sistema de inserção de parâmetros funciona da mesma maneira como foi apresentada na seção anterior. Sem embargo algumas instruções têm algumas características que necessitam ser explicadas por separado.

Instruções *ImportText, ExportText, ImportPlanivox e ExportPlanivox.*

Estas instruções são utilizadas para importar e exportar dados desde e para arquivos de tipo texto, pelo qual quando eles vão ser definidos é indispensável conhecer com anterioridade as características dos arquivos de texto que vão ser utilizados.

Para conhecer estas características o MATVOX implementa nos menus destas instruções, um mini leitor de arquivos texto (Fig. 6.6), o qual evita ter que abrir telas adicionais no sistema.

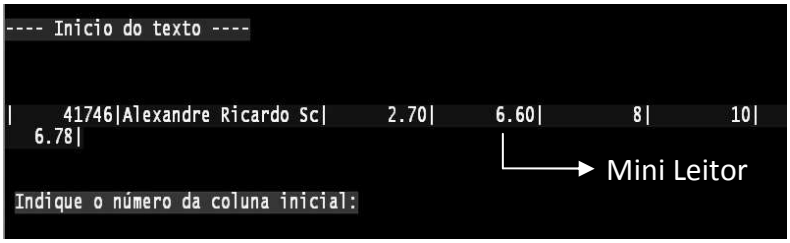


Fig. 6.6: Passo 4 na execução de um algoritmo no EDIVOX, algoritmo executado.

Este mini leitor de texto funciona da seguinte maneira:

No menu de comandos quando é selecionada qualquer destas instruções, o programa vai perguntar pelo nome do arquivo e apresenta uma listagem com os arquivos do diretório atual (no caso de *ExportText* e *ExportPlanivox*, ele da a possibilidade de criar um arquivo novo), neste ponto o usuário tem que selecionar um deles. Quando o arquivo é selecionado, imediatamente é executado o mini leitor de textos (Fig. 6.6). Este mini leitor é utilizado para escolher a posição no texto, na qual o usuário vai exportar os dados (no caso das instruções ExportText e

ExportPlanivox) ou a posição desde a qual estes vão ser importados (no caso das instruções ImportText e ImportPlanivox).

No mini leitor são utilizadas as seguintes teclas de controle:

Setas PARA CIMA e PARA ABAIXO: permite navegar verticalmente no texto.

Setas PARA DIREITA e PARA ESQUERDA: permite navegar horizontalmente no texto.

Tecla F2: fala o número da linha e coluna atual do cursor.

Tecla F3: retrocede o cursor 10 posições para a esquerda.

Tecla F4: retrocede o cursor 5 posições para a esquerda.

Tecla F5: avança o cursor 5 posições para a direita.

Tecla F6: avança o cursor 10 posições para a direita.

Tecla ENTER: seleciona e insere a posição atual do cursor.

Tecla ESC: fecha o mini leitor sem selecionar nenhuma posição.

6.2.2. Descrição de cada um dos menus.

Os menus que são encontrados no MATVOX são os seguintes:

Menu COMANDOS: contem cada uma das instruções disponíveis no MATVOX (BEGIN, IF, WHILE, FOR, END, etc.).

Menu VARIÁVEIS: contem os diferentes tipos de variáveis que podem ser definidos no MATVOX (numéricas, Sting, arrays, etc.).

Menu FUNÇÕES GERAIS: contem as funções básicas do MATVOX (Logaritmo natural, Valor Absoluto, Raiz quadrada, Exponencial, etc.).

Menu FUNÇÕES TRIGONOMÉTRICAS BÁSICAS: contem as principais funções trigonométricas (Seno, Cosseno, Tangente, etc.) em radianos e graus.

Menu FUNÇÕES TRIGONOMÉTRICAS BÁSICAS INVERSAS: contem as funções trigonométricas inversas (Arco Seno, Arco Cosseno, Arco Tangente, etc.) em radianos e graus.

Menu FUNÇÕES TRIGONOMÉTRICAS HIPERBÓLICAS: contem as funções trigonométricas hiperbólicas (Seno hiperbólico, Cosseno hiperbólico, Tangente hiperbólica, etc.) em radianos e graus.

Menu FUNÇÕES TRIGONOMÉTRICAS HIPERBÓLICAS INVERSAS: contem as funções trigonométricas hiperbólicas inversas (Arco Seno hiperbólico, Arco Cosseno hiperbólico, Arco Tangente hiperbólica, etc.) em radianos e graus.

Menu CONVERSÕES: contem algumas funções que permite fazer conversões entre diferentes unidades de medida (metros para pés, litros para onças, graus Kelvin para graus Centigrados, etc.).

Menu CONSTANTES FÍSICAS FUNDAMENTAIS: contem uma serie de constantes fundamentais da física tais como: Velocidade da luz, Constante de Avogadro, numero PI, Unidade de massa Atômica, etc.

Menu CONSTANTES DE USUÁRIO: contem as constantes que o usuário tem armazenado no sistema.

Menu EXPRESSÕES DE USUÁRIO: contem as expressões que o usuário tem armazenado no sistema.

6.3. Fala inteligente de algoritmos.

O MATVOX implementa uma opção que permite ao usuário ouvir de forma detalhadamente cada uma das linhas do algoritmo no EDIVOX. Para isso, só é necessário posicionar o cursor na linha desejada e posteriormente pressionar as teclas CTRL + F9.

Normalmente quando o cursor é posicionado em qualquer linha de código, o EDIVOX lê literalmente o que está escrito, por exemplo, na linha de código:

```
\ VarResult { A , B } ;
```

O EDIVOX vai falar "*barra invertida, varresult, abre chave, a, b, fecha chave*", pelo que este tipo de fala pode ser pouco claro e cansativo para o usuário que está programando. Sem embargo, quando o modo de fala inteligente é ativado (CTRL + F9) o EDIVOX vai ler a mesma linha de código da seguinte maneira: "*Comando resultado variável, serão impressas a variáveis: variável A, variável B*".

Por outra parte se a linha tiver algum tipo de erro (neste caso o erro está no caractere #), por exemplo:

```
\ VarResult { A , B# } ;
```

O programa vai falar: "*Comando resultado variável, serão impressas a variáveis: variável A, variável B number, variável a imprimir não valida, caráter number na variável não permitido*".

Em conclusão, este modo de fala aparte de lhe ajuda ao usuário o entendimento do algoritmo, permite também encontrar de forma fácil e detalhada os diferentes erros que possa chegar a ter o programa.

6.4. Regras gerais de criação de código

O MATVOX tem algumas características e regras gerais na linguagem, as quais têm que ser respeitadas e adotadas para o correto desenvolvimento dos algoritmos.

No MATVOX os espaços em branco, nas variáveis, funções, expressões, comentários, etc. são irrelevantes, ou seja, uma variável com nome **MASSA** tem o mesmo significado que a variável com nome **MA S S A**.

As letras em maiúscula ou minúscula também são irrelevantes, pelo qual as variáveis **Massa** e **MASSA** têm o mesmo significado.

Para definir números decimais pode ser utilizado o caractere ponto ou o caractere virgula, pelo qual os números **3,14156** e **3.14156** tem o mesmo valor.

Toda linha de programação no MATVOX tem que finalizar com o caractere ponto e vírgula a exceção dos comentários e das expressões. As expressões podem ser definidas em varias linhas de código, por exemplo, a expressão:

```
A = ( 5 + 4 * 7 ) / 8 ;
```

Pode ser escrita também desta forma:

```
A = ( 5 + 4 * 7 )
/ 8 ;
```

6.5. Variáveis

Uma variável em MATVOX é um espaço reservado no ordenador para conter valores, os quais podem mudar durante a execução do programa.

As variáveis podem ser criadas com muita liberdade, elas podem conter caracteres numéricos e alfanuméricos, e alguns caracteres especiais. Sem embargo, há que ter em conta que o nome da variável sempre tem que iniciar com uma letra ou uma vocal. Os caracteres permitidos são os seguintes:

- Caracteres de A – Z.
- Caracteres de 0 – 9.
- Caracteres: Abre colchete – " [" , Fecha colchete – "] ", Sublinhado – " _ ",

Por outra parte MATVOX contem uma serie de palavras reservadas, as quais têm um significado especial e por tanto não podem ser utilizadas para representar variáveis, estas palavras são constantes físicas as quais estão definidas dentro do programa. Elas podem ser chamadas, mas não podem ser modificas.

Também há que ter em conta que o MATVOX permite ao usuário armazenar constantes próprias no sistema, devido a isto as constantes que estão armazenadas, ficam reservadas enquanto estas não sejam removidas do sistema.

A continuação no Exemplo 6.3 e Exemplo 6.4 são apresentados vários casos, nos quais são definidas algumas variáveis de forma correta e incorreta.

```
A1 = 1 ;          < Variável definida com um valor inteiro>
Numero = 3,143 ;  < Variável definida com um valor de ponto flutuante >
b1A = 4E+4 ;      < Variável definida da forma exponencial ( notação cientifica ) >
Ex = 5E-5 ;       < Variável definida da forma exponencial ( notação cientifica ) >
```

Exemplo 6.3: Definição correta de variáveis.

```
1A = 1 ;          < A variável inicia com um caractere numérico >
N#1 = 3 ;         < A variável tem um caractere não válido " # " >
PI = 45,4 ;       < A variável "PI" está reservada pelo sistema >
```

Exemplo 6.4: Definição incorreta de variáveis.

6.5.1. Tipo de variáveis.

No MATVOX só existem dois tipos fundamentais de variáveis, as variáveis *numéricas* e as variáveis tipo *String* ou tipo *texto*. Para definir uma variável como numérica só é necessário definir um nome, tal e como foi explicado anteriormente, sem embargo para defini-la como *String,* é obrigatório adicionar o identificador ***STR***, no inicio do nome da variável.

```
Nota = 3.34 ;                    < Variável Numérica >
STR Nome = Tiago ;               < Variável tipo STRING >
Sobrenome = cubas ;              < Erro. "cubas" não é um valor numérico >
```

Exemplo 6.5: Definição de Variáveis numéricas e Variáveis tipo STRING.

No Exemplo 6.5 estão definidas algumas variáveis numéricas e de tipo *String*, sem embargo a variável Sobrenome tem um erro na sua definição, já que quando ela é definida como numérica, não pode armazenar valores de tipo *String*.

6.5.2. Arrays unidimensionais.

Um array é um tipo de dado estruturado, o qual permite agrupar dados utilizando um mesmo identificador. Todos os elementos de um array têm que ser do mesmo tipo e são acedidos mediante um índice.

Para definir um array é utilizado o identificador **"V_N"** para dados numéricos e **"V_S"** para dados tipo *String*, e mediante colchetes é definido o índice do mesmo. No Exemplo 6.6 são definidos alguns arrays numéricos e tipo *String*.

```
V_N Ex [ 0 ] = 3.4 ;     < Array numérico "Ex", posição "0" >
V_N Ex [ 1 ] = 5 ;       < Array numérico "Ex", posição "1" >
V_S Ax [ 0 ] = A ;       < Array Tipo String "Ax", posição "0" >
V_S Ax [ 1 ] = B ;       < Array Tipo String "Ax", posição "1" >
```

Exemplo 6.6: Definição de Arrays Unidimensionais.

6.5.3. Arrays bidimensionais.

Os arrays bidimensionais permitem agrupar dados em duas dimensões e são definidos mediantes os identificadores **"M_N"** para dados numéricos e **"M_S"** para dados tipo *String*. Seu direcionamento é feito mediante dois índices os quais são separados mediante o caractere vírgula. No Exemplo 6.7 são definidos alguns arrays bidimensionais numéricos e tipo *String*.

```
M_N Ex [ 0,0 ] = 4.4 ;      < Array numérico "Ex", posição "Coluna 0, fila 0" >
M_N Ex [ 1,1 ] = 8 ;        < Array numérico "Ex", posição "Coluna 1, fila 1" >
M_S Ax [ 0,1 ] = C ;        < Array Tipo String "Ax", posição "Coluna 0, fila 1" >
M_S Ax [ 1,0 ] = D ;        < Array Tipo String "Ax", posição "Coluna 1, fila 0" >
```

Exemplo 6.7: Definição de Arrays Bidimensionais.

6.6. Entrada de dados dinâmicos.

No MATVOX as variáveis podem ser definidas dinamicamente, de tal forma que quando o usuário executa o programa este vai perguntar pelo valor de cada uma destas. Este tipo de configuração permite criar algoritmos interativos com o usuário, já que seus parâmetros não são fixos e podem ser mudados facilmente entre uma execução e outra.

Para definir uma variável como dinâmica, só é necessário adicionar o identificador IN antes do nome desta.

```
IN Variavel ;
```

A continuação é apresentado um algoritmo (Exemplo 6.8) no qual as variáveis A, B, C são definidas de forma dinâmica e posteriormente são impressas na tela do EDIVOX.

```
\ BEGIN;                    %% Inicio do programa
  IN A;                     %% Variavel dinâmica
  IN B;                     %% Variavel dinâmica
  IN C;                     %% Variavel dinâmica
```

```
\ VarResult { A , B , C };
\ END;                                    %% Fim do programa
```

Exemplo 6.8:Variáveis dinâmicas.

Quando o algoritmo é executado, ele vai fazer as seguintes perguntas:

- *"Introduza o valor da variável A".*

Nesta parte o usuário tem que escrever o valor da variável que vai definir e posteriormente pressionar a tecla ENTER.

- Depois o programa vai perguntar pelas variáveis B e C.

6.7. Operadores

Os operadores são símbolos matemáticos que indicam que deve se realizar uma operação especifica sobre determinado número de operandos, os quais podem ser números, expressões, variáveis, etc. O MATVOX tem definidos dois tipos de operadores os aritméticos e os relacionais.

6.7.1. Operadores Aritméticos

Os operadores aritméticos são operadores binários, os quais necessitam sempre de dois operandos para sua correta operação. Eles fazem as operações básicas, SOMA, RESTA, MULTIPLICAÇÃO E DIVISÃO e permitem criar expressões.

No Exemplo 6.9 são apresentados expressões com operadores aritméticos.

```
A = B + 5 ;          < Operador SOMA >
A = B – 5 ;          < Operador RESTA >
A = B * 5 ;          < Operador MULTIPLICAÇÃO >
```

A = B **/** 5 ;	*< Operador DIVISÃO >*

Exemplo 6.9: Uso dos operadores aritméticos.

6.7.2. Operadores Relacionais

Os operadores relacionais são utilizados para definir expressões e para realizar comparações de igualdade, de desigualdade, de maior e de menor, estas são utilizadas em bifurcações e em loops, os quais serão estudados mais adiante. O resultado de estas comparações sempre é um valor ***True*** ou ***False*** dependendo do cumprimento ou não, da relação avaliada.

No Exemplo 6.10 são apresentados alguns casos da utilização de operadores relacionais em expressões e instruções.

A **=** B + 5;	*< Operador IGUAL em expressão >*
\IF {A **>** B};	*< Operador MAIOR no comando IF >*
\IF {A **<** B};	*< Operador MENOR no comando IF>*
\IF {A **>=** B};	*< Operador MAIOR ou IGUAL no comando IF >*
\IF {A **<=** B};	*< Operador MENOR ou IGUAL no comando IF >*
\IF {A **<>** B};	*< Operador DIFERENTE no comando IF >*
\IF {A = B};	*< Operador IGUAL no comando IF >*

Exemplo 6.10: Uso dos operadores relacionais.

7.Estruturas de programação

Nesta seção são apresentadas e analisadas as diferentes instruções e estruturas de programação do MATVOX, estas instruções têm um comportamento muito similar aos comandos encontrados em outras linguagens de programação tais como PASCAL, JAVA, C/C++, BASIC, etc.

O MATVOX conta com estruturas de programação ou de controle que permitem realizar ações tais como: tomar decisões, executar processos um determinado número de vezes, controlar o fluxo do programa, etc. Também conta com outros tipos de instruções, as quais tem diversas funcionalidades tais como: configurar e apresentar resultados ou dados ao usuário, armazenar ou apagar constantes ou expressões no sistema, avaliar funções, importar e exportar dados, etc. Todas estas instruções serão trabalhadas e explicadas nas seções seguintes.

7.1. Estrutura de um programa

No MATVOX, qualquer programa que vá ser criado, tem que ter uma determinada estrutura, a qual está definida por duas instruções, a instrução BEGIN e a instrução END.

A instrução BEGIN é utilizada para definir o inicio do programa, enquanto que a instrução END, indica o final do mesmo e no meio deles será definido o corpo do programa. Estas instruções só podem ser utilizadas uma só vez no programa, ou seja, no inicio e no final deste.

No Exemplo 7.1 é apresentada a estrutura geral, utilizada para criar qualquer programa.

```
\ BEGIN;                    %% Inicio do programa
                            %% Corpo do programa
\ END;                      %% Fim do programa
```

Exemplo 7.1: Estrutura geral de um programa.

7.2. Expressões

Uma expressão é um conjunto de variáveis, funções e/ou constantes unidas por operadores, os quais permitem definir uma formula matemática cuja avaliação especifica um valor numérico, uma expressão no MATVOX tem a seguinte sintaxe:

<*Variável dependente*> <*Operador Igual*> <*Expressão*>

No Exemplo 7.2 é definida uma expressão com variável dependente A.

```
\ BEGIN;                                          %% Inicio do programa
  A = ( 34 + ( ( 3– 5 ) * 5,4 ) + 6 ) * RSIN( 1 + 4 ) ;        %% Expressão
\ END;                                            %% Fim do programa
```

Exemplo 7.2: Definição de expressões.

7.3. Comentários

Os comentários são muitos úteis para organizar e entender o código utilizado, eles ajudam ao desenvolvedor a fazer futuras revisões e correções, pelo qual é importante e recomendável se acostumar a comentar o código desenvolvido. Alem disso, permite que qualquer pessoa diferente ao desenvolvedor original consiga entender rapidamente o código escrito.

No MATVOX os comentários podem ser definidos de duas formas, a primeira é mediante o uso dos caracteres **%%**, os quais têm que estar inseridos no inicio do comentário. Este tipo de definição faz que o texto que encontra se na parte direita dos caracteres **%%** seja ignorado pelo programa. A segunda forma de definir comentários é mediante o uso dos caracteres **% {** no inicio do comentário e dos caracteres **} %** no final do mesmo, o qual vai fazer que o texto que está no meio das chaves seja ignorado.

No Exemplo 7.3 são inseridos comentários em diferentes posições do algoritmo utilizando as duas formas de definição.

```
\ BEGIN;                                          %% Inicio do programa
%% Exemplo de comentários
  A %{ Variável A }% = B %{ Variável B }% + 2 ;   %% Expressão
\ END;                                            %% Fim do programa %%

Código correspondente

\ BEGIN;                                          %% Inicio do programa
  A = B + 2 ;                                     %% Expressão
\ END;                                            %% Fim do programa
```

Exemplo 7.3: Definição de comentários no algoritmo.

7.4. Saída de resultados

Todo programa necessita se comunicar com seu entorno, já seja para apresentar resultados ou informação, devido a isto, o MATVOX conta com três instruções as quais permitem imprimir variáveis, procedimentos ou informação.

7.4.1. Instrução VarResult.

A instrução VarResult, permite ao usuário apresentar na tela do EDIVOX o valor de uma ou mais variáveis de qualquer tipo, definidas previamente no programa. Estes valores são impressos na linha de embaixo do final do programa, ou seja, na linha seguinte da instrução END. Os resultados gerados por esta instrução são apresentados da seguinte forma:

< Nome da Variável > < Operador Igual > < Valor da Variável >

As variáveis que vão ser impressas podem ter qualquer valor, sem embargo elas tem que ter sido definidas previamente, ou seja, estar definidas nas linhas anteriores da instrução de

impressão. Se isto não é assim, o programa gerará um erro indicando que a variável a imprimir não foi encontrada.

A instrução VarResult é definida da seguinte forma:

```
\ VarResult { < Parâmetros > } ;
```

Na Tab. 7.1 é apresentada a estrutura dos parâmetros da instrução.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ VarResult { A , B, C } ;	A	Variável.
	<Vírgula>	Caractere de separação.
	B	Variável.
	<Vírgula>	Caractere de separação.
	C	Variável.

Tab. 7.1: Estrutura dos parâmetros da instrução VarResult.

No Exemplo 7.4 é definida uma expressão, e mediante a instrução VarResult, o resultado de esta, é impresso no EDIVOX.

```
\ BEGIN;                          %% Inicio do programa
  A = 2*5 +100;                        %% Expressão
  \VarResult { A };               %% Instrução de impressão
\ END;                            %% Final do programa

A = 110                                %% Resultado do programa
```

Exemplo 7.4: Uso da instrução VarResult.

Por outra parte, também é possível imprimir varias variáveis utilizando a mesma instrução, para isso só é necessário separar a variáveis com o caractere vírgula. No Exemplo 7.5 são impressas quatro variáveis utilizando a instrução *VarResult*.

```
\ BEGIN;                                  %% Inicio do programa
  A = 2;                                  %% Variável A
  STR B = Luiz;                           %% Variável tipo String "B"
  C = 0;                                  %% Variável C
  D = 1;                                  %% Variável D
  \VarResult { A , STR B , C , D };       %% Instrução de impressão
\ END;                                    %% Final do programa

A = 2                                     %% Resultado do programa
STRB = LUIZ                               %% Resultado do programa
C = 0                                     %% Resultado do programa
D = 1                                     %% Resultado do programa
```

Exemplo 7.5: Uso da instrução VarResult com múltiplas variáveis definidas nos seus parâmetros.

7.4.2. Erros comuns no uso da instrução VarResult

O Exemplo 7.6 apresenta o erro que é gerado quando o usuário intenta imprimir uma variável que não tem sido definida previamente no programa.

```
\ BEGIN;                                  %% Inicio do programa
  A = 2;                                      %% Variável A
  B = 3,4;                                %% Variável B
  D = 1;                                      %% Variável D
  \VarResult { A , B , C , D };           %% Instrução de impressão
\ END;                                    %% Final do programa

#E44 Erro: Comando VARRESULT não válido, variável não encontrada.
Parâmetro errôneo: C
Linha: 5
```

Exemplo 7.6: Erro na instrução VarResult, variável não definida.

O Exemplo 7.7 é uma variante do exemplo anterior, nele a variável foi definida, sem embargo, esta foi definida depois da instrução *VarResult*, pelo qual vai ser gerado o mesmo erro.

```
\ BEGIN;                                  %% Inicio do programa
  A = 2;                                  %% Variável A
  B = 3,4;                                %% Variável B
  D = 1;                                  %% Variável D
  \VarResult { A , B , C , D };           %% Instrução de impressão
  C = 0;                                  %% Variável C
\ END;                                    %% Final do programa

#E44 Erro: Comando VARRESULT não válido, variável não encontrada.
Parâmetro errôneo: C
Linha: 5
```

Exemplo 7.7: Erro na instrução VarResult, variável não definida.

7.4.3. Instrução PRINT

A instrução PRINT permite ao usuário imprimir texto e/ou variáveis mediante o uso do caractere de concatenação +. Nesta instrução as cadeias de texto têm que estar delimitadas por aspas, a diferença das variáveis, as quais não precisam de nenhuma delimitação especial. Esta instrução é útil quando deseja se apresentar o resultado com alguma informação ou mensagem especial na mesma linha de texto.

A instrução *PRINT* é definida da seguinte forma:

```
\ PRINT { < Parâmetros > } ;
```

Na Tab. 7.2 é apresentada a estrutura dos parâmetros da instrução.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ Print { "Texto" + B+ "Texto2" } ;	"Texto"	Cadeia de texto - *String*
	<Mais>	Caractere de concatenação.
	B	Variável.
	<Mais>	Caractere de concatenação.
	"Texto2"	Cadeia de texto - *String*

Tab. 7.2: Estrutura dos parâmetros da instrução PRINT.

No Exemplo 7.8 são definidas duas variáveis, as quais junto a algumas cadeias de texto serão impressas no EDIVOX.

```
\ BEGIN;                                  %% Inicio do programa
  A = 2;                                  %% Variável A
  B = 3,4;                                %% Variável B
  \ Print { "Variavel"+"A="+A };          %% Instrução de impressão
  \ Print { "Variavel"+"B="+B };          %% Instrução de impressão
\ END;                                    %% Final do programa

VARIAVEL A = 2                            %% Resultado do programa
VARIAVEL B = 3,4                          %% Resultado do programa
```

Exemplo 7.8: Uso da instrução PRINT.

Ao igual que na instrução *PRINT*, se o usuário tenta imprimir uma variável que não tenha sido definida previamente, o programa gerará um erro, o Exemplo 7.9 apresenta esta situação.

```
\ BEGIN;                                  %% Inicio do programa
  B = 3,4;                                %% Variável B
  \ Print { "Variavel"+"A="+A };          %% Instrução de impressão
  \ Print { "Variavel"+"B="+B };          %% Instrução de impressão
\ END;                                    %% Final do programa

#E77 Erro: Comando PRINT, não válido, variável não encontrada.
Parâmetro errôneo: A
Linha: 3
```

Exemplo 7.9: Erro na instrução PRINT, variável não definida.

Outro erro muito freqüente nesta instrução ocorre quando as aspas nos parâmetros da instrução, não são definidas corretamente, tal e como é apresentado no Exemplo 7.10.

```
\ BEGIN;                                  %% Inicio do programa
  \ Print { "Varia"vel="+A };             %% Aspas não válidas
```

```
  \ Print { "Variavel="+A" };                    %% Aspas não válidas
  \ Print { "Variavel=""+A };                    %% Aspas não válidas
\ END;                                           %% Final do programa

#E09 Erro: Variável não válida, caractere não permitido.
Parâmetro errôneo: =
Linha: 2
```

Exemplo 7.10: Erro na instrução PRINT, aspas mal definidas.

7.4.4. Instrução StepResult.

A instrução *StepResult* é muito útil quando deseja se apresentar um procedimento detalhado na avaliação de uma expressão, por o tanto, quando tem se uma expressão aninhada com varias operações ou sub expressões, esta instrução permite imprimir o valor da avaliação de cada uma das sub expressões.

Para utilizar está instrução, primeiro tem que se definir uma expressão com mais de uma operação, e depois nas linhas seguintes inserir a instrução *StepResult*, vale a pena aclarar que esta instrução atua sobre a ultima expressão inserida antes desta. Os resultados gerados por esta instrução serão apresentados da seguinte forma:

< Sub Expressão > < # da sub expressão > < Sub expressão > < Valor da sub expressão >

A instrução *StepResult* é definida da seguinte forma:

\ **StepResult** { < Parâmetros > } ;

Na Tab. 7.2 é apresentada a estrutura dos parâmetros da instrução, o parâmetro da instrução indica o número da sub expressão que será impressa.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ StepResult { 2 } ;	2	Constante
\ StepResult { B } ;	B	Variavel
\ StepResult { } ;	< Vazio >	Quando não é definido nenhum parâmetro, são impressas todas as sub expressões.

Tab. 7.3: Estrutura dos parâmetros da instrução StepResult.

No Exemplo 7.11 é definida uma expressão com varias sub expressões e mediante a instrução StepResult é impresso o resultado de cada uma destas.

```
\ BEGIN;                                    %% Inicio do programa
  A = 23,3;                                 %% Variável A
  A = ( (1 + 4,2)*5 - (3 - A)/4 );                  %% Expressão
  \ StepResult { 1 };                       %% Instrução de impressão
  \ StepResult { 2 };                       %% Instrução de impressão
  \ StepResult { 3 };                       %% Instrução de impressão
  \ StepResult { 4 };                       %% Instrução de impressão
  \ StepResult { 5 };                       %% Instrução de impressão
\ END;                                      %% Final do programa

Sub Expressão 1: 1+4,2 = 5,2                %% Resultado do programa
Sub Expressão 2: 3-A = -20,3
Sub Expressão 3: (1+4,2)*5 = 26
Sub Expressão 4: (3-A)/4 = -5,075
Sub Expressão 5: ((1+4,2)*5)-((3-A)/4) = 31,075
```

Exemplo 7.11: Uso da instrução StepResult.

No anterior exemplo, cada instrução *StepResult* imprimiu o resultado de cada sub expressão da expressão, sem embargo, quando é necessário imprimir o resultado de todas as sub expressões, não é necessário definir uma instrução para cada operação, só tem que se deixar o parâmetro da instrução *StepResult* vazio, como é o caso do *Exemplo 7.12*.

```
\ BEGIN;                                          %% Inicio do programa
  A = 23,3;                                       %% Variável A
  A = ( (1 + 4,2)*5 - (3 - A)/4 );                %% Expressão
  \ StepResult { };                               %% Instrução de impressão
\ END;                                            %% Final do programa

Sub Expressão 1: 1+4,2 = 5,2                      %% Resultado do programa
Sub Expressão 2: 3-A = -20,3
Sub Expressão 3: (1+4,2)*5 = 26
Sub Expressão 4: (3-A)/4 = -5,075
Sub Expressão 5: ((1+4,2)*5)-((3-A)/4) = 31,075
```

Exemplo 7.12: Uso da instrução StepResult, impressão de todos os resultados

Retomando o exemplo anterior, pode se observar que a expressão só tem cinco sub expressões, se o usuário tentar imprimir uma sexta ou outra sub expressão maior, o programa imprimira a ultima operação em qualquer dos casos, o *Exemplo 7.13* apresenta esta situação.

```
\ BEGIN;                                          %% Inicio do programa
  A = 23,3;                                       %% Variável A
  A = ( (1 + 4,2)*5 - (3 - A)/4 );                %% Expressão
  \ StepResult { 6 };                             %% Instrução de impressão
  \ StepResult { 10 };                            %% Instrução de impressão
  \ StepResult { 15 };                            %% Instrução de impressão
\ END;                                            %% Final do programa

Sub Expressão 5: ((1+4,2)*5)-((3-A)/4) = 31,075   %% Resultado do programa
Sub Expressão 5: ((1+4,2)*5)-((3-A)/4) = 31,075
Sub Expressão 5: ((1+4,2)*5)-((3-A)/4) = 31,075
```

Exemplo 7.13: Uso da instrução StepResult, impressão da ultima expressão.

7.4.5. Instrução N – Salto de linha

Esta instrução só é utilizada quando necessita se inserir um salto de linha nos resultados do algoritmo, ela é utilizada da seguinte maneira (Exemplo 7.14).

```
\ BEGIN;                                  %% Inicio do programa
  A = ( (1 + 4,2)*5 - (3 – 2.2)/4 );      %% Expressão
  \ Print { "O"+"resultado"+"foi" };      %% Instrução de impressão
  \ N;                                    %% Salto de linha
  \ Print { A };                          %% Instrução de impressão
\ END;                                    %% Final do programa

O RESULTADO FOI                           %% Resultado do programa
                                          %% Salto de linha inserido
25,8                                      %% Resultado do programa
```

Exemplo 7.14: Uso da instrução N – Salto de linha.

7.5. Blocos de instruções

Um bloco de instruções define um conjunto de instruções que são executadas num mesmo processo. Existem diversas instruções tais como: a instrução *IF*, *WHILE* ou *FOR*, as quais para sua execução precisam ter definido um bloco de instruções. Um bloco de instruções inicia com qualquer destas instruções e finaliza com a instrução *CONTINUE*.

A instrução *CONTINUE* tem que ser implementada da seguinte maneira:

```
<Instrução IF, WHILE ou FOR >
    < Bloco  de instruções >
    <Instrução CONTINUE >
```

Nas seções seguintes serão estudadas as instruções *IF*, *WHILE* e *FOR*, onde vai ser apresentado mais detalhadamente o processo de criação e definição de blocos de instruções.

7.6. Estruturas de controle de fluxo.

No MATVOX existem duas instruções para controlar o fluxo do programa, a primeira delas é a instrução *GOTO* a qual depende de uma etiqueta definida dentro do programa, e a segunda é a instrução *IF*, a qual depende de uma condição definida nos parâmetros desta.

7.6.1. Etiquetas.

As etiquetas são pontos de entrada ou de referencia para a instrução de controle de fluxo *GOTO*, elas podem ser definidas no inicio de qualquer linha do programa mediante o caractere dois pontos. Estas etiquetas podem ser criadas com muita liberdade, mas têm algumas restrições similares as restrições nas variáveis, os caracteres permitidos nestas são os seguintes:

- Caracteres de A – Z.
- Caracteres de 0 – 9.
- Caracteres: Abre colchete – [, Fecha colchete –], Sublinhado – _

As etiquetas têm a seguinte estrutura:

< Nome da etiqueta > Caractere Dois pontos

À igual que nas variáveis, o nome das etiquetas sempre tem que começar com uma letra ou uma vocal. No *Exemplo 7.15* e *Exemplo 7.16* são apresentadas algumas definições válidas e errôneas de etiquetas num programa.

```
\ BEGIN;                    %% Inicio do programa
  E0:                       %% Etiqueta 1 "E0"
  A = 2;                    %% Expressão
  E1: A = 5;                %% Etiqueta 2 "E1"
  E2: B = 3;                %% Etiqueta 3 "E2"
SAIDA: \ END;               %% Etiqueta 3 "SAIDA"
```

Exemplo 7.15: Etiquetas definidas corretamente.

```
\ BEGIN;                    %% Inicio do programa
  1E:                       %% O primeiro caractere da etiqueta é numérico
  A = 2;                    %% Expressão
  E%: A = 5;                %% A etiqueta tem um caractere não permitido
  B = 3; E2:                %% A etiqueta não esta no inicio da linha
\ END;                      %% Fim do programa

#E38 Erro: Etiqueta não válida, caractere numérico não permitido no inicio da etiqueta.
Parâmetro errôneo: 1
Linha: 2
```

Exemplo 7.16: Etiquetas definidas incorretamente.

7.6.2. Instrução GOTO.

Esta instrução de controle permite transferir o fluxo do programa diretamente ao ponto etiquetado com o parâmetro definido na instrução. A utilização da instrução *GOTO* não é muito recomendável já que ela não permite criar um código estruturado, o qual dificulta em grande medida o analises e o entendimento do código desenvolvido. Alem disso existem outros mecanismos e instruções que permitem evitar a utilização desta instrução, as quais serão estudadas mais adiante.

A instrução *GOTO* é definida da seguinte forma:

```
\ GOTO { < Etiqueta > } ;
```

Na Tab. 7.4 é apresentada a estrutura dos parâmetros da instrução, os quais indicam o nome da etiqueta à qual será transferido o fluxo do programa.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ GOTO { UM } ;	UM	Nome da etiqueta

Tab. 7.4: Estrutura dos parâmetros da instrução GOTO.

O *Exemplo 7.17* apresenta a forma correta de implementar a instrução *GOTO* num algoritmo, nele a linha 5 A = A + 1 não é executada, já que quando o programa executa a instrução *GOTO*, o fluxo do programa é desviado para a etiqueta DOIS, a qual está definida no final do programa.

```
\ BEGIN;                    %% Inicio do programa
  A = 5;                    %% Expressão
  B = 3;                    %% Expressão
  \ Goto { DOIS };          %% Instrução GOTO
  A = A+1;                  %% Expressão
DOIS: \ END;                %% Define etiqueta "DOIS"
```

Exemplo 7.17: Uso da instrução GOTO.

7.6.3. Instrução IF e ELSE.

Uma das instruções mais importantes que podem ser especificadas num programa, são aquelas que permitem ao programa executar determinadas tarefas dependendo de certas condições, ou seja, executar parte do código condicionalmente. Para isso só é necessário especificar as condições que permitiram a execução ou não das instruções definidas no bloco de instruções seguinte.

Para realizar o anterior, o MATVOX utiliza a instrução *IF*, na qual é definida uma condição e um bloco de instruções que será executado, dependendo da veracidade ou não, da condição definida previamente.

A instrução *IF* é definida da seguinte forma:

```
\ IF { < Parâmetros > } ;
```

Na Tab. 7.2 é apresentada a estrutura dos parâmetros da instrução, os quais indicam a condição que será avaliada para a execução ou não do bloco de instruções.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ IF { A + 5 >= B - 2} ;	A + 5	Expressão de comparação 1
	>=	Operador relacional
	B - 2	Expressão de comparação 2

Tab. 7.5: Estrutura dos parâmetros da instrução IF.

Os parâmetros de comparação são simplesmente expressões, as quais mediante um operador relacional são avaliadas e comparadas, obtendo como resultado um valor *True* ou *False*, o qual habilita a execução ou não do bloco de instruções seguinte.

Junto à instrução *IF* é possível utilizar a instrução *ELSE* a qual, quando a condição é *False*, executa um bloco de instruções diferente ao principal.

No *Exemplo 7.18* é apresentado o caso no qual, a condição da instrução IF é *True* pelo qual o bloco de instruções seguinte será executado.

```
\ BEGIN;                                          %% Inicio do programa
  A = 45;                                         %% Expressão
  B = 30;                                         %% Expressão
  \ IF { A + 5 > 15 + B } ;                       %% Inicio da instrução IF
    \ Print { "a"+"condição"+"é"+"verdadeira" } ; %% Bloco de instruções.
  \ CONTINUE;                                     %% Fim do bloco da instrução IF
\ END;                                            %% Fim do programa

A CONDIÇÃO É VERDADEIRA                           %% Resultado do programa
```

Exemplo 7.18: Uso da instrução IF.

No *Exemplo 7.19* é avaliada a mesma condição do *Exemplo 7.18*, sem embargo neste caso a condição é *False*, então é definida a instrução *ELSE*, a qual executará o bloco de instruções secundário.

```
\ BEGIN;                                          %% Inicio do programa
  A = 15;                                         %% Expressão
  B = 30;                                         %% Expressão
  \ IF { A + 5 > 15 + B } ;                              %% Inicio da instrução IF
    \ Print { "a"+"condição"+"é"+"verdadeira" } ;  %% Bloco instruções principal.
  \ ELSE ;
    \ Print { "a"+"condição"+"é"+"falsa" } ;       %% Bloco instruções secundário.
  \ CONTINUE;                                             %% Fim do bloco da instrução IF
\ END;                                            %% Fim do programa

A CONDIÇÃO É FALSA                                %% Resultado do programa
```

Exemplo 7.19: Uso da instrução ELSE.

Por outra parte quando necessitam se avaliar diversas condições, as instruções *IF* podem ser concatenadas. No *Exemplo 7.20* é avaliada uma expressão A, e posteriormente o programa determina a faixa de números [0 – 30], [30 – 50], [50 – 100] à qual pertence o resultado da expressão.

```
\ BEGIN;                                          %% Inicio do programa
  A = 2 * 110 / 5,3;                              %% Expressão
  STR B = Faixa_nao_valida;                       %% Expressão

  \ IF { A  > 0 };                                %% Inicio da instrução IF
    STR B = Faixa [0 – 30];                       %% Bloco de instruções.
  \ CONTINUE;                                     %% Fim do bloco da instrução IF

  \ IF { A  > 30 };                               %% Inicio da instrução IF
    STR B = Faixa [30 – 50];                      %% Bloco de instruções.
  \ CONTINUE;                                     %% Fim do bloco da instrução IF

  \ IF { A  > 50 };                               %% Inicio da instrução IF
     STR B = Faixa [50 - 100];                    %% Bloco de instruções.
  \ CONTINUE;                                     %% Fim do bloco da instrução IF

  \ IF { A  > 100 };                              %% Inicio da instrução IF
    STR B = Faixa_nao_valida;                     %% Bloco de instruções.
```

```
  \ CONTINUE;                          %% Fim do bloco da instrução IF

  \ VarResult { A };                   %% Instrução de impressão.
  \ Print { STR B };                   %% Instrução de impressão.
\ END;                                 %% Fim do programa

A = 41,5094339622642                   %% Resultado do programa
FAIXA[30-50]
```

Exemplo 7.20: Uso de instruções IF concatenadas.

No *Exemplo 7.20* as condições foram avaliadas separadamente, sem embargo, em muitos casos é necessário avaliar diferentes condições ao mesmo tempo, para este caso as instruções *IF* são utilizadas de forma aninhada.

No Exemplo 7.21 a variável A é avaliada de acordo as seguintes condições:

- É um número negativo par?
- É um número negativo impar?
- É um número positivo par?
- É um número positivo impar?
- É igual a zero?

```
\ BEGIN;                               %% Inicio do programa
  N = - 4.3 ;
  A = Round(N * 30.5 – 100);           %% Expressão
  STR B = Zero;                        %% Expressão

  \ IF { A  > 0 };                     %% Inicio da instrução IF
    \ IF { Frac (A/2)  <>  0 };
      STR B = Positivo_Impar;
   \ ELSE;
      STR B = Positivo_Par;
    \ CONTINUE;
  \ CONTINUE;                          %% Fim do bloco da instrução IF

  \ IF { A  < 0 };                     %% Inicio da instrução IF
```

```
   \ IF { Frac (A/2)  <>  0 };
      STR B = Negativo_Impar;
   \ ELSE;
      STR B = Negativo _Par;
    \ CONTINUE;
  \ CONTINUE;                          %% Fim do bloco da instrução IF

  \ VarResult { A };                   %% Instrução de impressão.
  \ Print { STR B };                   %% Instrução de impressão.
\ END;                                 %% Fim do programa

A = -231                               %% Resultado do programa
NEGATIVO_IMPAR
```

Exemplo 7.21: Uso de instruções IF aninhadas.

7.6.4. Erros comuns no uso da instrução IF.

Nesta secção serão apresentados alguns exemplos com casos de erro nos quais estão envolvidas as instruções *IF*, tais como: parâmetros mal definidos, blocos não encerrados, entre outros.

No Exemplo 7.22, o operador de comparação da instrução *IF* não foi definido pelo qual é gerado um erro indicando a ausência deste.

```
\ BEGIN;                               %% Inicio do programa
  A = 5;
  \ IF { A   4 };                      %% Inicio da instrução IF
    A = 0;
  \ CONTINUE;                          %% Fim do bloco da instrução IF
\END;                                  %% Fim do programa

#E21 Erro: Comando IF, não válido, operador de comparação, não encontrado.
Linha: 3
```

Exemplo 7.22: Erro na instrução IF, operador não definido.

No *Exemplo 7.23*, são utilizadas duas instruções *IF* aninhadas, as quais têm seus parâmetros todos corretos, sem embargo, um dos seus blocos não foi fechado corretamente, pelo qual o programa gera um erro indicando a ausência de uma instrução *CONTINUE*, este erro é muito freqüente quando são utilizadas varias instruções deste tipo, pelo qual é importante ter sempre em conta, que por cada instrução *IF* utilizada, é necessária uma instrução *CONTINUE*.

```
\ BEGIN;                          %% Inicio do programa
  A = 2;
  \ IF { A  > 5 };                %% Inicio da instrução IF - 1
    A = 5;
  \ IF { A > 3 };                 %% Inicio da instrução IF - 2
    A = 4;
  \ CONTINUE;                     %% Fim do bloco da instrução IF - 2
\END;                             %% Fim do programa

#E49 Erro: Bloco de instruções não válido, comando CONTINUE, não encontrado.
Linha: 8
```

Exemplo 7.23: Erro na instrução IF, bloco não encerrado.

Também pode existir o caso no qual a instrução *CONTINUE* está numa posição incorreta, este caso é mostrado no *Exemplo 7.24*.

```
\ BEGIN;                          %% Inicio do programa
  A = 5;
  \ CONTINUE;                     %% Instrução não válida.
  \ IF { A >= 4 };                %% Inicio da instrução IF
    A = 0;
  \ CONTINUE;                     %% Fim do bloco da instrução IF
\END;                             %% Fim do programa

#E48 Erro: Comando CONTINUAR, não válido. o comando já foi definido.
Linha: 3
```

Exemplo 7.24: Erro na instrução IF, instrução CONTINUE não válida.

7.6.5. Uso da instrução GOTO como instrução iterativa.

A instrução *GOTO* também pode ser utilizada como uma instrução iterativa, ou seja, uma instrução que permite executar um bloco de instruções repetidas vezes, sem embargo quando utiliza se desta forma é importante e indispensável definir uma condição de saída que permita finalizar o ciclo de execução, já que de modo contrario o programa ficará se executando indefinidamente.

No Exemplo 7.25 mediante o uso de instruções *GOTO* e instruções *IF*, é calculado o fatorial do numero 10.

```
\ BEGIN;                              %% Inicio do programa
  A = 0;                              %% Variável A
  N = 10;
  Fatorial = 1;                       %% Variável Fatorial

Inicio:  A = A + 1;                   %% Expressão
         Fatorial = Fatorial * A;     %% Expressão
         \IF { A = N };               %% Condição de saída
           \GOTO { Saida };           %% Instrução GOTO
         \ CONTINUE;                  %% Instrução CONTINUE, fim da instrução IF
         \ GOTO { Inicio };           %% Instrução GOTO


Saida:  \ VARRESULT { fatorial };     %% Instrução de impressão
\ END;                                %% Fim do programa

FATORIAL = 3628800                    %% Resultado do programa.
```

Exemplo 7.25: Instrução GOTO como instrução iterativa. Calculo de fatorial.

No exemplo anterior foram utilizadas duas variáveis, a variável A, a qual é utilizada como condição de saída, e a variável Fatorial, na qual é armazenado o resultado. Quando o programa executa a instrução *GOTO* com o parâmetro Inicio, o fluxo do programa retorna à linha com etiqueta Inicio, e todo o processo é executado novamente. Quando a condição de saída

é atingida, ou seja, quando a variável A é igual à variável N então o bloco da instrução *IF* é executado e o programa pula para a etiqueta saída.

Por outra parte, se desejar se calcular um fatorial diferente, só é necessário mudar a variável N, mudando desta maneira a condição de saída, ou seja, a condição da instrução *IF*. Sem embargo este programa pode se configurar com variáveis de entrada dinâmicas, o que vai permitir ao usuário inserir o fatorial que vai calcular no começo da execução do algoritmo. Neste caso a variável N fica desta forma: IN N (Exemplo 7.26).

```
\ BEGIN;                          %% Inicio do programa
  A = 0;                          %% Variável A
  IN N;                           %% Variável Dinâmica N
  Fatorial = 1;                   %% Variável Fatorial
Inicio:  A = A + 1;               %% Expressão
         Fatorial = Fatorial * A; %% Expressão
         \IF { A = N };           %% Condição de saída
           \GOTO { Saida };       %% Instrução GOTO
         \ CONTINUE;              %% Instrução CONTINUE, fim da instrução IF
         \ GOTO { Inicio };       %% Instrução GOTO

Saida:  \ VARRESULT { fatorial };  %% Instrução de impressão
\ END;                             %% Fim do programa
```

Exemplo 7.26: Calculo de fatorial com entrada dinâmica.

7.6.6. Erros comuns no uso da instrução GOTO.

Nesta secção são apresentados alguns casos nos quais a instrução *GOTO* gera erros e exceções devido à definição dos seus parâmetros ou simplesmente a sua lógica no programa.

No Exemplo 7.27, é definida uma instrução *GOTO*, sem embargo sua referencia não está definida em nenhuma parte do programa pelo qual é gerado um erro indicando a ausência da etiqueta no programa.

```
\ BEGIN;                          %% Inicio do programa
  A = 1;
  \GOTO { saida };                %% Instrução GOTO
\ END;                            %% Fim do programa

#E42 Erro: Comando GOTO, não válido, etiqueta não encontrada.
Parâmetro errôneo: SAIDA
Linha: 3
```

Exemplo 7.27: Erro na instrução GOTO, etiqueta não encontrada.

No *Exemplo 7.28*, a instrução *GOTO* é utilizada como uma instrução iterativa, sem embargo, não tem definida nenhuma condição de saída, pelo qual o programa fica se executando indefinidamente ate que o usuário aborte o programa.

```
\ BEGIN;                          %% Inicio do programa
  A = 0;
  Inicio: A = A + 1;
  \GOTO { inicio };               %% Instrução GOTO
\ END;                            %% Fim do programa

#E118 Execução abortada pelo usuário.
Linha: 3
```

Exemplo 7.28: Loop infinito na instrução GOTO.

No Exemplo 7.29 embora que, a condição de saída foi definida ela nunca vai ser atingida, pelo qual teoricamente o programa será executado indefinidamente até que o usuário aborte o programa.

```
\ BEGIN;                          %% Inicio do programa
  A = 0;                          %% Expressão
  Inicio: A = A + 1;              %% Expressão
  \IF { A = 0 };                  %% Condição de saída
```

```
   \GOTO { Saida };                    %% Instrução GOTO
  \ CONTINUE;                          %% Instrução Continuar, fim da instrução IF
  \ GOTO { Inicio };                   %% Instrução GOTO
Saida:  \ END;                         %% Fim do programa

#E118 Execução abortada pelo usuário.
Linha: 3
```

Exemplo 7.29: Loop infinito na instrução GOTO.

7.7. Estruturas de repetição.

Uma instrução iterativa é utilizada quando necessita se executar um mesmo código ou bloco de instruções um número determinado de vezes, o qual dependerá do cumprimento de determinadas condições definidas nos parâmetros da instrução.

O MATVOX conta com duas instruções que tem este comportamento, a instrução *WHILE* e a instrução *FOR*.

7.7.1. Instrução WHILE.

A instrução *WHILE* define uma condição e um bloco de instruções, o qual será executado repetidamente, enquanto que, a condição definida previamente seja *True*.

A instrução *WHILE* é definida da seguinte forma:

```
\ WHILE { < Parâmetros > } ;
```

Na Tab. 7.6 é apresentada a estrutura dos parâmetros da instrução, os quais indicam a condição que será avaliada para a execução ou não do bloco de instruções.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ WHILE { A + 5 >= B - 2} ;	A + 5	Expressão de comparação 1
	>=	Operador relacional
	B - 2	Expressão de comparação 2

Tab. 7.6: Estrutura dos parâmetros da instrução WHILE.

Os parâmetros de comparação são simplesmente expressões, as quais mediante um operador relacional são comparadas obtendo como resultado um valor *True* ou *False*. Quando o resultado é *True* o bloco de instruções seguinte é executado e ao finalizar sua execução a condição volta a ser avaliada, este processo é repetido enquanto o resultado da condição não seja *False*, momento no qual o bloco de instruções é ignorado.

Anteriormente foi apresentado um exemplo (Exemplo 7.25) no qual era calculado o fatorial do número 10 utilizando instruções *GOTO*, sem embargo, disse se que seu uso não era o mais adequado nem conveniente, a continuação, no *Exemplo* 7.30 será solucionado o mesmo problema utilizando instruções *WHILE*. Nele a variável N define o fatorial que vai ser calculado e a variável A atua como um contador que vai permitir finalizar a execução da instrução WHILE.

```
\BEGIN;                          %% Inicio do programa.
  A = 0;
Fatorial = 1;
  N = 10;
  \WHILE { A < N };              %% Ciclo WHILE
    A = A + 1;                   %% Contador
    Fatorial = fatorial * A;     %% Calcula Fatorial.
  \ CONTINUE;                    %% Fim do ciclo WHILE.
  \ VarResult { Fatorial };      %% Instrução de impressão.
\END;                            %% Fim do programa.

FATORIAL = 3628800               %% Resultado do programa.
```

Exemplo 7.30: Uso da instrução WHILE, calculo de fatorial.

No Exemplo 7.30 foi utilizada a instrução *VarResult* para imprimir o valor da variável Fatorial no final do ciclo, sem embargo se deseja se apresentar o valor desta variável em cada uma das interações do ciclo, só é necessário inserir a instrução de impressão dentro do bloco de instruções da instrução *WHILE*, como é apresentado no *Exemplo 7.31*.

```
\BEGIN;                                          %% Inicio do programa.
  A = 0;
  N = 10;
  Fatorial = 1;
  \WHILE { A < N };                              %% Ciclo WHILE
    A = A + 1;                                   %% Contador.
    Fatorial = fatorial * A;                     %% Calcula Fatorial.
    \ PRINT {"Fatorial"+"iteração"+A+"="+fatorial};  %% Instrução de impressão.
  \ CONTINUE;                                          %% Fim do ciclo WHILE.
\END;                                            %% Fim do programa.

FATORIAL ITERAÇÃO 1 = 1                          %% Resultado do programa
FATORIAL ITERAÇÃO 2 = 2
FATORIAL ITERAÇÃO 3 = 6
FATORIAL ITERAÇÃO 4 = 24
FATORIAL ITERAÇÃO 5 = 120
FATORIAL ITERAÇÃO 6 = 720
FATORIAL ITERAÇÃO 7 = 5040
FATORIAL ITERAÇÃO 8 = 40320
FATORIAL ITERAÇÃO 9 = 362880
FATORIAL ITERAÇÃO 10 = 3628800
```

Exemplo 7.31: Uso da instrução WHILE, calculo de fatorial passo a passo.

Nos exemplos anteriores pode se observar que o uso de instruções iterativas como a instrução *WHILE* permite fazer algoritmos mais claros e estruturados que usando instruções *GOTO*, sem embargo está instrução também tem algumas considerações especiais que tem que ser tidas em conta para evitar problemas na execução do programa, as quais serão apresentadas na seguinte seção.

7.7.2. Erros comuns no uso da instrução WHILE.

Ao igual que com a instrução *GOTO*, quando utilizam se instruções *WHILE* podem ser gerados ciclos ou loops infinitos, no quais o programa fica se executando indefinidamente. Neste caso, isto ocorre quando a lógica do bloco de instruções não permite que em determinado momento a condição da instrução WHILE seja atingida.

No *Exemplo 7.32* é apresentado o caso no qual, o bloco de instruções do ciclo *WHILE* não tem definida nenhuma expressão que permita mudar o valor da sua condição, pelo qual está condição sempre vai ser verdadeira e o programa vai se ficar executando até que o usuário aborte sua execução.

```
\BEGIN;                                  %% Inicio do programa.
  A = 0;
  Fatorial = 1;
  \WHILE { A < 10 };                     %% Ciclo WHILE.
                                         %% Ausência do contador.
    Fatorial = fatorial * A;
  \ CONTINUE;                            %% Fim ciclo WHILE
  \ VarResult { Fatorial };

%% A variável A sempre vai ser igual a zero pelo qual a condição da instrução WHILE sempre
%% vai ser verdadeira

\END;                                    %% Fim do programa.

#E118 Execução abortada pelo usuário.
Linha: 5
```

Exemplo 7.32: Loop infinito na instrução WHILE, contador não definido.

Outro caso no qual pode ser gerada está exceção, ocorre quando os valores das expressões da condição mudam, mas sua variação não permite que em determinado momento a condição seja *False*, pelo qual, neste caso o bloco de instruções também será executado indefinidamente. O *Exemplo 7.33* apresenta este caso.

```
\BEGIN;                                  %% Inicio do programa.
A = 0;
Fatorial = 1;
\WHILE { A < 10 };                       %% Ciclo WHILE.
A = A - 1;                               %% Contador.
\ CONTINUE;
\ VARRESULT { Fatorial };

%% A variável A nunca vai a convergir a 10 pelo qual a condição da instrução WHILE sempre
%% vai ser verdadeira.

\END;                                    %% Fim do programa.

#E118 Execução abortada pelo usuário.
Linha: 5
```

Exemplo 7.33: Loop infinito na instrução WHILE, condição divergente.

7.7.3. Instrução FOR.

A instrução *FOR* é uma instrução iterativa a qual utiliza um contador com incremento de um, para executar um bloco de instruções repetidamente. A instrução *FOR* tem um comportamento muito similar à instrução *WHILE*, com a diferença que esta, não depende de nenhuma condição, já que seu comportamento só depende dos parâmetros definidos inicialmente, os quais são estáticos e não mudam com a execução do bloco de instruções.

A instrução *FOR* é definida da seguinte forma:

```
\ FOR { < Parâmetros > };
```

Na Tab. 7.7 é apresentada a estrutura dos parâmetros da instrução, os quais definem o número de vezes que vai ser executado o bloco de instruções.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ FOR { I = B – 2 : 100 } ;	I	Variável do contador
	B -2	Inicio do contador
	100	Final do contador

Tab. 7.7: Estrutura dos parâmetros da instrução FOR.

O inicio e o final do contador são expressões, as quais definem a faixa de valores, na qual a variável do contador vai variar. Esta variação faz que em cada iteração e execução do bloco de instruções, a variável do contador incremente seu valor em um.

Quando os parâmetros do contador são definidos, o valor inicial da variável definida no FOR é sempre o valor do inicio do contador, no Exemplo 7.34, inicialmente a variável A tem o valor de 3, mas no contador ela muda seu valor a zero. Por outra parte, esta instrução trabalha com valores inteiros, pelo qual quando são definidos valores com ponto flutuante, a parte decimal do número é ignorada.

Também pode se observar que os valores que definem o contador são estáticos, neste caso o rango dele depende da variável A, pelo qual ele ficou definido de 0 até 7, sem embargo, ainda que a variável A muda seu valor no bloco de instruções A = 1, o rango do contador não muda e os valores inicialmente definidos são mantidos.

```
\BEGIN;                                  %% Inicio do programa.
  A = 3;

\FOR { A = 0 : A + 1,9 };                %% Inicio ciclo FOR.
    \Print{"ITERAÇÃO"+A};
    A = 1;
    \Print{"VARIÁVEL"+"A="+A};
    \N;
  \CONTINUE;                             %% Fim ciclo FOR.

\END;                                    %% Fim do programa.

ITERAÇÃO 0                               %% Resultado do programa.
```

```
VARIÁVEL A= 1

ITERAÇÃO 1
VARIÁVEL A= 1

ITERAÇÃO 2
VARIÁVEL A= 1

ITERAÇÃO 3
VARIÁVEL A= 1

ITERAÇÃO 4
VARIÁVEL A= 1
```

Exemplo 7.34: Comportamento dos parâmetros da Instrução FOR.

O comportamento da instrução *FOR* pode se comparar desde certo ponto de vista com o funcionamento da instrução *WHILE*, com a diferença que a primeira não precisa definir nenhuma condição de saída. O *Exemplo 7.35* apresenta a equivalência funcional das instruções *FOR* e *WHILE*.

```
\BEGIN;                                  %% Inicio do programa.

 \ Print{"Instrução"+"FOR"};
  B = 3;
  \FOR { A = 2 : B + 4 };                %% Inicio ciclo FOR.
    \ VarResult { A };
  \CONTINUE;                             %% Fim ciclo FOR.
  \N;
  \ Print{"Instrução"+"WHILE"};
  A = 2;
  B = 3;
  \WHILE { A <= B + 4 };                 %% Inicio ciclo WHILE.
    \ VarResult { A };
    A = A + 1;                           %% Contador.
  \CONTINUE;                             %% Fim ciclo WHILE.

\END;                                    %% Fim do programa.
```

```
INSTRUÇÃO FOR                          %% Resultados ciclo FOR.
A = 2
A = 3
A = 4
A = 5
A = 6
A = 7

INSTRUÇÃO WHILE                        %% Resultados ciclo WHILE.
A = 2
A = 3
A = 4
A = 5
A = 6
A = 7
```

Exemplo 7.35: Comparação funcional entre as instruções FOR e WHILE.

No Exemplo 7.30 foi criado um algoritmo que calculava o fatorial do número 10 mediante o uso da instrução *WHILE*, este mesmo exemplo pode ser feito também, utilizando a instrução *FOR*, a qual neste caso simplifica bastante o algoritmo. No Exemplo 7.36 é apresentado este caso.

```
\BEGIN;                                          %% Inicio do programa.
  FATORIAL=1;
  \FOR {A=1:10};                                 %% Inicio ciclo FOR.
    FATORIAL=FATORIAL*A;                         %% Calcula fatorial.
    \Print {"Fatorial"+"iteração"+A+"="+FATORIAL};
  \CONTINUE;                                     %% Fim ciclo FOR.
\END;                                            %% Fim do programa.

FATORIAL ITERAÇÃO 1 = 1                          %% Resultado do programa.
FATORIAL ITERAÇÃO 2 = 2
FATORIAL ITERAÇÃO 3 = 6
FATORIAL ITERAÇÃO 4 = 24
FATORIAL ITERAÇÃO 5 = 120
```

```
FATORIAL ITERAÇÃO 6 = 720
FATORIAL ITERAÇÃO 7 = 5040
FATORIAL ITERAÇÃO 8 = 40320
FATORIAL ITERAÇÃO 9 = 362880
FATORIAL ITERAÇÃO 10 = 3628800
```

Exemplo 7.36: Uso da Instrução FOR, calculo de fatorial.

7.7.4. Erros comuns no uso da instrução FOR.

Nas secções anteriores fala se que em alguns casos, quando as instruções iterativas estavam mal definidas, o algoritmo podia ficar se executando indefinidamente, sem embargo, a instrução *FOR* não tem esse problema, graças a que o rango do seu contador é estático, ou seja, seus valores não são mudados durante a execução do ciclo.

Por outra parte, alguns erros nesta instrução podem ser gerados, por exemplo, quando o valor inicial do contador é maior ao valor final do mesmo, este caso é apresentado no Exemplo 7.37.

```
\BEGIN;                                  %% Inicio do programa.
  \FOR { A = 10 : 5 };                   %% Inicio ciclo FOR.
    \VarResult { A };
  \CONTINUE;                             %% Fim ciclo FOR.
\END;                                    %% Fim do programa.
#E58 Erro: Comando FOR, não válido, o inicio do contador, tem que ser maior ao final do
     contador.
Parâmetro errôneo: 10>=5
Linha: 2
```

Exemplo 7.37: Erro instrução FOR, parâmetros mal definidos.

7.8. Instruções de persistência.

A persistência é a ação de preservar informação de forma permanente no sistema, e a sua vez refere se à capacidade de poder recuperar a informação para que esta possa ser usada novamente.

O MATVOX implementa algumas instruções que permitem ao usuário armazenar ou remover dados permanentemente no sistema. Estes dados podem ser variáveis, constantes ou expressões, as quais uma vez armazenadas podem ser lidas e utilizadas por futuros programas.

7.8.1. Instrução SaveConstant.

A instrução *SaveConstant* permite armazenar permanentemente no sistema, constantes definidas previamente pelo usuário. Estas constantes podem ser utilizadas mais adiante por outros programas, mas estas não podem ser modificadas por estes.

A instrução *SaveConstant* é definida da seguinte forma:

\ **SaveConstant** { < Parâmetros > } ;

Na Tab. 7.8 é apresentada a estrutura dos parâmetros da instrução, os quais indicam os nomes das constantes que serão armazenadas no sistema.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ SaveConstant { A , B, C } ;	A	Variável armazenada
	<Vírgula>	Caractere de separação.
	B	Variável armazenada
	<Vírgula>	Caractere de separação.
	C	Variável armazenada

Tab. 7.8: Estrutura dos parâmetros da instrução SaveConstant.

No Exemplo 7.38, a constante ConstA e ConstB são armazenadas permanente no sistema mediante a instrução *SaveConstant*, isto quer dizer, que ainda que o programa seja fechado, estas constantes vão ficar gravadas e poderão ser utilizadas mais adiante.

```
\ BEGIN;                              %% Inicio do programa.
  ConstA = 2.343;                     %% Constante.
  ConstB = 5E-2 + 4E-3;               %% Constante.

  \ SaveConstant { ConstA, ConstB };  %% Instrução de armazenamento.
\ END;                                %% Fim do programa.
```

Exemplo 7.38: Uso da instrução SaveConstant, armazenamento de constantes.

Uma vez que a constante é armazenada, esta pode ser utilizada por qualquer programa como é apresentado no Exemplo 7.39, onde as constantes ConstA e ConstB são lidas para avaliar uma expressão.

```
\ BEGIN;                              %% Inicio do programa.

  B = (ConstA + 1)*ConstB;            %% Expressão.
  \ VarResult { B };                  %% Instrução de impressão.

\ END;                                %% Fim do programa.

B = 0.180522                          %% Resultado do programa.
```

Exemplo 7.39: Leitura e uso de constantes de usuário.

7.8.2. Erros comuns no uso da instrução SaveConstant

No MATVOX, quando um programa armazena uma constante no sistema, esta automaticamente ficará reservada por este, isto quer dizer que enquanto uma constante esteja armazenada, ela não poderá ser modificada por nenhum programa, ou seja, esta só poderá ser

lida. No Exemplo 7.40, a constante ConstA (a qual no Exemplo 7.38 foi armazenada no sistema) vai tentar modificar seu valor, o que causa um erro na execução do programa.

```
\ BEGIN;                    %% Inicio do programa.
  ConstA = 5;               %% Constante.
\ END;                      %% Fim do programa.

#E53 Erro: Variável não válida, palavra ou constante reservada pelo sistema.
Parâmetro errôneo: CONSTA
Linha: 2
```

Exemplo 7.40: Erro, constante reservada pelo sistema.

7.8.3. Instrução DeleteConstant.

A instrução *DeleteConstant* é utilizada quando deseja se apagar uma ou mais constantes de usuário do sistema e é definida da seguinte forma:

```
\ DeleteConstant { < Parâmetros > } ;
```

Na Tab. 7.9 é apresentada a estrutura dos parâmetros da instrução, os quais indicam as constantes que serão apagadas do sistema.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ DeleteConstant { A , B, C } ;	A	Constante apagada
	<Vírgula>	Caractere de separação.
	B	Constante removida
	<Vírgula>	Caractere de separação.
	C	Constante removida
\ DeleteConstant { } ;	<Vazio>	Todas as constates são apagadas

Tab. 7.9: Estrutura dos parâmetros da instrução DeleteConstant.

```
\ BEGIN;                                  %% Inicio do programa.
  \ DeleteConstant { ConstA, ConstB };
\ END;                                    %% Fim do programa.
```

Exemplo 7.41: Uso da instrução DeleteConstant.

No Exemplo 7.41, são apagadas as constantes ConstA e ConstB as quais foram armazenadas previamente. Sem embargo, para apagar todas as constantes do sistema, não é pratico ter que escrever cada uma delas, para isso só é necessário deixar os parâmetros da instrução sem definir, como é apresentado no Exemplo 7.42.

```
\ BEGIN;                                  %% Inicio do programa.
\ DeleteConstant { };                     %% Apaga todas as constantes.
\ END;                                    %% Fim do programa.
```

Exemplo 7.42: Uso da instrução DeleteConstant, todas as constantes são apagadas.

7.8.4. Instrução SaveExpression

A instrução *SaveExpression* permite armazenar expressões ou equações no sistema, as quais podem ser utilizadas mais adiante em outros programas. Ela é definida da seguinte forma:

```
\ SaveExpression { < Parâmetros > } ;
```

Na Tab. 7.10 é apresentada a estrutura dos parâmetros da instrução, os quais indicam o nome e a expressão que será armazenada no sistema.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ SaveExpression { A = B + C- 1} ;	A	Nome da expressão
	<Igual>	Operador igual.
	B + C - 1	Expressão

Tab. 7.10: Estrutura dos parâmetros da instrução SaveExpression.

No Exemplo 7.43 é armazenada a equação quadrática

$$(Quad = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}).$$

```
\ BEGIN;                                    %% Inicio do programa.
  \ SaveExpression {Quad=( -b + sqrt ( sqr(b) – 4a*c ))/(2a) };
\ END;                                      %% Fim do programa.
```

Exemplo 7.43: Uso da instrução SaveExpression, armazenamento da equação quadrática.

7.8.5. Instrução DeleteExpression.

A instrução DeleteExpression permite apagar uma ou mais expressões de usuário que estejam armazenadas no sistema. Ela é definida da seguinte forma:

```
\ DeleteExpression { < Parâmetros > } ;
```

Na Tab. 7.11 é apresentada a estrutura dos parâmetros da instrução, os quais indicam o nome das expressões que serão apagadas do sistema.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ DeleteExpression { A , B, C } ;	A	Nome da expressão apagada
	<Vírgula>	Caractere de separação.
	B	Nome da expressão apagada
	<Vírgula>	Caractere de separação.
	C	Nome da expressão apagada
\ DeleteConstant { } ;	<Vazio>	Todas as expressões são apagadas

Tab. 7.11: Estrutura dos parâmetros da instrução DeleteExpression.

No Exemplo 7.44 são apagadas algumas expressões do sistema. À igual que na instrução *DeleteConstant*, se os parâmetros da instrução não são definidos, todas as expressões serão apagadas.

```
\ BEGIN;                                  %% Inicio do programa.
  \ DeleteExpression { Quad };                %% Apaga a expressão QUAD
  \ DeleteExpression { };                 %% Apaga todas as expressões.
\ END;                                    %% Fim do programa.
```

Exemplo 7.44: Uso da instrução DeleteRxpression.

7.8.6. Instrução SaveMemory.

Quando num programa são gerados muitos dados e estes precisam ser utilizados mais adiante, fica complicado armazenar cada um deles mediante instruções *SaveConstant*. Para fazer isto, pode ser utilizada a instrução *SaveMemory*, a qual permite armazenar automaticamente todas ou algumas das variáveis do programa num arquivo tipo CSV.

A instrução *SaveMemory* é definida da seguinte forma:

```
\ SaveMemory { < Parâmetros > } ;
```

Na Tab. 7.12 é apresentada a estrutura dos parâmetros da instrução, os quais indicam o nome do arquivo que será gerado.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ SaveMemory { Nome } ;	Nome	Nome do arquivo CSV no qual vão ser armazenadas as variáveis.
\ SaveMemory { Nome , V_N} ;	Nome	Nome do arquivo CSV no qual vão ser armazenadas as variáveis.
	<Vírgula>	Caractere de separação.
	V_N	Variavel (simples ou *Array* de dados) que vai ser armazenada no

		arquivo CSV.

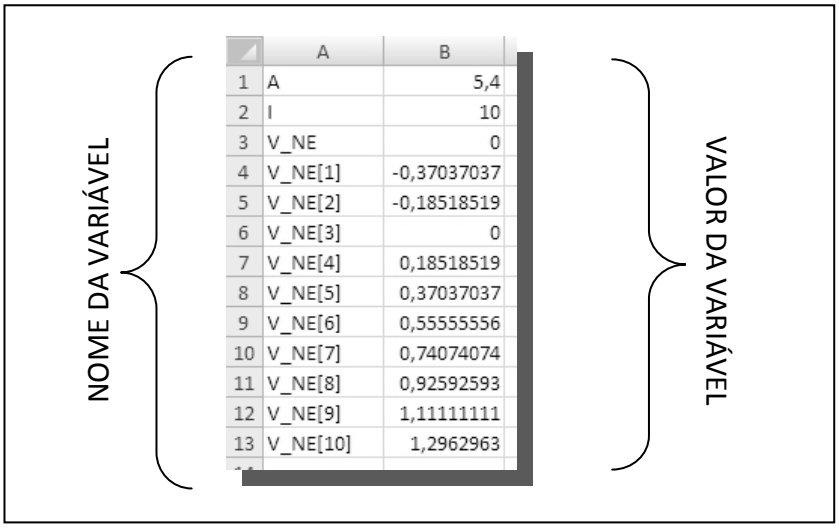
Tab. 7.12: Estrutura dos parâmetros da instrução SaveMemory.

No Exemplo 7.45 é gerado um *Array* com 10 dados, os quais são armazenados num arquivo de nome: *MA*. Neste caso, na instrução *SaveMemory* só foi definido o nome do arquivo CSV, pelo qual, automaticamente são armazenadas todas as variáveis.

```
\ BEGIN;                          %% Inicio do programa.
  A=5.4;
  \ FOR { i = 1 : 10 };
    V_N E [ i ] = (i-3/A);
  \ CONTINUE;
  \SaveMemory { MA };             %% Instrução de armazenamento.
\END;                             %% Fim do programa.
```

Exemplo 7.45: Uso da instrução SaveMemory, armazenagem de todas as variáveis.

Na Fig. 7.1 podem se observar as variáveis do programa no arquivo MA.CSV , os qual foi gerado pela instrução.



	A	B
1	A	5,4
2	I	10
3	V_NE	0
4	V_NE[1]	-0,37037037
5	V_NE[2]	-0,18518519
6	V_NE[3]	0
7	V_NE[4]	0,18518519
8	V_NE[5]	0,37037037
9	V_NE[6]	0,55555556
10	V_NE[7]	0,74074074
11	V_NE[8]	0,92592593
12	V_NE[9]	1,11111111
13	V_NE[10]	1,2962963

Fig. 7.1: Arquivo MA.CSV gerado pela instrução SaveMemory.

Por outra parte quando é necessário armazenar só uma variável ou um *Array* de dados, então tem que se especificar o nome deste, nos parâmetros da instrução. No Exemplo 7.46 são gerados dois *Array* numéricos (V_N E e V_N F) os quais são posteriormente armazenados em arquivos CSV diferentes. (arquivo M1.CSV e M2.CSV).

```
\ BEGIN;                                  %% Inicio do programa.
  A=5.4;
  \ FOR { i = 1 : 10 };
    V_N E [ i ] = (i-3/A);
    V_N F [ i ] = (i-3/A);
  \ CONTINUE;
  \SaveMemory { M1, V_N E };              %% Instrução de armazenamento.
  \SaveMemory { M2, V_N F };              %% Instrução de armazenamento.
\END;                                     %% Fim do programa.
```

Exemplo 7.46: Uso da instrução SaveMemory, armazenagem de variáveis separadas.

7.8.7. ReadMemory

A instrução *ReadMemory* permite carregar no algoritmo, as variáveis que tenham sido armazenadas no arquivo CSV pela instrução *SaveMemory*. Esta instrução é definida da seguinte forma:

```
\ ReadMemory { < Parâmetros > } ;
```

Na Tab. 7.13 é apresentada a estrutura dos parâmetros da instrução, os quais indicam o nome do arquivo que vai ser carregado no programa.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ ReadMemory { Nome } ;	Nome	Nome do arquivo CSV que vai ser carregado.

Tab. 7.13: Estrutura dos parâmetros da instrução SaveMemory.

No Exemplo 7.48 são carregados os dados armazenados no Exemplo 7.46 e posteriormente são impressos na tela do EDIVOX. As variáveis que são carregadas ficam com o mesmo nome que tinham inicialmente, neste caso V_N E e V_N F.

```
\ BEGIN;                                    %% Inicio do programa.
  \ReadMemory { M1 };
  \ReadMemory { M2 };

  \ FOR { i = 1 : 10 };
    \ VarResult { V_N E [i] , V_N F [i] };
  \ CONTINUE;
\END;                                       %% Fim do programa.

V_NE[1] = 0,444444444444445
V_NF[1] = 0,444444444444445
V_NE[2] = 1,44444444444445
V_NF[2] = 1,44444444444445
V_NE[3] = 2,44444444444445
V_NF[3] = 2,44444444444445
V_NE[4] = 3,44444444444444
V_NF[4] = 3,44444444444444
V_NE[5] = 4,44444444444444
V_NF[5] = 4,44444444444444
V_NE[6] = 5,44444444444444
V_NF[6] = 5,44444444444444
V_NE[7] = 6,44444444444444
V_NF[7] = 6,44444444444444
V_NE[8] = 7,44444444444444
V_NF[8] = 7,44444444444444
V_NE[9] = 8,44444444444444
V_NF[9] = 8,44444444444444
V_NE[10] = 9,44444444444444
V_NF[10] = 9,44444444444444
```

Exemplo 7.47: Uso da instrução ReadMemory.

7.9. Importação e Exportação de dados.

O MATVOX tem algumas instruções que permitem manipular dados externos, mediante as quais o usuário pode importar ou exportar dados (geralmente de tipo numérico) desde ou para arquivos com extensão TXT ou CSV.

Os arquivos de texto puro ou texto plano TXT (*plain text)* são arquivos sem nenhum tipo de formatação, tais como negrito, itálico e sublinhado, e estão compostos só por caracteres. Por outra parte os arquivos CSV (*comma separated values)* são um tipo de arquivo que permite armazenar dados tabelados, onde as colunas são separadas pelo caractere *ponto e vírgula* e as filas por *quebras de linha*.

Num arquivo CSV a seguinte tabela (Fig. 7.2) vai ter a seguinte formatação (Fig. 7.3).

A	B	C	D	E
1	2	3	4	5

Fig. 7.2: Exemplo arquivo CSV - Tabela de dados.

```
A ; B ; C ; D ; E
1 ; 2 ; 3 ; 4 ; 5
```

Fig. 7.3: Formatação de dados num arquivo CSV.

7.9.1. Instrução ImportText.

A instrução *ImportText* permite importar dados numéricos ou de tipo *String* desde um arquivo de texto TXT, os quais são armazenados numa variável simples ou num *Array* de dados, dependendo do número de dados importados. Esta instrução é definida da seguinte forma:

\ **ImportText** { < Parâmetros > } ;

Na Tab. 7.14 é apresentada a estrutura dos parâmetros da instrução, os quais indicam a origem e o destino dos dados.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ ImportText { Arquivo , A, B, C, D, Variavel} ;	Arquivo	Nome do arquivo TXT.
	<Vírgula>	Caractere de separação.
	A	Coluna Inicial
	<Vírgula>	Caractere de separação.
	B	Coluna Final
	<Vírgula>	Caractere de separação.
	C	Fila Inicial
	<Vírgula>	Caractere de separação.
	D	Fila final
	<Vírgula>	Caractere de separação.
	Variavel	Nome da variável que vai armazenar os dados importados.

Tab. 7.14: Estrutura dos parâmetros da instrução ImportText.

Arquivo: Corresponde ao nome do arquivo de extensão **TXT**, do qual vão ser importados os dados. Se o arquivo não existe, o programa gerará um erro indicando a ausência deste. Para definir este parâmetro só é necessário indicar o nome do arquivo, pelo qual não é necessário indicar a extensão deste.

Coluna Inicial: Corresponde à posição horizontal inicial do texto que vai ser importado.

Coluna Final: Corresponde à posição horizontal final do texto que vai ser importado.

A diferença entre a coluna final e a coluna inicial, corresponde ao número de caracteres de cada dado que vai ser importado. Esta diferencia tem que ser maior a zero, pelo qual, sempre a coluna final tem que ser maior à coluna inicial.

Fila Inicial: Corresponde à posição vertical do primeiro dado que vai ser importado.

Fila Final: Corresponde à posição vertical do ultimo dado que vai ser importado.

A diferença entre a fila final e a fila inicial, corresponde ao número de dados que vão ser importados. Esta diferencia tem que ser maior a zero, pelo qual, sempre a fila final tem que ser maior à fila inicial.

Variável de armazenamento: Corresponde ao nome da variável na qual vão ficar armazenados os dados importados, sem embargo, tem que ter em conta que quando são importados mais de um dado, esta variável tem que ser de tipo **Array**, onde seu índice indicará a posição a partir da qual serão armazenados os dados. Por outra parte, quando são importados dados que não são numéricos, a variável de armazenamento tem que ser de tipo *String*.

A continuação no Exemplo 7.48 é importada uma serie de dados desde o arquivo de texto *Tabela1.TXT*, os quais posteriormente são armazenados no sistema (arquivos CSV). Este algoritmo vai calcular os gastos mensais e totais da tabela apresentada na Fig. 7.4. Para isso é adotado o seguinte procedimento.

A. **Identificar o nome do arquivo texto:** Neste caso o nome do arquivo é Tabela1.TXT.

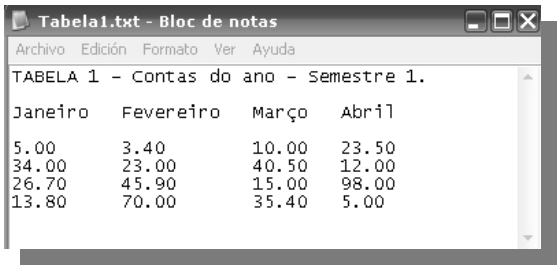


Fig. 7.4: Tabela de dados, Tabela1.TXT.

B. **Identificar no arquivo texto os dados que vai ser importados:** Para o Exemplo 7.48, vão ser importados os dados correspondentes aos meses de janeiro, fevereiro, março e abril.

D. Identificar a coluna inicial e final dos dados correspondentes ao mês de janeiro: Neste caso, na Fig. 7.4 a coluna inicial corresponda à posição 1, e a coluna final à posição 5.

E. Identificar a primeira e ultima fila dos dados correspondentes ao mês de janeiro: Neste caso, na Fig. 7.4 a fila inicial corresponde à posição 5, e a fila final à posição 8. Esta faixa de valores, indica que vão ser importados 4 dados para o mês de janeiro.

O procedimento dos itens 3 e 4 é repetido para os meses de fevereiro, março e abril.

Ao final, os valores das colunas e das filas dos dados que vão ser importados ficaram da seguinte maneira (Tab. 7.15):

	Coluna Inicial	Coluna Final	Fila Inicial	Fila Final	Número de dados
Janeiro	1	5	5	8	4
Fevereiro	11	15	5	8	4
Março	23	27	5	8	4
Abril	31	35	5	8	4

Tab. 7.15: Tabela de fila e colunas dos dados do Exemplo 7.48

F. Identificar o tipo de dados que vão ser importados: Este item é muito importante, já que se os dados importados são de tipo *String*, eles não podem ser armazenados numa variável numérica. Neste caso os dados importados são numéricos, pelo qual, eles podem ser armazenados em variáveis numéricas ou de tipo *String*.

G. Desenvolver o algoritmo.

```
\ BEGIN;                                    %% Inicio do programa.
%% é definido o comando para importar os dados correspondentes ao mês de janeiro,
```

```
%% utilizando os parâmetros calculados anteriormente.
  \ ImportText { Tabela1, 1, 5, 5, 8, V_N Janeiro[0] };
%% Dados mês de fevereiro.
  \ ImportText { Tabela1, 11, 15, 5, 8, V_N Fevereiro[0] };
%% Dados mês de março.
  \ ImportText { Tabela1, 23, 27, 5, 8, V_N Marco[0] };
%% Dados mês de abril.
  \ ImportText { Tabela1, 31, 35, 5, 8, V_N Abril[0] };
%% Gastos totais de cada mês.
  \ FOR { i = 0 : 3 };
    V_N T[i] = V_N T [i] + V_N Janeiro[i];              %% Gastos Janeiro
    V_N T[i] = V_N T [i] + V_N Fevereiro[i];          %% Gastos Fevereiro
    V_N T[i] = V_N T [i] + V_N Marco[i];               %% Gastos Março
    V_N T[i] = V_N T [i] + V_N Abril[i];               %% Gastos Abril
  \ CONTINUE;
  V_N T[4] = V_N T[0]+ V_N T[1]+ V_N T[2]+ V_N T[3];   %% Gastos totais do semestre.
  \ Print { "GASTOS" };
  \N;
  \ Print { "Janeiro:"+ V_N T[0]+ "REAIS" };
  \ Print { "Fevereiro:"+ V_N T[1]+ "REAIS" };
  \ Print { "Março:" + V_N T[2]+ "REAIS" };
  \ Print { "Abril:" + V_N T[3]+ "REAIS" };
  \N;
  \ Print { "Totais:" + V_N T[4]+ "REAIS" };
  \ SaveMemory { T1, V_N Janeiro };                    %% Armazenamento de dados no sistema
  \ SaveMemory { T2, V_N Fevereiro };
  \ SaveMemory { T3, V_N Marco };
  \ SaveMemory { T4, V_N Abril };
  \ SaveMemory { TT, V_N T };
\ END;                                                 %% Fim do programa
```

Exemplo 7.48: Uso da instrução ImportText.

No Exemplo 7.48, a instrução *ImportText* permitiu importar alguns dados numéricos do arquivo de texto Tabela1.TXT, com os quais foram feitas algumas operações. Os dados importados e os resultados das distintas operações foram armazenas em arquivos CSV (T1, T2, T3, T4, TT) no sistema.

7.9.2. Instrução ExportText.

A instrução ExportText permite exportar dados numéricos ou de tipo *String*, para um arquivo de texto TXT, novo ou já existente. Os dados exportados podem ser inseridos ou sobrescritos no arquivo dependendo dos parâmetros definidos na instrução. Esta instrução é definida da seguinte forma:

\ **ExportText** { < Parâmetros > } ;

Na Tab. 7.16 é apresentada a estrutura dos parâmetros da instrução, os quais indicam a origem e o destino dos dados.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ ExportText { Arquivo , A, B, Variavel, True } ;	Arquivo	Nome do arquivo TXT do qual serão importados os dados.
	<Vírgula>	Caractere de separação.
	A	Coluna Inicial
	<Vírgula>	Caractere de separação.
	B	Coluna Final
	<Vírgula>	Caractere de separação.
	Variavel	Variável que vai ser exportada
	<Vírgula>	Caractere de separação.
	True	Este parâmetro é opcional, se ele é definido como True, os dados são sobrescritos, senão ele não é definido então os dados são inseridos mas não sobrescritos.

Tab. 7.16: Estrutura dos parâmetros da instrução ExportText.

Arquivo: Corresponde ao nome do arquivo de extensão TXT, ao qual vão ser exportados os dados. Se o arquivo não existe, será criado um arquivo novo com o nome definido no

parâmetro. Para definir este parâmetro só é necessário indicar o nome do arquivo, pelo qual não é necessário indicar a extensão deste.

Coluna Destino: Corresponde à posição horizontal no arquivo, na qual será inserido ou sobrescrito o dado que vai ser exportado.

Fila Destino: Corresponde à posição vertical no arquivo, na qual será inserido ou sobrescrito o dado que vai ser exportado.

Variável ou dado a exportar: Corresponde ao nome da variável que vai ser exportada, se a variável não existe ou não está definida, o programa identificará este parâmetro como um dado tipo *String* e, por conseguinte este será exportado como texto.

Parâmetro de configuração: Este parâmetro é opcional, se ele é definido como True então os dados são sobrescritos, senão eles são inseridos.

No Exemplo 7.49 são exportados para o arquivo Resultados.TXT os dados que foram importados e armazenados no Exemplo 7.48, para fazer isto foi adotado o seguinte procedimento.

A. Definir o nome de arquivo destino: Resultados.TXT.

B. Definir as posições no arquivo Resultados.TXT, nas quais vão ser inseridos os dados (Tab. 7.17)

	Coluna	Fila	Número de dados
Texto: "Tabela de resultados"	1	2	1
Texto: "Janeiro"	1	4	1
Dados do arquivo T1	1	6-9	4
Dados do arquivo TT[0]	1	11	1
Texto: "Fevereiro"	10	4	1
Dados do arquivo T2	10	6-9	4
Dados do arquivo TT[1]	10	11	1
Texto: "Marco"	21	4	1
Dados do arquivo T3	21	6-9	4
Dados do arquivo TT[2]	21	11	1
Texto: "Abril"	30	4	1
Dados do arquivo T4	30	6-9	4

Dados do arquivo TT[3]	30	11	1
Texto: “Gastos totais”	1	13	1
Dados do arquivo TT[4]	10	13	1

Tab. 7.17:Exemplo de configuração de dados da instrução ExportText.

C. Definir os dados que vão ser exportados.

D. Criar o algoritmo.

```
\ BEGIN;                                    %% Inicio do programa.

  \ ReadMemory { T1 };                      %% Carga de arquivos CSV.
  \ ReadMemory { T2 };
  \ ReadMemory { T3 };
  \ ReadMemory { T4 };
  \ ReadMemory { TT };

%% exporta dados
  \ ExportText { Resultados, 1, 2, Tabela_de_resultados, true };
  \ ExportText { Resultados, 1, 4, Janeiro,true };
  \ ExportText { Resultados, 1, 11, V_N T[0], true };
  \ ExportText { Resultados, 10, 4, Fevereiro,true };
  \ ExportText { Resultados, 10, 11, V_N T[1], true };
  \ ExportText { Resultados, 21, 4, Marco,true };
  \ ExportText { Resultados, 21, 11, V_N T[2], true };
  \ ExportText { Resultados, 30, 4, Abril,true };
  \ ExportText { Resultados, 30, 11, V_N T[3], true };

  \FOR {i=0:3};
    a=i+6;
    \ ExportText { Resultados, 1, a, V_N Janeiro[i], true };
    \ ExportText { Resultados, 10, a, V_N Fevereiro[i], true };
    \ ExportText { Resultados, 21, a, V_N Marco[i], true };
    \ ExportText { Resultados, 30, a, V_N Abril[i], true };
  \ CONTINUE;

  \ ExportText { Resultados, 1, 13, Gastos_totais,true };
  \ ExportText { Resultados, 20, 13, V_N T[4], true };

\ END;
```

Exemplo 7.49: Uso da instrução ExportText.

No Exemplo 7.49 foram utilizados os dados gerados anteriormente, estes foram importados utilizando instruções *ReadMemory*, logo mediante a instrução *ExportText* foi gerado o arquivo Resultados.TXT o qual fico da seguinte forma (Fig. 7.5):

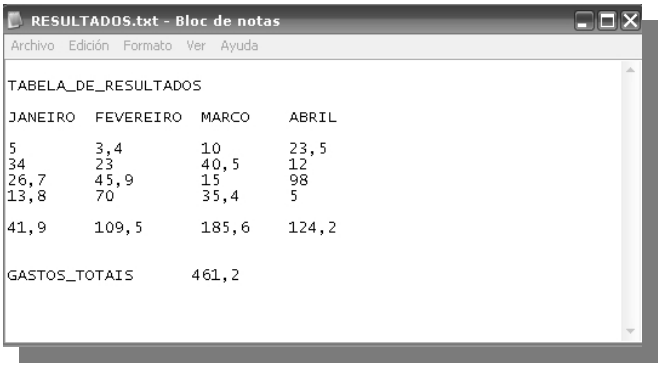


Fig. 7.5: Arquivo Resultados.TXT, arquivo gerado pela instrução ExportText.

7.9.3. Instrução ImportPlanivox

A instrução *ImportPlanivox* permite importar dados numéricos ou de tipo *String* desde um arquivo de texto CSV, os quais são armazenados numa variável simples ou num *Array* de dados dependendo do número de dados importados. Esta instrução é definida da seguinte forma:

\ **ImportPlanivox** { < Parâmetros > } ;

Na Tab. 7.18 é apresentada a estrutura dos parâmetros da instrução, os quais indicam a origem e o destino dos dados.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ ImportPlanivox { Arquivo , A, B, C, Variavel} ;	Arquivo	Nome do arquivo CSV do qual serão importados os dados.
	<Vírgula>	Caractere de separação.
	A	Coluna
	<Vírgula>	Caractere de separação.
	B	Fila Inicial
	<Vírgula>	Caractere de separação.
	C	Fila Final
	<Vírgula>	Caractere de separação.
	Variavel	Nome da variável que vai armazenar os dados importados.

Tab. 7.18: Estrutura dos parâmetros da instrução ImportPlanivox.

Arquivo: Corresponde ao nome do arquivo de extensão **CSV** do qual vão ser importados os dados. Se o arquivo não existe, o programa gerará um erro indicando a ausência deste. Para definir este parâmetro só é necessário indicar o nome do arquivo, pelo qual não é necessário indicar a extensão deste.

Coluna: Corresponde à posição horizontal do dado que vai ser importado.

Fila Inicial: Corresponde à posição vertical do primeiro dado que vai ser importado.

Fila Final: Corresponde à posição vertical do ultimo dado que vai ser importado.

A diferença entre a fila final e a fila inicial, corresponde ao número de dados que vão ser importados. Esta diferencia tem que ser maior a zero, pelo qual, sempre a fila final tem que ser maior à fila inicial.

Variável de armazenamento: Corresponde ao nome da variável na qual vão ficar armazenados os dados importados, sem embargo, tem que ter em conta que quando são importados mais de um dado, esta variável tem que ser de tipo **Array**, onde seu índice indicará a posição a partir da qual serão armazenados os dados. Por outra parte, quando são importados dados que não são numéricos, a variável de armazenamento tem que ser de tipo *String*.

No Exemplo 7.50 são importados dados numéricos e string desde o arquivo Planilha1.CSV e posteriormente são apresentados na tela do EDIVOX. O procedimento para realizar isto é muito similar ao procedimento da instrução ImportText e é explicado a continuação.

A. Identificar o nome do arquivo CSV: Neste caso o nome do arquivo é Planilha.CSV. (Fig. 7.6).

	A	B	C
1			
2			
3		Dados da planilha	
4			
5		Dado 1	34
6		Dado 2	23,3
7		Dado 3	12
8		Dado 4	43
9		Dado 5	545
10		Dado 6	233
11		Dado 7	1000
12		Dado 8	4

Fig. 7.6: Tabela de dados, Planilha.CSV.

B. Identificar no arquivo CSV os dados que vai ser importados: Para o Exemplo 7.50, vão ser importados os dados correspondentes as colunas B e C.

C. Identificar a primeira e ultima fila dos dados que vão ser importados: Neste caso, na Fig. 7.6 a fila inicial corresponde à posição 5, e a fila final à posição 12. Esta faixa de valores, indica que vão ser importados 8 dados.

D. Identificar o tipo de dados que vão ser importados: Para a coluna B os dados são *String* e para a coluna C os dados são numéricos.

E. Desenvolver o algoritmo.

```
\ BEGIN;                                          %% Inicio do programa.
  \ ImportPlanivox { Planilha1,2,5,12,V_S B [0] }; %% Importa dados.
```

```
  \ ImportPlanivox { Planilha1,3,5,12,V_N C[0] };

  \ FOR { i=0:7 };
    \ VarResult { V_S B [i], V_N C [i] };          %% Imprime dados.
  \CONTINUE;

\END;

V_SB[0] = Dado 1                                   %% Resultados do programa.
V_NC[0] = 34
V_SB[1] = Dado 2
V_NC[1] = 23,3
V_SB[2] = Dado 3
V_NC[2] = 12
V_SB[3] = Dado 4
V_NC[3] = 43
V_SB[4] = Dado 5
V_NC[4] = 545
V_SB[5] = Dado 6
V_NC[5] = 233
V_SB[6] = Dado 7
V_NC[6] = 1000
V_SB[7] = Dado 8
V_NC[7] = 4
```

Exemplo 7.50: Uso da instrução ImportPlanivox.

7.9.4. Instrução ExportPlanivox

A instrução *ExportPlanivox* permite exportar dados numéricos ou de tipo *String*, para um arquivo CSV, novo ou já existente. Esta instrução é definida da seguinte forma:

```
\ ExportPlanivox { < Parâmetros > } ;
```

Na Tab. 7.19 é apresentada a estrutura dos parâmetros da instrução, os quais indicam a origem e o destino dos dados.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ ExportPlanivox { Arquivo , A, B, Variavel, True } ;	Arquivo	Nome do arquivo CSV do qual serão importados os dados.
	<Vírgula>	Caractere de separação.
	A	Coluna Destino
	<Vírgula>	Caractere de separação.
	B	Fila Destino
	<Vírgula>	Caractere de separação.
	Variavel	Variável que vai ser exportada.

Tab. 7.19: Estrutura dos parâmetros da instrução ExportPlanivox.

Arquivo: Corresponde ao nome do arquivo de extensão CSV, ao qual vão ser exportados os dados. Se o arquivo não existe, será criado um arquivo novo com o nome definido no parâmetro. Para definir este parâmetro só é necessário indicar o nome do arquivo, pelo qual não é necessário indicar a extensão deste.

Coluna Destino: Corresponde à posição horizontal no arquivo, na qual será inserido o dado que vai ser exportado.

Fila Destino: Corresponde à posição vertical no arquivo, na qual será inserido o dado que vai ser exportado.

Variável ou dado a exportar: Corresponde ao nome da variável que vai ser exportada, se a variável não existe ou não está definida, o programa identificará este parâmetro como um dado tipo *String* e, por conseguinte este será exportado como texto.

No *Exemplo 7.51* é exportado um *Array* de dados para o arquivo Dados.CSV, este processo pode ser feito de duas maneiras as quais será apresentada no mesmo exemplo, o procedimento para realizar isto é o seguinte:

A. **Definir o nome de arquivo destino:** Dados.CSV.
B. Definir as posições no arquivo Dados.CSV, nas quais vão ser inseridos os dados (posições 2-2 e 4-2).

C. Definir os dados que vão ser exportados (V_N um).

D. Criar o algoritmo.

```
\ BEGIN;                                    %% Inicio do programa.

  \ FOR { I=0:10 };
    M_N UM [0,I] = I *4.52 – PI;
    A=2+i;

%% Forma um
    \ExportPlanivox { Dados,2,A,M_N UM[0,i]};
  \ CONTINUE;

%% Forma dois
  \ExportPlanivox { Dados,4,2,M_N UM };

\ END;
```

Exemplo 7.51: Uso da instrução ExportPlanivox.

O arquivo gerado no Exemplo 7.51 tem a seguinte apresentação (Fig. 7.7):

	A	B	C	D
1				
2		-3,14159265		-3,14159265
3		1,37840735		1,37840735
4		5,89840735		5,89840735
5		10,4184073		10,4184073
6		14,9384073		14,9384073
7		19,4584073		19,4584073
8		23,9784073		23,9784073
9		28,4984073		28,4984073
10		33,0184073		33,0184073
11		37,5384073		37,5384073
12		42,0584073		42,0584073

Fig. 7.7:Arquivo Dados.CSV, arquivo gerado pela instrução ExportPlanivox.

No *Exemplo 7.51*, a primeira forma de exportar os dados foi utilizando um ciclo *FOR*, isto permitiu exportar todos os dados de um em um, sem embargo quando vão ser exportados muitos dados, este procedimento pode ser muito demorado. Outra forma de fazer isto consiste em não definir o índice do *Array* (Forma dois), desta forma o programa vai assumir que tem que exportar todo o *Array*. Este procedimento diminui o tempo de execução, mas não permite escolher os dados do *Array* vão ser exportados, já que ele os exporta a todos eles.

7.10. Instrução Decimal.

A instrução *Decimal* permite configurar o número de casas decimais que serão apresentadas no sistema. Esta instrução é definida da seguinte forma:

```
\ Decimal { < Parâmetros > } ;
```

Na Tab. 7.20 é apresentada a estrutura dos parâmetros da instrução, os quais indicam o número de casas decimais.

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ Decimal { 4 } ;	4	Indica que são apresentadas 4 casas decimais.
\ Decimal { } ;	<Vazio>	São apresentadas todas as casas decimais.

Tab. 7.20: Estrutura dos parâmetros da instrução Decimal.

No Exemplo 7.52 o resultado da expressão é definida inicialmente com 5 casas decimais e posteriormente com todas elas.

```
\ BEGIN;                                    %% Inicio do programa.

  A = PI * 5;
%% Define 5 casas decimais
  \ Decimal {5};
  \ Print { "5"+"casas"+"decimais" };
  \ VarResult { A };
  \N;

%% Define todas as casas decimais
  \ Decimal {};
  \ Print { "casas"+"decimais"+"completas" };
  \ VarResult { A };

\ END;
5 CASAS DECIMAIS
A = 1,57080E+1

CASAS DECIMAIS COMPLETAS
A = 15,707963267949
```

Exemplo 7.52: Uso da instrução Decimal.

7.11. Instrução Evaluate

A instrução *Evaluate* permite avaliar uma função numa faixa de números definida pelo usuário e armazenar estes dados num arquivo CSV. Esta instrução é definida da seguinte forma:

```
\ Evaluate { < Parâmetros > };
```

Na Tab. 7.21 é apresentada a estrutura dos parâmetros da instrução:

INSTRUÇÃO	PARÂMETROS	DESCRIÇÃO
\ Evaluate { X + 2 : X: 1: 100: 1	X + 2	Função

: fun} ;	<Dois pontos>	Caractere de separação.
	X	Variavel independente
	<Dois pontos>	Caractere de separação.
	1	Valor inicial
	<Dois pontos>	Caractere de separação.
	100	Valor final.
	<Dois pontos>	Caractere de separação.
	1	Incremento
	<Dois pontos>	Caractere de separação.
	fun	Arquivo de armazenamento CSV

Tab. 7.21: Estrutura dos parâmetros da instrução Evaluate.

No Exemplo 7.53 é avaliada a função SENO em graus na faixa de 0° - 360°, e os resultados da avaliação são armazenados no arquivo funcaoSeno.CSV.

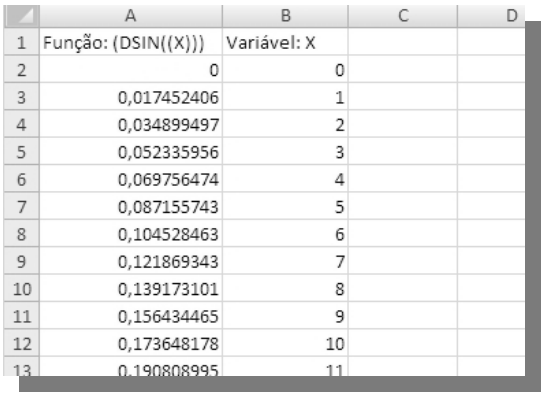
```
\ BEGIN;                                    %% Inicio do programa.

%% Avaliação da função Seno em graus
  \ Evaluate { DSin(X): X: 0: 360: 1: funcaoSeno };

\ END;
```

Exemplo 7.53: Uso da instrução Evaluate.

O arquivo CSV gerado tem a seguinte forma (Fig. 7.8), onde na primeira coluna estão os valores do resultado da avaliação da função e na segunda coluna os valores da variável independente.



	A	B	C	D
1	Função: (DSIN((X)))	Variável: X		
2	0	0		
3	0,017452406	1		
4	0,034899497	2		
5	0,052335956	3		
6	0,069756474	4		
7	0,087155743	5		
8	0,104528463	6		
9	0,121869343	7		
10	0,139173101	8		
11	0,156434465	9		
12	0,173648178	10		
13	0,190808995	11		

Fig. 7.8: Avaliação da função SENO em graus, arquivo FuncaoSeno.CSV.

7.12. Instrução ReadFunction

A instrução ReadFunction é utilizada para sintetizar auditivamente funções matemáticas. Esta instrução reproduze tonos a freqüências diferentes, de acordo aos valores de amplitude da função, onde os tonos mais agudos (freqüências maiores) representam amplitudes grandes e os tonos mais graves (freqüências menores) representam amplitudes pequenas.

Por outra parte a duração destes tonos dependem da pendente de cada um dos valores da função (tempos curtos para pendentes grandes, tempos compridos para pendentes pequenas).

Esta instrução só sintetiza as funções que tem sido avaliadas previamente mediante a instrução *Evaluate*, pelo qual o parâmetro de entrada da instrução *ReadFunction* corresponde ao arquivo CSV gerado mediante a instrução *Evaluate*. Esta instrução é definida da seguinte forma:

```
\ ReadFunction { < Arquivo CSV > } ;
```

A execução desta instrução é diferente à execução de todas as demais instruções, já que esta tem que ser executada de forma isolada. Para fazer isto só é necessário escrever a instrução na tela do EDIVOX e posteriormente pressionar as teclas CTRL + F9 (Exemplo 7.54).

```
\ ReadFunction { FuncaoSENO };
```

Exemplo 7.54: Uso da instrução ReadFunction, síntese do arquivo FuncaoSENO.CSV gerado pela instrução Evaluate.

A qualidade da síntese da função depende da resolução (quantidade de dados) que tem o arquivo CSV gerado pela instrução *Evaluate*, no Exemplo 7.55 são avaliadas duas funções SENO com resoluções diferentes, na primeira o incremento é de 2 pelo qual são gerados 180 dados e na segunda o incremento é de 10 pelo qual são gerados só 36 dados.

```
\ BEGIN;                                %% Inicio do programa.

%% Avaliação da função (Seno + 1) com 180 dados de resolução.
%% Incremento de 2
  \ Evaluate { DSin(X)+1: X: 0: 360: 2: Seno180 };

%% Avaliação da função (Seno+1) com 36 dados de resolução.
%% Incremento de 10
  \ Evaluate { DSin(X)+1: X: 0: 360: 10: Seno36};

\ END;
Resolução maior.
\ ReadFunction { Seno180 };

Resolução menor.
\ ReadFunction { Seno36 };
```

Exemplo 7.55: Uso da instrução ReadFunction, qualidade na síntese da função.

Na Fig. 7.9 é apresentado o gráfico da função do arquivo Seno180.CSV (alta resolução) e na Fig. 7.10 o gráfico do arquivo Seno36.CSV (baixa resolução). Comparando estes dois gráficos pode se observar que o primeiro tem uma definição maior que o segundo, pelo qual sua síntese vai ser muito melhor.

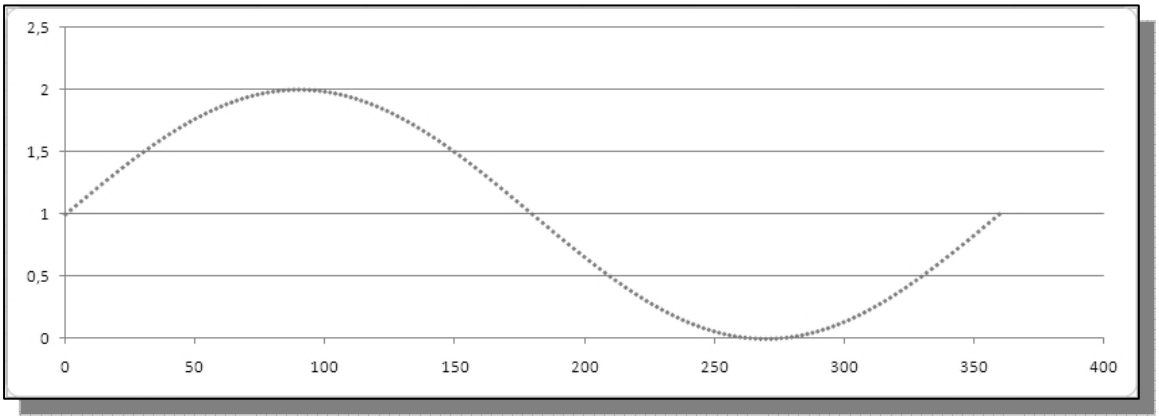


Fig. 7.9: Função do arquivo Seno180.CSV, alta resolução.

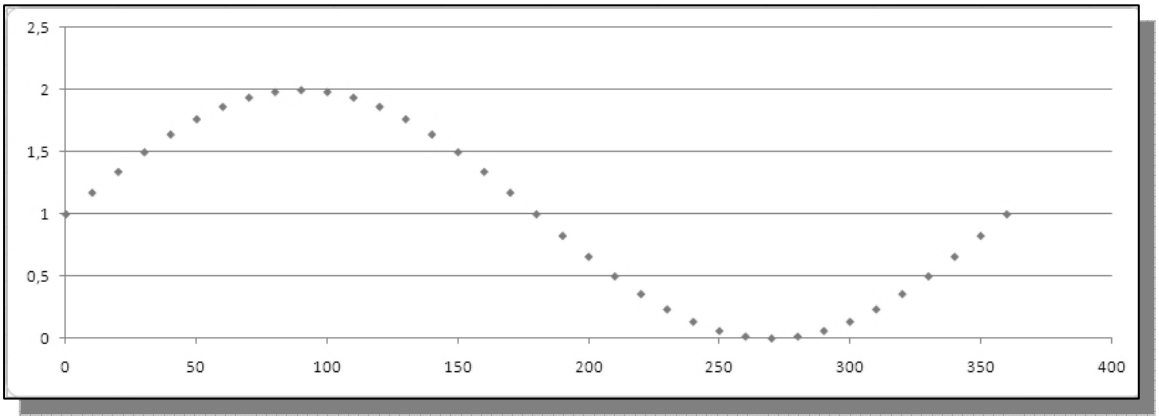


Fig. 7.10: Função do arquivo Seno36.CSV, baixa resolução.

7.13. Funções

O MATVOX inclui uma grande variedade de funções matemáticas, as quais podem ser utilizadas na definição de expressões. Estas funções podem se dividir principalmente em três grupos: *Funções gerais, Funções Trigonométricas e conversões de medida.*

Funções gerais: estas funções são principalmente funções exponenciais, de aproximação, de logaritmo, entre outras. Elas são apresentadas na Tab. 7.22.

FUNÇÃO	DESCRIÇÃO
LN (X)	Retorna o Logaritmo natural de X
EXP (X)	Retorna o Exponencial **"e"** de X
NEG (X)	Retorna o Negativo de X.
SQR (X)	Retorna o Quadrado de X.
SQRT(X)	Retorna a Raiz Quadrada de X.
ABS (X)	Retorna o Valor absoluto de X.
ROUND(X)	Retorna o redondeo de X.
TRUNC (X)	Retorna o valor sem fração de X.
FRAC (X)	Retorna a parte Decimal de X.
INT (X)	Retorna a parte Inteira de X.
(A) ROOT (B)	Retorna a Raiz A de B.
(A) POW (B)	Retorna a Potencia B de A
INC (X)	Retorna o incremento em um de X.
DEC (X)	Retorna o decremento em um de X.
LOG (X)	Retorna o Logaritmo em base 10 de X.
(A) LOGB (B)	Retorna o Logaritmo em base B de A.

Tab. 7.22: Funções Gerais.

O Exemplo 7.56 apresenta a forma correta de utilizar algumas destas instruções num algoritmo no EDIVOX. Quando a função tem só um parâmetro, este tem que estar encerrado por parênteses. Quando a função tem dois parâmetros, os parênteses são obrigatórios só quando o parâmetro não é constante.

```
\ BEGIN;                                    %% Inicio do programa.
  A = LN (5);
  B = LN (A);
  A1 = 2;
%% Funcoes com dois parâmetros.
  C = 2 Root 16;
  D = (A1) Root (16);
  E = 2 Pow (A1 + 1);
  F = (3) LogB 10;

  \ VarResult { A, B, C, D, E, F };

\END;                                       %% Fim do programa.

A = 1,6094379124341
B = 0,47588499532711
C = 4
D = 4
E = 8
F = 0,477121254719662
```

Exemplo 7.56: Uso de Funções Gerais.

Funções Trigonométricas: estas funções estão compostas por as funções trigonométricas básicas, inversas e hiperbólicas as quais podem ser definidas em graus ou radianos. Elas são apresentadas nas seguintes tabelas.

Funções Trigonométricas básicas: Tab. 7.23**.**

FUNÇÃO	DESCRIÇÃO
RCOS (X)	Retorna o cosseno em radianos de X.
RSIN (X)	Retorna o seno em radianos de X.
RTAN (X)	Retorna a tangente em radianos de X.
RCOT (X)	Retorna a cotangente em radianos de X.
RSEC (X)	Retorna a secante em radianos de X.

RCSC (X)	Retorna a cossecante em radianos de X.
DCOS (X)	Retorna o cosseno em graus de X.
DSIN (X)	Retorna o seno em graus de X.
DTAN (X)	Retorna a tangente em graus de X.
DCOT (X)	Retorna a cotangente em graus de X.
DSEC (X)	Retorna a secante em graus de X.
DCSC (X)	Retorna a cossecante em graus de X.

Tab. 7.23: Funções Trigonométricas Básicas.

Funções Trigonométricas Inversas: Tab. 7.24.

FUNÇÃO	DESCRIÇÃO
RACOS (X)	Retorna o arco cosseno em radianos de X.
RASIN (X)	Retorna o arco seno em radianos de X.
RATAN (X)	Retorna a arco tangente em radianos de X.
RACOT (X)	Retorna a arco cotangente em radianos de X.
RASEC (X)	Retorna a arco secante em radianos de X.
RACSC (X)	Retorna a arco cossecante em radianos de X.
DACOS (X)	Retorna o arco cosseno em graus de X.
DASIN (X)	Retorna o arco seno em graus de X.
DATAN (X)	Retorna a arco tangente em graus de X.
DACOT (X)	Retorna a arco cotangente em graus de X.
DASEC (X)	Retorna a arco secante em graus de X.
DACSC (X)	Retorna a arco cossecante em graus de X.

Tab. 7.24: Funções Trigonométricas Inversas.

Funções Trigonométricas Hiperbólicas: Tab. 7.25.

FUNÇÃO	DESCRIÇÃO
RCOSH (X)	Retorna o cosseno hiperbólico em radianos de X.

RSINH (X)	Retorna o seno hiperbólico em radianos de X.
RTANH (X)	Retorna a tangente hiperbólica em radianos de X.
RCOTH (X)	Retorna a cotangente hiperbólica em radianos de X.
RSECH (X)	Retorna a secante hiperbólica em radianos de X.
RCSCH (X)	Retorna a cossecante hiperbólica em radianos de X.
DCOSH (X)	Retorna o cosseno hiperbólico em graus de X.
DSINH(X)	Retorna o seno hiperbólico em graus de X.
DTANH (X)	Retorna a tangente hiperbólica em graus de X.
DCOTH (X)	Retorna a cotangente hiperbólica em graus de X.
DSECH (X)	Retorna a secante hiperbólica em graus de X.
DCSCH (X)	Retorna a cossecante hiperbólica em graus de X.

Tab. 7.25:Funções Trigonométricas Hiperbólicas.

Funções Trigonométricas Hiperbólicas Inversas: Tab. 7.26.

FUNÇÃO	DESCRIÇÃO
RACOSH (X)	Retorna o arco cosseno hiperbólico em radianos de X.
RASINH (X)	Retorna o arco seno hiperbólico em radianos de X.
RATANH (X)	Retorna a arco tangente hiperbólica em radianos de X.
RACOTH (X)	Retorna a arco cotangente hiperbólica em radianos de X.
RASECH (X)	Retorna a arco secante hiperbólica em radianos de X.
RACSCH(X)	Retorna a arco cossecante hiperbólica em radianos de X.
DACOSH (X)	Retorna o arco cosseno hiperbólico em graus de X.
DASINH (X)	Retorna o arco seno hiperbólico em graus de X.
DATANH (X)	Retorna a arco tangente hiperbólica em graus de X.
DACOTH (X)	Retorna a arco cotangente hiperbólica em graus de X.
DASECH (X)	Retorna a arco secante hiperbólica em graus de X.
DACSCH (X)	Retorna a arco cossecante hiperbólica em graus de X.

Tab. 7.26: Funções Trigonométricas Hiperbólicas Inversas.

Conversões: é um tipo de função que permite fazer conversões entre diferentes unidades de medidas. Elas podem se classificar da seguinte forma:

Conversões de Comprimento: Tab. 7.27.

FUNÇÃO	DESCRIÇÃO
M_IN (X)	Converte X Metros para Polegadas
M_FT (X)	Converte X Metros para Pés.
M_YD (X)	Converte X Metros para Jardas.
IN_M (X)	Converte X Polegadas para Metros.
FT_M (X)	Converte X Pés para Metros.
YD_M (X)	Converte X Jardas para Metros.

Tab. 7.27: Conversões de Comprimento.

Conversões de Massa: Tab. 7.28.

FUNÇÃO	DESCRIÇÃO
KG_OZ (X)	Converte X Quilogramas para Onças.
KG_LB (X)	Converte X Quilogramas para Libras.
KG_TN (X)	Converte X Quilogramas para Toneladas.
OZ_KG (X)	Converte X Onças para Quilogramas.
LB_KG (X)	Converte X Libras para Quilogramas.
TN_KG (X)	Converte X Toneladas para Quilogramas.

Tab. 7.28: Conversões de Massa.

Conversões de Volume: Tab. 7.29.

FUNÇÃO	DESCRIÇÃO
L_OZ (X)	Converte X Litros para Onças.
L_GAL (X)	Converte X Litros para Galões.
OZ_L (X)	Converte X Onças para Litros.
GAL_L (X)	Converte X Galões para Litros.

Tab. 7.29: Conversões de Volume.

Conversões de Pressão: Tab. 7.30.

FUNÇÃO	DESCRIÇÃO
ATM_BAR (X)	Converte X Atmosferas para Bares.
ATM_KPA (X)	Converte X Atmosferas para Quilo Pascais.
ATM_PSI (X)	Converte X Atmosferas para Libras por Polegada quadrada.
ATM_TORR (X)	Converte X Atmosferas para Torricelli.
BAR_ATM (X)	Converte X Bares para Atmosferas.
KPA_ATM (X)	Converte X Quilo Pascais para Atmosferas.
PSI_ATM (X)	Converte X Libras por Polegada quadrada para Atmosferas.
TORR_ATM (X)	Converte X Torricelli para Atmosferas.

Tab. 7.30: Conversões de Pressão.

Conversões de Potência: Tab. 7.31.

FUNÇÃO	DESCRIÇÃO
W_HP (X)	Converte X Watts para Cavalos de força.
W_CALS (X)	Converte X Watts para Calorias por segundo.
HP_W (X)	Converte X Calorias por segundo para Watts.
CALS_W (X)	Converte X Calorias por segundo para Watts.

Tab. 7.31: Conversões de Potência.

Conversões de Velocidade: Tab. 7.32.

FUNÇÃO	DESCRIÇÃO
MS_FTS (X)	Converte X Metros por segundo para Pés por segundo .
MS_KN (X)	Converte X Metros por segundo para Nós.
MS_MPH (X)	Converte X Metros por segundo para Milhas por hora.
MS_KPH (X)	Converte X Metros por segundo para Quilômetros por hora.
FTS_MS (X)	Converte X Pés por segundo para Metros por segundo.
KN_MS (X)	Converte X Nós para Metros por segundo.
MPH_MS (X)	Converte X Milhas por hora para Metros por segundo.

KPH_MS (X)	Converte X Quilômetros por hora para Metros por segundo.

Tab. 7.32: Conversões de Velocidade.

Conversões de Força: Tab. 7.33.

FUNÇÃO	DESCRIÇÃO
N_DYN (X)	Converte X Newton para Dinas.
N_JM (X)	Converte X Newton para Joules por metro.
N_KIN (X)	Converte X Newton para Quilo Polegadas.
DYN_N (X)	Converte X Dinas para Newton.
JM_N (X)	Converte X Joules por metro para Newton.
KIN_N (X)	Converte X Quilo Polegadas para Newton.

Tab. 7.33: Conversões de Força.

Conversões de Temperatura: Tab. 7.34.

FUNÇÃO	DESCRIÇÃO
DC_DK (X)	Converte X graus Celsius para graus Kelvin.
DC_DF (X)	Converte X graus Celsius para graus Fahrenheit.
DK_DC (X)	Converte X graus Kelvin para graus Celsius.
DF_DC (X)	Converte X graus Fahrenheit para graus Celsius.
DK_DF (X)	Converte X graus Kelvin para graus Fahrenheit.
DF_DK (X)	Converte X graus Fahrenheit para graus Kelvin.

Tab. 7.34: Conversões de Temperatura.

Conversões de Ângulo: Tab. 7.35.

FUNÇÃO	DESCRIÇÃO
D_RAD	Converte de Graus para Radianos.
RAD_D	Converte de Radianos para Graus.

Tab. 7.35: Conversões de Ângulo.

7.14. Resumo de Instruções, tipos de Variáveis e Operadores.

A continuação nas seguintes tabelas são apresentadas todas as instruções, todos os tipos de variáveis e todos os operadores disponíveis no MATVOX.

Tipo de Variáveis: Tab. 7.36

VARIÁVEL	DEFINIÇÃO	DESCRIÇÃO
Numérica	< Sem definição>	Armazena números reais.
STR	STR Variável	Variável tipo STRING permite armazenar cadeias de caracteres.
V_N	V_N Variavel [X]	Permite agrupar dados numéricos num array de uma dimensão.
V_S	V_N Variavel [X]	Permite agrupar variáveis de tipo STRING num array de uma dimensão.
M_N	V_N Variavel [X , Y]	Permite agrupar dados numéricos num array de duas dimensões.
M_S	V_N Variavel [X , Y]	Permite agrupar variáveis de tipo STRING num array de duas dimensões.
IN	IN Variavel	Variável de definição dinâmica.

Tab. 7.36: Tipos de variáveis do MATVOX

Operadores: Tab. 7.37

OPERADOR	DESCRIÇÃO
+	Operador de Soma.
-	Operador de Resta.
*	Operador de Multiplicação.
/	Operador de Divisão.

>	Operador Maior.
<	Operador Menor.
>=	Operador Maior ou Igual.
<=	Operador Menor ou Igual.
<>	Operador Diferente
=	Operador Igual.

Tab. 7.37: Operadores do MATVOX

Instruções: (X => Parâmetros da função) Tab. 7.38**.**

INSTRUÇÃO	DESCRIÇÃO
\ BEGIN ;	Indica o inicio do programa no EDIVOX.
\ END ;	Indica o final do programa no EDIVOX.
\ VarResult { X } ;	Permite imprimir na tela do EDIVOX, o valor de uma ou mais variáveis.
\ Print { X } ;	Permite imprimir na tela do EDIVOX, uma cadeia de texto composta por variáveis e/ou Strings.
\ StepResult { X } ;	Permite imprimir na tela do EDIVOX, o valor das sub expressões de uma expressão.
\ N ;	Permite imprimir um salto de linha.
\ GOTO { X } ;	Transfere o fluxo do programa para a instrução seguinte à etiqueta definida nesta.
\ IF { X } ;	Executa um bloco de instruções quando a condição definida previamente é verdadeira.
\ ELSE ;	Executa um bloco de instruções quando a condição definida na instrução IF é falsa.
\ WHILE { X } ;	Executa um bloco de instruções repetidamente quando a condição definida previamente é verdadeira.
\ FOR { X } ;	Executa um bloco de instruções determinado número de vezes.
\ SaveConstant { X } ;	Permite salvar no sistema uma ou mais variáveis.