CENTRO UNIVERSITÁRIO FEEVALE

MÁRCIO MIGUEL GOMES

COMPILADOR APLICADO À EXTRAÇÃO DE DADOS DE ANÁLISES AMBIENTAIS

Novo Hamburgo, dezembro de 2007.

MÁRCIO MIGUEL GOMES

COMPILADOR APLICADO À EXTRAÇÃO DE DADOS DE ANÁLISES AMBIENTAIS

Centro Universitário Feevale
Instituto de Ciências Exatas e Tecnológicas
Curso de Ciência da Computação
Trabalho de Conclusão de Curso

Professor Orientador: Ricardo Ferreira de Oliveira

Novo Hamburgo, dezembro de 2007.

AGRADECIMENTOS

A Deus pela vida e saúde, aos pais André e Felícita pelo amor, carinho e educação, aos irmãos Adriano, Regina e Henrique pelo companheirismo, aos mestres pelo conhecimento, aos amigos pela alegria e descontração, e a minha esposa Elisabeth, simplesmente por existir.

RESUMO

Os avanços da ciência permitiram o desenvolvimento de novas técnicas analíticas e isto se refletiu na evolução tecnológica dos equipamentos de análises laboratoriais. Juntamente com o crescimento da informática, criou-se um cenário propício para a automação da coleta de dados brutos, extração de informações e armazenamento de dados. Hoje não existe um padrão mundial para a disposição e armazenamento desses dados resultantes das análises laboratoriais, embora diversas normas e certificações exijam o armazenamento desses dados para questões de rastreabilidade. A automação na extração dos dados brutos e a padronização no armazenamento dos dados processados proporcionam inúmeras vantagens, como integridade, confidencialidade e rapidez. Isto reflete para o laboratório em forma de maior capacidade produtiva, menor quantidade de falhas operacionais e de transcrição de dados e principalmente em redução de custos, que resulta em uma preciosa vantagem competitiva. Este estudo tem por objetivo propor um modelo de compilador capaz de interpretar dados brutos de diversos equipamentos analíticos e armazená-los em banco de dados para uso futuro.

Palavras-chave: Aquisição de dados. Extração de dados. Automação de laboratórios. Compiladores.

ABSTRACT

The advances of science have allowed the development of new analytical techniques and this is reflected in the technological evolution of the analytical equipments. With the growth of computer science, it was created an ideal scene for the automation of raw data acquisition, information extraction and data storage. Nowadays, does not exist a world-wide standard for the presentation and storage of data from laboratories analyses, even so, a several norms and certifications demands the storage of these data for tracking reasons. The automation in the extraction of raw data and the standardization in the storage of the processed data provide a lot of advantages, as integrity, confidentiality and speed. This reflects for the laboratory as a bigger productive capacity, lesser amount of operational errors in data transcription and mainly in reduction of costs, which results in a precious competitive advantage. This research has as objective to consider a model of compiler capable to process raw data from diverse analytical equipments and to store them in data base for future use.

Keywords: Data Acquisition. Data Extraction. Laboratories automation. Compilers.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AFD	Autômato Finito Determinístico (AHO et al, 1995, p.52)
AFN	Autômato Finito Não-Determinístico (AHO et al, 1995, p.52)
AM	Ante Meridiem ou antes do meio dia
ASCII	American Standard Code for Information Interchange ou Código Padrão Americano Para Troca de Informações
BNF	Forma de Backus-Naur ou Forma Normal de Backus (GRUNE et al, 2001, p.34)
FDA	Food and Drug Administration ou Administração de Alimentos e Drogas
HTML	HyperText Markup Language ou Linguagem de Marcação de Hipertexto
ISO	International Standardization Organization ou Organização Internacional para Padronização
PC	Personal Computer ou Computador Pessoal
PM	Post Meridiem ou após o meio dia
RDC	Resolução da Diretoria Colegiada

SUMÁRIO

INTRODUÇÃO

O avanço do uso da informática em todas as áreas da sociedade nas últimas décadas também ocorreu para laboratórios de controle de qualidade. Equipamentos analíticos tecnologicamente ultrapassados e técnicas manuais foram gradativamente sendo substituídos por equipamentos atualizados e com novos recursos disponíveis.

Em decorrência de um mercado cada vez mais competitivo e da alta exigência de qualidade, muitos laboratórios tiveram nos últimos anos um incremento significativo do número de amostras e das análises realizadas e, com a pressão pela redução de custos, reduziram suas equipes ou as mantiveram igual no melhor dos casos.

Em função dos ambientes regulamentados por normas como a ISO 17025, RDC 210 e FDA, a carga de trabalho administrativo dos analistas aumentou drasticamente em função da necessidade de incremento de registros necessários para garantir e evidenciar a confiabilidade e rastreabilidade dos resultados gerados pelo laboratório.

O aumento da rigidez nessas normas que regem o controle de qualidade de laboratórios de análises, juntamente com o aumento da concorrência no mercado, impulsionaram a necessidade de informatização e automatização de processos.

Assim, as empresas investiram em novas técnicas analíticas ou equipamentos que tornaram o processo de análise mais rápido e confiável, resultando em aumento de qualidade, confiabilidade e conseqüentemente lucratividade dos laboratórios. Entretanto, na maioria dos casos, todo esse volume de medições e dados gerados continua sendo coletado, processado, gerenciado e armazenado de forma manual.

Uma das maneiras de aumentar a confiabilidade e reduzir o tempo de processamento dos resultados de análises consiste em automatizar a leitura e transcrição dos resultados

encontrados durante a realização das análises. Processos até então manuais de transcrição de dados, com um enorme potencial de falha humana e desperdício de tempo podem ser substituídos por processos informatizados que reduzem o erro a níveis mínimos e elevam a produtividade a níveis nunca antes imaginados.

Os equipamentos analíticos comercializados atualmente possuem recursos que vão muito além da realização da análise propriamente dita. Uma grande quantidade de equipamentos é totalmente automatizada por algum *software* que roda em um PC externo ao equipamento analítico. Outros equipamentos possuem processadores ou controladores internos, mas disponibilizam formas de comunicação de dados com o ambiente externo.

Todos esses equipamentos possuem uma forma eletrônica de apresentação dos resultados obtidos no processo de análise. Normalmente equipamentos que possuem um PC e *software* de controle possuem também uma maneira de exportar os dados, como arquivos texto, por exemplo. Já equipamentos com controladores internos comumente disponibilizam seus dados por uma interface de comunicação serial ou paralela.

Os dados exportados pelos equipamentos na maioria dos casos não apresentam uma estrutura simples, tampouco possuem padronização de formato entre fabricantes distintos. Também ocorre de equipamentos distintos de um mesmo fabricante apresentarem formatos de relatórios completamente diferentes. Mesmo assim, esses dados brutos são estruturados e podem perfeitamente ser interpretados por um *software* preparado especialmente para isso.

Para um *software* conseguir interpretar e extrair informações a partir desses relatórios ele dever ter a capacidade de reconhecer os padrões e as estruturas desses arquivos de dados. Ele deverá ser completo o bastante a ponto de perceber variações do padrão e estrutura, e mesmo assim conseguir interpretar os dados e extrair as informações desejadas.

GRUNE et al (2001, p.7) mostram que “As técnicas de construção de compiladores podem ser e são aplicadas fora da construção de compiladores em seu sentido mais estrito. [...] Os exemplos são a leitura de dados estruturados, a introdução rápida de novos formatos e os problemas gerais de conversão de arquivos”.

Segundo KAPLAN (1994), linguagens pequenas são ferramentas importantes para programadores e usuários de sistemas. Uma situação onde se aplica a construção de uma

pequena linguagem é uma aplicação que necessita executar uma tarefa repetidas vezes, com pequenas variações, como um arquivo de entrada.

Assim, o objetivo principal deste trabalho concentra-se no estudo de compiladores para serem usados na interpretação de dados brutos e extração de informações, e não para serem usados na sua maneira mais clássica, ou seja, leitura de uma linguagem de programação de alto nível e estruturada e geração de código de máquina executável.

A primeira seção deste trabalho registra o estudo teórico dos compiladores com o objetivo de adquirir conhecimento suficiente para certificar-se que é possível utilizar um compilador na extração de informações a partir de dados brutos. Esse estudo também serve de base para a etapa complementar deste trabalho, que é a construção de um protótipo do compilador.

A segunda seção apresenta a realidade geral dos laboratórios de análises ambientais nas regiões Sul e Sudeste do Brasil, embasado em experiência própria durante nove anos, em trabalhos diários com automação de diversos processos em dezenas de laboratórios de controle de qualidade.

A terceira seção define um modelo de relatório de dados brutos a ser utilizado como base para o projeto e desenvolvimento do protótipo do compilador e segue com diversos estudos em cima de problemas reais de identificação de padrões e formas de extração de informações a partir dos dados brutos.

1 COMPILADORES

> “Em sua forma mais geral, um compilador é um programa que aceita como entrada um texto de programa em uma certa linguagem e produz como saída um texto de programa em outra linguagem, enquanto preserva o significado deste texto. Este processo é chamado de tradução, como seria denominado se os textos estivessem em linguagens naturais.” (GRUNE et al, 2001, p.1).

Um compilador, segundo AHO et al (1995), é um programa que lê um programa em uma linguagem fonte e o traduz em uma outra linguagem alvo. (ver Figura 1.1)

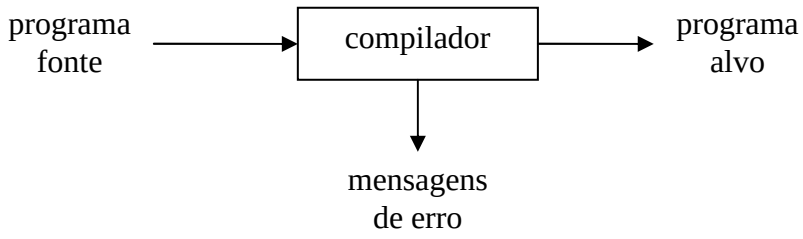

Figura 1.1 - Um compilador. (AHO et al, 1995, p.1).

A compilação pode ser dividida em duas grandes partes: análise e síntese. Segundo AHO et al (1995), a etapa de análise divide o programa fonte em pequenas partes e cria representações intermediárias para essas partes, enquanto a síntese constrói o programa alvo desejado a partir dessas representações intermediárias.

Para RANGEL (1999), a sintaxe é associada à idéia de forma, em oposição à semântica, associada a significado e conteúdo. Assim, em princípio, a sintaxe de uma linguagem de programação deve descrever todos os aspectos relativos à forma de construção de programas corretos na linguagem, enquanto a semântica deve descrever o que acontece quando o programa é executado. Portanto, toda a análise está relacionada com sintaxe, e a semântica deveria corresponder apenas à geração de código, que deve preservar o significado do programa fonte, construindo um programa objeto com o mesmo significado.

1.1 Gramáticas

> “As gramáticas ou, mais precisamente gramáticas livres de contexto, constituem o formalismo essencial para descrever a estrutura de programas em uma linguagem de programação. Em princípio, a gramática de uma linguagem descreve apenas a estrutura sintática, mas como a semântica de uma linguagem é definida em termos de sintaxe, a gramática também é instrumental na definição da semântica.” (GRUNE et al, 2001, p.31).

RANGEL (1999) afirma que quase universalmente, a sintaxe das linguagens de programação é descrita por gramáticas livres de contexto, em uma notação chamada BNF (Forma de Backus-Naur ou ainda Forma Normal de Backus), ou em alguma variante ou extensão dessa notação.

Uma gramática consiste em um conjunto de regras de produção e um símbolo de partida. GRUNE (2001) estrutura uma gramática da seguinte forma:

- Cada regra de produção define uma construção sintática nomeada;

- Uma regra de produção consiste em duas partes: um lado esquerdo e um lado direito, separados por um caractere especial;

- O lado esquerdo é o nome da construção sintática e o lado direito mostra uma forma possível da construção sintática;

- O lado direito de uma regra de produção pode conter dois tipos de símbolos: terminais e não terminais. Símbolo terminal é um ponto final do processo de produção.

Um exemplo de regra de produção é:

expressão → '(' expressão operador expressão ')'

A Figura 1.2 mostra a forma de derivação da string (1*(1+1)).

```
expressão
'(' expressão operador expressão ')'
'(' '1' operador expressão ')'
'(' '1' '*' expressão ')'
'(' '1' '*' '(' expressão operador expressão ')' ')'
'(' '1' '*' '(' '1' operador expressão ')' ')'
'(' '1' '*' '(' '1' '+' expressão ')' ')'
'(' '1' '*' '(' '1' '+' '1' ')' ')'
```

Figura 1.2 - Derivação da string (1*(1+1)) (GRUNE et al, 2001, p.33).

1.2 Análise

A análise é a etapa da compilação em que é verificado se o programa fonte pertence a uma gramática, e é composta por 3 fases: léxica, sintática e semântica.

RANGEL (1999) afirma que a análise léxica tem como finalidade a separação e identificação dos elementos componentes do programa fonte e normalmente esses componentes são especificados através de expressões regulares. A análise sintática deve reconhecer a estrutura global do programa, descrita através de gramáticas livres de contexto. A análise semântica se encarrega da verificação das regras restantes. Essas regras tratam quase sempre da verificação de que os objetos são usados no programa da maneira prevista em suas declarações, por exemplo, verificando que não há erros de tipos.

1.2.1 Análise léxica

> "Após a análise léxica, itens como identificadores, operadores, delimitadores, palavras chave e palavras reservadas estão identificados normalmente através de duas informações: um código numérico e uma cadeia de símbolos. Estes itens são conhecidos como componentes léxicos do programa, lexemas, ou ainda tokens." (RANGEL, 1999).

A análise léxica é a primeira etapa de um compilador e serve para separar e classificar os elementos de um programa fonte. Ela também elimina elementos decorativos e de organização visual, como espaços, quebras de linha e comentários. Pode ser subdividida em duas tarefas: varredura ou *scanner* e análise léxica.

A varredura consiste em ler caractere por caractere do programa fonte e entregá-los ao analisador léxico, que por sua vez realiza a análise do conjunto de caracteres que recebeu do *scanner* para identificar lexemas conhecidos da gramática. Essa identificação pode ser feita através de uma técnica de autômatos finitos - também chamada de máquina de estados - que compara a seqüência de caracteres recebidos do *scanner* com expressões regulares. Assim, a expressão regular identifica o tipo de dado que a seqüência de caracteres representa.

Para RANGEL (1999), a implementação de reconhecedores de linguagens regulares pela técnica de autômatos finitos é mais simples e mais eficiente do que a implementação de reconhecedores de linguagens livres de contexto utilizando autômatos de pilha. Como é possível usar expressões regulares para descrever a estrutura de componentes básicos das linguagens de programação, tais como identificadores, palavras reservadas, literais numéricos,

operadores e delimitadores, a análise léxica é implementada separadamente pela simulação de autômatos finitos.

Para a linha de comando `montante := deposito + taxa_juros * 60` o analisador léxico encontraria os seguintes *tokens*:

Tabela 1.1 - *Tokens* encontrados pelo analisador léxico.

Tipo do token	Valor do token
Identificador	montante
Símbolo de atribuição	:=
Identificador	deposito
Operador de adição	+
Identificador	taxa_juros
Operador de multiplicação	*
Número	60

1.2.2 Expressões regulares

Expressões regulares são modelos matemáticos que definem quais caracteres são aceitos em um conjunto, qual critério de repetição e o posicionamento dos caracteres dentro do conjunto.

Uma expressão regular, segundo GRUNE et al (2001), é uma fórmula que descreve um conjunto de *strings* possivelmente infinito. Como uma gramática, ela pode ser vista tanto como uma receita para gerar esses *strings* quanto como um padrão para combiná-los.

Existem diversas operações que podem ser aplicadas a expressões regulares, mas conforme AHO et al (1995), para a análise léxica são utilizadas as expressões de união, concatenação e fechamento, conforme Tabela 1.2.

Tabela 1.2 - Definições de operações em linguagens (AHO et al, 1995, p.43).

Operação	Definição
união de L e M, escrita $L \cup M$	$L \cup M = \{s \mid s$ está em L ou s está em $M\}$
concatenação de L e M, escrita LM	$LM = \{st \mid s$ está em L e st está em $M\}$
fechamento de Kleene de L, escrito L*	$L^* = \bigcup_{i=0}^{\infty} L^i$ L^* denota “zero ou mais concatenações de” L.
Fechamento positivo de L, escrito L^+	$L^+ = \bigcup_{i=1}^{\infty} L^i$ L^* denota “uma ou mais concatenações de” L.

Dessa forma é possível verificar se um conjunto de caracteres lidos pelo *scanner* pode ser gerado por determinada expressão regular ou não, e classificar tal conjunto de caracteres com sendo um identificador, operador, símbolo de atribuição, número etc.

1.2.3 Diagramas de transições

O diagrama de transições é a representação gráfica (Figura 1.3) de uma máquina de estados. As posições são representadas por círculos e são chamadas de estados. “Os estados são conectados por setas chamados de *lados*. Os lados que deixam o estado *s* possuem rótulos indicando os caracteres de entrada que podem aparecer após o diagrama de transições ter atingido o estado *s*. O rótulo *outro* se refere a qualquer caractere que não seja indicado por qualquer um dos lados que deixam *s*”, dizem AHO et al (1995, p.45).

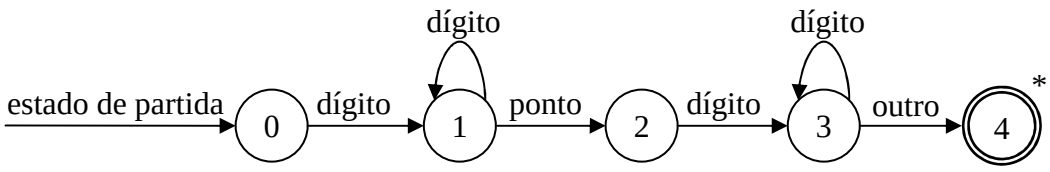

Figura 1.3 - Diagrama de transição para número (AHO et al, 1995, p.47).

1.2.4 Autômatos finitos

Autômato finito é o modelo matemático de uma máquina de estados finitos. Pode ser representado por um diagrama de estados conforme Figura 1.3, onde se definem os estados, transições e ações.

Um autômato finito pode ser determinístico ou não determinístico, onde “não determinístico” significa que mais de uma transição para fora de um estado pode ser possível para um mesmo símbolo de entrada, afirmam AHO et al (1995, p.51).

1.2.4.1 Autômatos finitos não-determinísticos

Um *autômato finito não-determinístico* (AFN) é um modelo matemático que, segundo AHO et al (1995), consiste em:

um conjunto de estados *s*

um conjunto de símbolos de entrada Σ

uma função de transição, movimento, que mapeia pares *estado-símbolo* em conjunto de estados

um estado s_0 que é distinguido como o *estado de partida*

um conjunto de estados *F* distinguidos como *estados de aceitação*

A particularidade do AFN é que o mesmo caractere pode rotular duas ou mais transições para fora de um mesmo estado e os lados podem ser rotulados pelo símbolo especial $\in$ bem como pelos símbolos de entrada. O símbolo $\in$ representa cadeias de caracteres vazias.

O diagrama da Figura 1.4 mostra um AFN que aceita, por exemplo, as cadeias de entrada *abb*, *aabb*, *babb*, *aaabb*. Esse autômato aceita qualquer combinação das letras *a* e *b*, desde que finalizada pela seqüência *abb*.

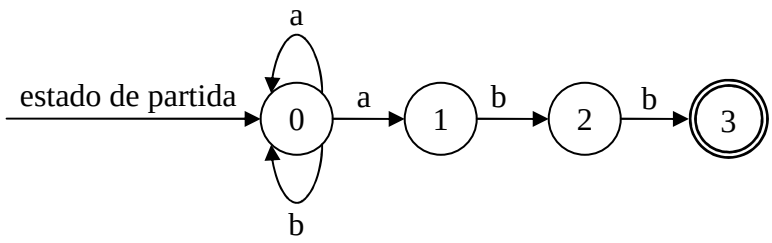

Figura 1.4 - Um autômato finito não-determinístico (AHO et al, 1995, p.52).

1.2.4.2 Autômatos finitos determinísticos

Um autômato *finito determinístico* (AFD) é um caso especial de autômato finito não-determinístico, no qual:

nenhum estado possui uma transição-$\in$**,** isto é, uma transição à entrada $\in$**,** e;

para cada estado *s* e símbolo de entrada *a* existe no máximo *um* lado rotulado *a* deixando *s*;

"Um autômato finito possui no máximo uma transição, a partir de cada estado, para qualquer símbolo de entrada. Como conseqüência, é muito fácil determinar se um AFD aceita uma cadeia de entrada, dado que existe no máximo um único percurso, rotulado por aquela cadeia, a partir do estado inicial." (AHO et al, 1995, p.53).

A Figura 1.5 mostra a representação de um AFD que corresponde ao AFN da Figura 1.4.

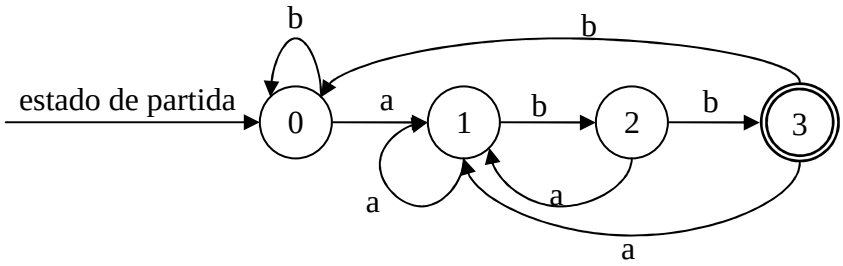

Figura 1.5 - Um autômato finito determinístico (AHO et al, 1995, p.53).

1.2.5 Análise sintática

A análise sintática, ou *parser*, é a segunda etapa de um compilador. O analisador sintático obtém uma cadeia de *tokens* provenientes do analisador léxico e verifica se a mesma pode ser gerada pela gramática da linguagem fonte, segundo AHO et al (1995).

A função do analisador sintático é verificar se as construções usadas no programa estão gramaticalmente corretas. Com a gramática correta, o analisador sintático busca descobrir formas de derivação dessa gramática. (NETO, 1987)

A análise sintática deve reconhecer a estrutura global do programa, por exemplo, verificando que programas, comandos, declarações, expressões etc têm as regras de composição respeitadas.

O analisador sintático, ao interpretar o código

```
if x > 0 then modx := x else modx := (-x)
```

deve identificar que o texto se trata de um `<comando>`, no caso um `<comando-if>`, composto pela palavra reservada `if`, seguida de uma `<expressão>`, seguida da palavra reservada `then` etc. Os itens `<expressão>` e `<atribuição>` ainda podem ser decompostos em fragmentos menores, conforme Figura 1.6. (RANGEL, 1999).

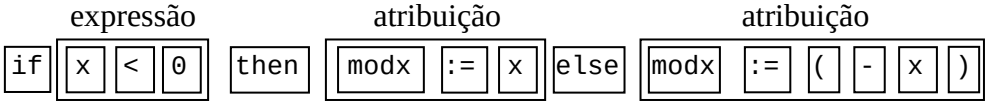

Figura 1.6 - Comando-If (RANGEL, 1999).

Para RANGEL (1999), os métodos de análise sintática podem ser classificados segundo a maneira pela qual a árvore de derivação da cadeia analisada x é construída:

- nos métodos descendentes, a árvore de derivação correspondente a x é construída de cima para baixo, ou seja, da raiz (o símbolo inicial S) para as folhas, onde se encontra x.

- nos métodos ascendentes, a árvore de derivação correspondente a x é construída de baixo para cima, ou seja, das folhas, onde se encontra x, para a raiz, onde se encontra o símbolo inicial S.

1.2.5.1 Análise sintática descendente top-down

A análise sintática *top-down* pode ser vista como uma tentativa de se encontrar uma derivação mais a esquerda para uma cadeia de entrada. Equivalentemente, pode ser vista como uma tentativa de se construir uma árvore gramatical, para a cadeia de entrada, a partir da raiz, criando os nós da árvore gramatical em pré-ordem. (AHO et al, 1995).

Para GRUNE et al (2001), um analisador sintático *top-down* começa construindo o nó superior da árvore, e segue construindo os demais nós da árvore sintática em pré-ordem, o que significa que a parte superior de uma árvore é construída antes de qualquer um de seus nós inferiores. A Figura 1.7 ilustra a seqüência de construção dos nós em uma análise sintática descendente.

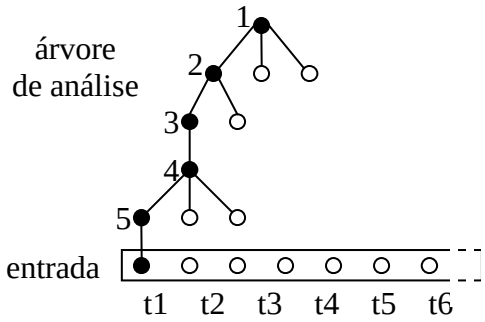

Figura 1.7 - Um analisador *top-down* (GRUNE et al, 2001, p.104).

1.2.5.2 Análise sintática ascendente bottom-up

O método de análise *bottom-up* constrói os nós da árvore sintática em pós-ordem. GRUNE et al (2001) afirmam que a parte superior de uma subárvore é construída após todos os seus nós inferiores terem sido construídos. Quando um analisador *bottom-up* constrói um

nó, todos os seus filhos já foram construídos, estão presentes e são conhecidos. O rótulo do próprio nó também é conhecido. Em seguida o analisador cria o nó, o identifica e o conecta a seus filhos. A Figura 1.8 ilustra a seqüência de construção dos nós em uma análise sintática ascendente.

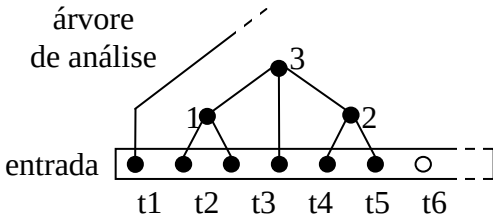

Figura 1.8 - Um analisador *bottom-up* (GRUNE et al, 2001, p.105).

1.2.6 Análise semântica

Análise semântica é a terceira fase da compilação onde se verificam os erros semânticos no código-fonte, por exemplo, uma multiplicação entre tipos de dados diferentes. A análise semântica também coleta as informações necessárias para a fase seguinte da compilação, que é a geração de código-fonte.

Fundamentalmente, a análise semântica trata os aspectos sensíveis ao contexto da sintaxe das linguagens de programação. RANGEL (1999) dá como exemplo que não é possível representar em uma gramática livre de contexto uma regra do tipo: "Todo identificador deve ser declarado antes de ser usado.". A verificação de que essa regra foi aplicada cabe à análise semântica.

> "... o tratamento de contexto se preocupa com relações de longa distância. Por exemplo, ele relaciona o tipo de variável em uma instrução a seu uso em uma expressão, e relaciona a posição de um rótulo a seu uso em uma instrução *goto*. Os conectores nessas relações de longo alcance são os identificadores. Todas as ocorrências aplicadas de um identificador *i* em uma expressão são plugues que se encaixam em um soquete - a declaração para *i* - e desse soquete eles obtêm informações sobre o tipo, a duração e assim por diante." (GRUNE et al, 2001, p.400).

Além da verificação de tipos de dados, o analisador semântico é responsável por perceber as falhas em fluxos de controle e unicidade de comandos.

Uma possibilidade de falha de tipos de dados ocorre quando membros de uma expressão possuem tipos de dados incompatíveis, por exemplo, uma operação de soma não pode conter um elemento de texto, apenas elementos numéricos.

> "Um importante componente da análise semântica é a verificação de tipos. Nela, o compilador checa se cada operador recebe os operandos que são permitidos pela especificação da linguagem fonte. Por exemplo, muitas definições nas linguagens de programação requerem que o compilador relate um erro a cada vez que um número real seja usado para indexar um *array*. No entanto, a especificação da linguagem pode permitir algumas coerções de operandos, como, por exemplo, quando um operador aritmético binário é aplicado a um inteiro e a um real. Nesse caso, o compilador pode precisar converter o inteiro para real". (AHO et al, 1995, p.4).

Uma falha em fluxo de controle ocorre quando uma instrução de quebra de seqüência de comandos é declarada fora de um comando de repetição, conforme Figura 1.9. Por exemplo, na linguagem Pascal o comando de quebra de execução *break* obrigatoriamente deve ser usado dentro de um bloco de repetição, como o *for* ou *while*. Ver Figura 1.10.

```
for i := 2 to 10 do
  x := x * i;

if x > 1000 then
  break              // uso incorreto da instrução break
else
  print(x);
```

Figura 1.9 - Uso incorreto do comando *break.*

```
for i := 2 to 10 do
  begin
    x := x * i;
    if x > 1000 then
      break          // uso correto da instrução break
    else
      print(x);
  end;
```

Figura 1.10 - Uso correto do comando *break.*

Uma falha em unicidade de comandos pode ocorrer quando um elemento somente pode ser declarado uma única vez em. Um bom exemplo de falha de unicidade ocorre quando dentro de uma instrução *case* na linguagem Pascal é declarado mais de uma vez o mesmo rótulo, conforme Figura 1.11.

```
case i of
  1: ExecutaTarefaUm;
  1: ExecutaTarefaUm;   // falha de unicidade de comandos
  2: ExecutaTarefaDois;
  3: ExecutaTarefaTres;
else
  ExecutaTarefaExcecao;
end;
```

Figura 1.11 - Falha em unicidade de comandos.

2 SITUAÇÃO ATUAL DOS LABORATÓRIOS DE ANÁLISE

Um laboratório de análises ambientais realiza basicamente análises em amostras de águas, efluentes, solos e emissões atmosféricas. As amostras analisadas representam muitas vezes um dejeto resultante de um processo industrial a ser tratado antes de ser liberado para a natureza. Tais análises têm por objetivo verificar se os parâmetros analisados estão de acordo com as muitas legislações em vigor. Esse estudo analítico auxilia na escolha do tratamento a ser aplicado no dejeto para adequação às normas ambientais, define se o resíduo pode ser incinerado ou não e ainda pode classificar o resíduo quanto à forma de armazenamento em um depósito de lixo contaminado.

2.1 Cenário dos laboratórios

Georgio Raphaelli e Márcio Gomes são sócios da empresa Labsoft, especializada em automação de laboratórios de controle de qualidade, com atuação em todo o território nacional há mais de doze anos.

Estudos realizados por eles em diversos laboratórios de ensaios e controle de qualidade nas regiões Sul e Sudeste do Brasil mostram que os administradores e gerentes desses laboratórios desconhecem ou ignoram um excelente recurso que já possuem a disposição, capaz de aumentar a qualidade e confiabilidade das análises realizadas, reduzir drasticamente o tempo de permanência de uma amostra dentro do laboratório e aumentar a capacidade produtiva sem a necessidade de aquisição de novos equipamentos ou contratação de funcionários. Esse recurso é a comunicação de dados entre equipamentos analíticos e o sistema de gerenciamento de informações do laboratório.

Quase que a totalidade dos laboratórios de ensaios e controle de qualidade possui equipamentos analíticos digitais, com geração de relatórios em arquivos texto ou exportação

de dados eletrônicos. Porém, esse importantíssimo recurso é simplesmente ignorado pela maioria das empresas, fazendo com que muitas horas de trabalho sejam literalmente desperdiçadas em transcrição, digitação e conferência de resultados.

Através de estudos e trabalhos realizados durante dez anos em dezenas de laboratórios em diversos estados brasileiros, chegou-se aos seguintes dados:

O trabalho administrativo do laboratório gasto em registro de informações para rastreabilidade, transcrição, conferência de dados e elaboração de relatórios, dependendo do tipo de laboratório e matrizes trabalhadas, toma até 60% do tempo total dos analistas;

Tipicamente o volume de dados e informações que saem do laboratório para outras áreas ou para os clientes em forma de resultados finais, é de no máximo 10% do volume total de dados que são coletados, tratados e registrados dentro de um laboratório;

Com esse levantamento, consegue-se ter uma boa visão da quantidade de dados brutos que são gerados durante uma etapa analítica e quantos desses dados são realmente utilizados no final do processo.

Todos esses dados são necessários por questões de rastreabilidade e não podem ser eliminados, porém é possível melhorar os processos através da informatização e coleta e dados para reduzir a intervenção humana, e conseqüentemente liberar o tempo dos analistas para trabalhos mais nobres ou que exijam mais qualificação.

2.2 Realização de análises

Para a realização das análises nas amostras coletadas, são utilizadas diversas técnicas e equipamentos distintos. Cada tipo de amostra ou matriz (água, solo, ar) necessita de uma metodologia de trabalho diferente e equipamentos analíticos diferentes.

Entre os equipamentos analíticos de um laboratório existem desde vidrarias (balão volumétrico, copo de Becker, pipeta volumétrica, bureta etc) até equipamentos eletrônicos muito sofisticados, como espectrômetros, cromatógrafos e fotômetros.

As análises feitas através de vidrarias são totalmente manuais e a leitura dos resultados é feita quase sempre por método visual. Por exemplo, visualiza-se o volume de um líquido em uma escala ou se lê a temperatura de um fluido em um termômetro analógico de

mercúrio. Depois se anota as medições em uma ficha de análise para posterior uso em cálculos e digitação em uma planilha eletrônica ou editor de textos.

Já os equipamentos mais modernos possuem exportação de dados em formato texto ou comunicação serial ou paralela. Nesses casos, a captura dos dados em meio eletrônico é possível, e a interpretação e extração de informações desses dados brutos dependem apenas de se programar rotinas capazes de executarem tal tarefa.

Porém, alguns equipamentos eletrônicos são muito antigos e não possuem interfaces para exportação de dados. Outros possuem tal interface, mas utilizam conectores fora do padrão. Também existem casos em que os dados exportados estão dispostos em formatos desorganizados ou binários, complicando muito sua extração. Raros são os equipamentos que possuem uma interface de exportação de dados padronizada e documentada, o que facilitaria muito a leitura das medições por um sistema externo.

2.3 Transcrição de dados

Uma das tarefas mais importantes que ocorrem em um laboratório é a transcrição dos dados obtidos durante as análises. Para laboratórios não informatizados ou parcialmente informatizados, o procedimento de realização da análise em uma amostra é sempre acompanhado da ficha de análise, onde o analista registra manualmente com caneta as medições que vai realizando.

Após o registro manual das medições, a ficha de análise é digitada em uma planilha eletrônica ou em muitos casos, são feitos cálculos manuais para obtenção de resultados de medições indiretas antes da digitação.

Por questões de tempo e custo, esse processo de digitação e cálculo nem sempre é feito pela própria pessoa que realizou a análise. Considerando-se a possibilidade de erro humano na leitura e interpretação dos valores escritos manualmente na ficha de coleta e a possibilidade de erro de digitação, o processo de transcrição de resultados além de lento é muito susceptível a falhas.

Se pensarmos que um dado pode ter sido interpretado ou digitado de maneira errada por uma pessoa em um sistema informatizado, o trabalho do analista foi em vão, pois o

resultado final da análise realizada que consta no relatório final da amostra não condiz com a realidade, por uma simples falha humana.

Como exemplo de tempo gasto em transcrição de resultados por mês para apenas um equipamento, pode-se usar a transcrição manual de resultados de análises de um equipamento de Raio X com média de 50 amostras por dia e avaliação típica de 20 metais por amostra.

Tipicamente toma-se de 3 a 4 segundos para a leitura visual e digitação de cada um dos valores da ficha de coleta, com cerca de 4 algarismos cada. Contabilizando 50 amostras, 20 metais e 3,5 segundos por digitação, obtém-se 58 minutos por dia ou cerca de 21 horas por mês apenas com o trabalho de digitação.

Somando a isso o trabalho prévio de anotação das medições na ficha de análise e a revisão dos dados digitados, chega-se seguramente ao dobro do tempo, ou seja, 42 horas por mês em um trabalho que pode ser substituído com grandes vantagens pela automação na coleta de dados.

2.4 Exportação de dados

Os equipamentos analíticos possuem basicamente dois tipos de interfaceamento: exportação de dados através de porta de comunicação (serial ou paralela) ou geração de arquivo texto com caracteres ASCII. No caso da comunicação de dados, geralmente os equipamentos trabalham com um protocolo unidirecional através de uma porta serial no padrão RS232C (TORRES, 2001). Alguns equipamentos possuem porta paralela. Nesse caso, a finalidade da porta é transmitir dados para uma impressora, mas é possível capturar os dados e interpretá-los. É raro existir um protocolo de comunicação bidirecional onde o sistema externo precisa trocar dados com o equipamento analítico.

No caso de geração de arquivo texto, o equipamento analítico já possui um PC com um *software* de controle (chamado de *Workstation*), e esse *software* se encarrega de controlar a realização das análises e gerar o arquivo texto em disco. É comum nos equipamentos atuais esse PC ser uma máquina externa, padrão de mercado, conectada ao equipamento via um cabo específico e uma placa ligada a um barramento interno desse PC. Nesse caso fica fácil instalar uma placa de rede e possibilitar o acesso aos dados remotamente. Porém em equipamentos mais antigos, o PC pode ser interno e fazer parte da máquina, podendo até possuir *hardware*

específico do fabricante. É muito comum nesses casos não existir interface de rede, o que complica a coleta de dados.

Para os casos em que os equipamentos geram relatórios em arquivos texto, um *software* pode ser desenvolvido para monitorar uma determinada pasta ou subpastas a procura desses arquivos gerados. Ao encontrar um arquivo, o *software* pode então processá-lo e extrair as informações de interesse para a continuidade do trabalho analítico.

Já para equipamentos que transmitem os resultados através de comunicação de dados serial ou paralela, o *software* deve abrir um canal de comunicação e ficar na escuta da porta, guardando em um *buffer* os dados que chegam para posterior processamento e extração de informações.

A aquisição dos dados diretamente dos equipamentos para um banco de dados centralizado agiliza todo o processo analítico, pois é possível reduzir significativamente o tempo de leitura de resultados, anula o risco de erro humano na transcrição de dados, centraliza em um único local os dados de análises realizadas além de padronizar a forma com que os mesmos são armazenados. Isto permite que seja mantido o histórico de análises realizadas durante muitos anos por questões de rastreabilidade e permite que diferentes sistemas busquem nessa fonte de dados centralizada as informações que precisarem.

2.5 Proposta de solução

> "As técnicas de construção de compiladores podem ser e são aplicadas fora da construção de compiladores em seu sentido mais estrito. Como uma alternativa, outras formas de programação podem ser consideradas construção de compiladores, mais do que se consideraria tradicionalmente. Os exemplos são a leitura de dados estruturados, a introdução rápida de novos formatos e os problemas gerais de conversão de arquivos."
>
> "Se os dados têm uma estrutura clara, em geral é possível escrever uma gramática para eles. Utilizando-se um gerador de analisador, pode-se então gerar automaticamente um analisador. Por exemplo, tais técnicas podem ser aplicadas para criar rapidamente rotinas 'de leitura' para arquivos HTML, arquivos Postscript etc." (GRUNE et al, 2001, p.7-8).
>
> "Uma pequena linguagem é caracterizada pelos seguintes aspectos: pequeno numero de sentenças, propósito específico e não geral e a facilidade de uso" (KAPLAN, 1996).

A solução proposta é desenvolver uma pequena linguagem para um compilador capaz de reconhecer padrões gramaticais em dados brutos gerados por equipamentos analíticos e extrair as informações das medições realizadas.

Como cada conjunto de dados brutos vindo de equipamentos distintos provavelmente terá estrutura gramatical única, a solução prevê um compilador flexível e ajustável para reconhecer as diversas estruturas de dados e conseguir realizar sua interpretação e extração.

Após a extração, os dados serão gravados em banco de dados para uso futuro, como consultas, cálculos, comparativos com legislações etc. A estrutura da tabela de banco de dados utilizada para armazenar as informações extraídas dos dados brutos pode ser vista na Tabela 2.1.

Tabela 2.1 - Estrutura da tabela para armazenamento de dados.

Nome do campo	Tipo de dado	Descrição
AMOSTRA	Texto (50)	Identificação da amostra analisada
IDMEDICAO	Texto (100)	Identificação da medição
VLMEDICAO	Texto (250)	Valor da medição
UNMEDICAO	Texto (50)	Unidade da medição
DILMEDICAO	Fracionário	Diluição da amostra
DTMEDICAO	Data e Hora	Data e hora de realização da análise
RESPONSÁVEL	Texto (50)	Identificação do responsável pela análise
EQUIPAMENTO	Texto (100)	Equipamento utilizado na análise

3 LEVANTAMENTO DE DADOS BRUTOS

Devido a enorme variedade de equipamentos analíticos e relatórios de dados em formatos diversos, torna-se necessário escolher um modelo de relatório para dar continuidade a este trabalho. Com esse relatório definido, se iniciam os estudos das estruturas de dados e padrões, para que sejam definidas as regras gramaticais que o compilador utilizará na extração das informações.

Um tipo de equipamento muito comum em laboratórios de análises ambientais é o espectrômetro gasoso (GCMS). Ele é utilizado para detectar concentrações de certas moléculas em uma amostra. O relatório final que um espectrômetro gera ao analisar uma amostra consiste basicamente em um cabeçalho com a identificação da amostra analisada, data e hora da análise, fator de diluição da amostra e identificação do arquivo de dados. Após o cabeçalho segue uma listagem com os nomes dos compostos encontrados e suas respectivas concentrações, seguidas da unidade de medição utilizada.

3.1 Escolha de um modelo de relatório

Para o desenvolvimento deste projeto, será utilizado como modelo o relatório de dados brutos de um espectrômetro gasoso GCMS que utiliza o *software* Chromquest da empresa Thermo. Para exportação dos dados é utilizado o *software* XCalibur. Os dados brutos de um espectrômetro gasoso GCMS exportados pelo XCalibur estão listados na Figura 3.1.

O conjunto de dados contidos no relatório pode variar muito. No cabeçalho, além dos dados citados anteriormente, podem ainda constar nome do método utilizado para leitura dos compostos, nome do operador do equipamento, identificação do equipamento utilizado e até informações diversas configuráveis pelo próprio operador. Na listagem dos compostos encontrados na amostra pode conter o número seqüencial do composto encontrado, o tempo

de retenção do composto dentro do equipamento, o tempo de retenção esperado e uma infinidade de informações muito específicas da área química analítica.

Além da vasta combinação de dados possíveis nos relatórios de um equipamento, ainda existe a variação de formatos de relatórios por versões diferentes dos *softwares* de controle dos equipamentos analíticos e ainda mais diferenças entre equipamentos de diferentes fabricantes.

```
Data File:              MS124371
Original Data Path:     MS124371.RAW
Sample Name:            22548
Acquisition Date:       01/18/07 16:50:19
Dilution Factor:        1.00
Instrument Method:      C:\Xcalibur\methods\pah_sim.meth

Name
Calculated Amount
Units
Acenafteno
5.282
PPB
2-Fluorbifenil
437.365
PPB
Fluoreno
N/A
PPB
Acenaftileno
0.925
PPB
Criseno
N/A
PPB
Fluoranteno
0.935
PPB
D10-Acenafteno
N/A
ppb
Naftaleno
26.073
PPB
D14-Terfenil
900.374
PPB
D12-Perileno
N/A
ppb
Benzo(k)fluoranteno
N/A
PPB
```

Figura 3.1 - Dados brutos de um espectrômetro gasoso GCMS.

3.2 Identificação dos dados

Nesse exemplo da Figura 3.1 é possível distinguir o cabeçalho do relatório entre os textos `Data File` e `Instrument Method` e o conjunto de dados analisados a partir do texto `Name`.

Dessa forma, o compilador a ser desenvolvido para extração dos dados deve identificar cada um dos textos `Data File`, `Sample Name`, `Acquisition Date`, `Dilution Factor` e `Instrument Method`, e extrair os dados referentes a esses identificadores. Também tem que ser capaz de localizar o bloco com a seqüência de compostos analisados, para extração do nome do composto, concentração e unidade de medição.

3.3 Problemas na identificação de dados e padrões

Conforme citado anteriormente, os relatórios de dados gerados pelos equipamentos não necessariamente são idênticos em formato. Dependendo do equipamento, versão do *software* de controle ou configuração utilizada, os formatos dos dados podem ser diferentes para equipamentos e métodos analíticos iguais.

A Tabela 3.1 lista e relaciona informações iguais, mas com identificadores e formatos distintos entre um cabeçalho de dados do equipamento GCMS 01 (Figura 3.2) e do equipamento GCMS 02 (Figura 3.3).

Tabela 3.1 - Diferenças de padronização entre dados brutos

Informação	Identificação GCMS 01	Identificação GCMS 02
Amostra	Sample Name	Sample ID
Data da Análise	Acquisition Date (mm/dd/aaaa hh:mm:ss)	Acquisition Date (m/d/aaaa hh:mm AM/PM)
Fator de Diluição	Dilution Factor	---

```
Data File:              MS124371
Original Data Path:     MS124371.RAW
Sample Name:            22548
Acquisition Date:       01/18/07 16:50:19
Dilution Factor:        1.00
Instrument Method:      C:\Xcalibur\methods\pah_sim.meth
```

Figura 3.2 - Cabeçalho do equipamento GCMS 01

```
Sample ID:        15357
Instrument ID:    Saturn 2100D
Acquisition Date: 5/11/2005 4:32 AM
Data File:        c:\varianws\data\maio\100505-noite\15357.sms
Calculation Date: 5/24/2005 2:40 PM
Method:           C:\VarianWS\Metodo\voc-rapido.mth
```

Figura 3.3 - Cabeçalho do equipamento GCMS 02

Por isso, deve-se definir uma gramática flexível e projetar um compilador capaz de perceber tais diferenças e tratá-las de forma a extrair os dados corretamente.

3.3.1 Informações não constantes

Algumas informações não são essenciais para realização da análise e podem opcionalmente constar no relatório exportado. Em muitos dos casos, essas informações são apenas informativas ou observações escritas pelo analista no momento da execução da análise. Dessa forma, informações que não aparecem em todos os relatórios gerados devem ser previstas como opcionais nas regras sintáticas.

A Figura 3.4 mostra um cabeçalho completo de um relatório de dados brutos de um espectrômetro GCMS. Já a Figura 3.5 mostra um relatório semelhante do mesmo equipamento, porém sem as informações `Original Data Path` e `Instrument Method`.

```
Data File:             MS124371
Original Data Path:    MS124371.RAW
Sample Name:           22548
Acquisition Date:      01/18/07 16:50:19
Dilution Factor:       1.00
Instrument Method:     C:\Xcalibur\methods\pah_sim.meth
```

Figura 3.4 - Cabeçalho completo de um espectrômetro GCMS.

```
Data File:             FID45754
Sample Name:           37844
Acquisition Date:      01/18/07 16:50:19
Dilution Factor:       1.00
```

Figura 3.5 - Cabeçalho incompleto de um espectrômetro GCMS.

3.3.2 Informações com formatos distintos

Os problemas mais comuns de formatos distintos para uma mesma informação consistem em definição do caractere separador decimal para tipos de dados numéricos e a ordem e quantidade de dígitos para expressar datas e horas.

O problema do caractere separador decimal é relativamente simples de resolver, pois no momento da análise léxica, é possível assumir como separador decimal tanto o ponto quanto a vírgula. Nesse caso, um exemplo de autômato finito que represente um número fracionário com separador decimal ponto ou vírgula é ilustrado na Figura 3.6.

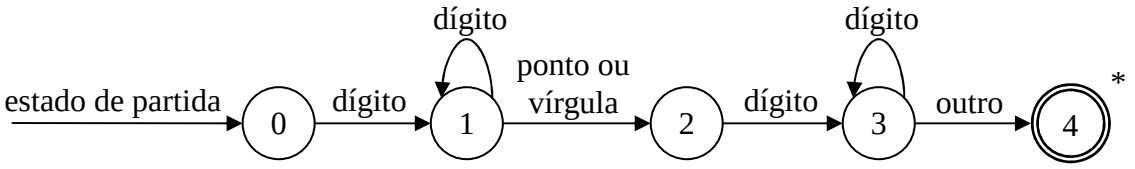

Figura 3.6 - Autômato finito de número com separador decimal ponto ou vírgula.

Já o problema de formatação de data e hora é bem mais complexo, pois esse tipo de dado pode aparecer nos formatos mais distintos. A Tabela 3.2 apresenta o formato ideal para dados que representam data e hora, e a Tabela 3.3 mostra um formato de data e hora que exige uma interpretação distinta e suposição dos dois algarismos de milhar e centena do ano.

Tabela 3.2 - Data no formato dd/mm/aaaa hh:nn:ss.

Ordem	Dado	Formato
1	dia	2 algarismos
2	separador	barra
3	mês	2 algarismos
4	separador	barra
5	ano	4 algarismos
6	separador	espaço
7	hora	2 algarismos
8	separador	dois pontos
9	minuto	2 algarismos
10	separador	dois pontos
11	segundo	2 algarismos

Tabela 3.3 - Data no formato m/d/aa hh:nn:ss.

Ordem	Dado	Formato
1	mês	1 ou 2 algarismos
2	separador	barra

3	dia	1 ou 2 algarismos
4	separador	barra
5	ano	2 algarismos
6	separador	espaço
7	hora	2 algarismos
8	separador	dois pontos
9	minuto	2 algarismos
10	separador	dois pontos
11	segundo	2 algarismos

Além desses formatos ilustrados acima, verificou-se que é possível ocorrer também tipos de dados data e hora com o mês expresso pelas três primeiras letras do nome do mês e hora com 12 horas seguida do texto “AM” ou “PM”, que expressa se é antes ou após o meio dia.

Outro exemplo clássico de dados com formatos distintos ocorre quando uma análise não pode ser realizada ou apenas um elemento da análise não foi analisado por motivos diversos. Nesse caso, é muito comum constar como resultado da análise não um número fracionário, mas o texto “N.A.”, “N/A”, “NA” ou derivações dele, que significam “Não Analisado”. Em outros casos, uma análise não realizada pode simplesmente não retornar valor algum. A Figura 3.7 ilustra um relatório que apresenta elementos que não puderam ser analisados, que são o `Estireno`, o `Bromobenzeno` entre outros.

```
Name
Calculated Amount
trans-1,2-Dicloroeteno
1.740
Estireno
N/A
Diclorodifluorometano
2.444
Bromobenzeno
N/A
Bromoclorometano
6.143
Butilbenzeno
N/A
Ethylbenzene
N/A
Clorometano
2.009
```

Figura 3.7 - Exemplo de relatório com elementos não analisados.

Uma variação dessa característica ocorre quando o valor de concentração encontrado para um elemento é muito baixo, fora do limite mínimo confiável de detecção do equipamento analítico ou muito alto, além da capacidade de detecção.

Analogamente, em uma régua escolar que possui escala milimétrica e tamanho de 30 centímetros, é impossível medir com ela dimensões inferiores a 1 milímetro e superiores a 30 centímetros com precisão. Da mesma forma, um espectrômetro possui um limite inferior e um limite superior para medições. Quando um valor medido é inferior ao limite de detecção, pode ocorrer de o valor ser expresso com o algarismo zero ou os textos “N.D.”, “N/D”, “ND” ou derivações, que significam “Não Detectado”.

Quando a medição é superior ao limite máximo da escala do equipamento, ocorre uma “saturação” do sensor, e o resultado indica uma concentração além do valor máximo suportado pelo equipamento. Nesses casos, é necessário fazer uma diluição da amostra para que a concentração seja reduzida e o equipamento possa detectar a quantidade correta do composto.

Nesses casos em que é necessário fazer a diluição da amostra, o valor direto da concentração encontrado na análise deve ser posteriormente multiplicado pelo fator de diluição aplicado na amostra.

Por exemplo, uma amostra de 10 mL de água de rio apresentou índice de mercúrio de 2,7 mg/L, em um equipamento que possui faixa de escala para análise de mercúrio de 0,01 a 2,0 mg/L. Nesse caso, a amostra foi diluída 10 vezes através da adição de 100 mL de água destilada. A análise foi feita novamente e o valor encontrado de mercúrio foi de 0,261 mg/L. Como esse valor está dentro da escala de medição do equipamento, é considerado um valor válido, porém, como a amostra foi diluída 10 vezes, o valor real da concentração de mercúrio da amostra é 0,261 multiplicado por 10, ou seja, 2,61 mg/L.

3.3.3 *Tokens* semelhantes sensíveis ao contexto

Pode ocorrer de palavras iguais terem significados distintos em um relatório de dados brutos. Uma palavra pode ter um significado quando está no cabeçalho e outro significado quando está no conjunto de dados. A Figura 3.8 mostra que a palavra “`Name`” antecedida pela palavra “`Sample`” no cabeçalho serve para identificar o código da amostra analisada, enquanto

na seqüência, a mesma palavra “Name” tem o significado de iniciar o bloco de resultados dos compostos analisados na amostra.

```
Sample Name:            22548
Acquisition Date:       01/18/07 16:50:19
Dilution Factor:        1.00
Instrument Method:      C:\Xcalibur\methods\pah_sim.meth

Name
Calculated Amount
Units
Acenafteno
5.282
PPB
```

Figura 3.8 - Palavras iguais em contextos diferentes.

3.3.4 Remoção de sujeira

Algumas informações devem ser ignoradas do conjunto de dados brutos, pois são apenas decorativas ou não têm valor como medição analítica. Espaços e linhas em branco devem ser considerados como opcionais na definição das regras gramaticais para não interferirem na extração das informações.

Na Figura 3.9 é possível perceber os textos “Name”, “Calculated Amount” e “Units”. Apenas o texto “Name” é necessário para a definição do início do bloco que contém os resultados da análise realizada. Os demais textos são apenas decorativos e não devem ser extraídos do conjunto de dados brutos.

```
Name
Calculated Amount
Units
Acenafteno
5.282
PPB
2-Fluorbifenil
437.365
PPB
```

Figura 3.9 - Remoção de sujeira dos dados brutos.

4 DESENVOLVIMENTO DO COMPILADOR

4.1 Escolha dos arquivos de dados brutos

Para a definição das regras dos analisadores léxicos e sintáticos, é necessário definir os arquivos de dados brutos a serem utilizados como modelo para extração de dados. Para o desenvolvimento deste projeto, serão utilizados como modelo os relatórios de dados brutos de dois espectrômetros gasosos GCMS.

A escolha de espectrômetros gasosos GCMS deve-se ao fato de que são equipamentos muito utilizados na área analítica ambiental. Os modelos escolhidos são de grandes fabricantes mundiais, presentes na maioria dos laboratórios analíticos.

Um desses equipamentos, aqui denominado GCMS01, utiliza o *software* Chromquest da empresa Thermo, em conjunto com o *software* XCalibur para exportação dos dados. O outro espectrômetro, denominado GCMS02, é fabricado pela empresa Varian, e utiliza para exportação dos dados um software do próprio fabricante, chamado Star Work Station. Um exemplo de arquivo de dados brutos gerado pelo espectrômetro GCMS01 é apresentado na Figura 4.1, enquanto um arquivo de dados brutos gerados pelo espectrômetro GCMS02 é apresentado na Figura 4.2

```
Data File:              MS124371
Original Data Path:     MS124371.RAW
Sample Name:            22548
Acquisition Date:       01/18/07 16:50:19
Dilution Factor:        1.00
Instrument Method:      C:\Xcalibur\methods\pah_sim.meth

Name
Calculated Amount
Units
Acenafteno
5.282
PPB
2-Fluorbifenil
437.365
PPB
Fluoreno
N/A
PPB
Acenaftileno
0.925
PPB
Criseno
N/A
PPB
Fluoranteno
0.935
PPB
D10-Acenafteno
N/A
ppb
Naftaleno
26.073
PPB
D14-Terfenil
900.374
PPB
D12-Perileno
N/A
ppb
Benzo(k)fluoranteno
N/A
PPB
Benzo(b)fluoranteno
N/A
PPB
Pireno
1.691
PPB
Benzo(a)pireno
1.898
PPB
Fenantreno
2.271
PPB
```

Figura 4.1 - Arquivo de dados brutos do equipamento GCMS01

```
Sample Report (Standard)
---------------------------------------
MS Data File Information
---------------------------------------
Sample ID:  15357
Operator:   Luci Andrietta
Instrument ID:    Saturn 2100D
Last Calibration: 5/9/2005 7:36 AM
Acquisition Date: 5/11/2005 4:32 AM
Data File:  c:\varianws\data\maio\100505-noite\15357.sms
Calculation Date: 5/24/2005 2:40 PM
Method:     C:\VarianWS\Metodo\voc-rapido.mth
---------------------------------------
Target Compounds
---------------------------------------
Peaks: 39
(#    RT    Compound Name     Res   Quan  Area  Amount)
1   1.726  Cloreto de vinila        Miss.  61.9   0      0.000   ppb
2   1.943  1,1-Dicloroeteno         Miss.  61.0   0      0.000   ppb
3   2.028  Dissulfeto de Carbono    Miss.  76.0   0      0.000   ppb
4   2.121  cis-1,2-Dicloroeteno     Miss.  61.0   0      0.000   ppb
5   2.179  1,1-Dicloroetano         Miss.  63.2   0      0.000   ppb
6   2.333  trans-1,2-Dicloroeteno   Miss.  61.0   0      0.000   ppb
7   2.460  Dibromofluorometano      Id.    113.0  2838   82.398  ppb
8   2.552  1,1,1-Tricloroetano      Miss.  97.2   0      0.000   ppb
9   2.618  1,1-Dicloropropeno       Miss.  75.0   0      0.000   ppb
10  2.677  Tetracloreto de Carbono  Miss.  118.9  0      0.000   ppb
11  2.978  Tricloroeteno            Miss.  132.0  0      0.000   ppb
12  3.107  Bromodiclorometano       Miss.  83.1   0      0.000   ppb
13  3.360  cis-1,3-Dicloropropeno   Miss.  75.0   0      0.000   ppb
14  3.579  Tolueno-d8               Id.    98.2   21696  65.477  ppb
15  3.620  Tolueno                  Id.    91.2   653    0.823   ppb
16  3.757  1,1,2-Tricloroetano      Miss.  97.0   0      0.000   ppb
17  3.846  1,3-Dicloropropano       Miss.  76.0   0      0.000   ppb
18  4.050  Tetracloroeteno          Miss.  166.0  0      0.000   ppb
19  4.538  Clorobenzeno             Miss.  112.1  0      0.000   ppb
20  4.604  1,1,1,2-tetracloroetano  Miss.  132.9  0      0.000   ppb
21  4.668  Etilbenzeno              Miss.  91.1   0      0.000   ppb
22  4.776  m,p-Xilenos              Miss.  91.1   0      0.000   ppb
23  5.043  o-Xileno                 Miss.  91.1   0      0.000   ppb
24  5.101  Tribromometano           Miss.  173.1  0      0.000   ppb
25  5.470  Bromofluorobenzeno       Id.    175.9  4503   54.134  ppb
26  5.545  Bromobenzeno             Miss.  158.0  0      0.000   ppb
27  5.739  Propilbenzeno            Miss.  91.1   0      0.000   ppb
28  5.906  1,2,3-Trimetilbenzeno    Miss.  105.1  0      0.000   ppb
29  6.204  1,2,4-Trimetilbenzeno    Miss.  105.1  0      0.000   ppb
30  6.882  Butilbenzeno             Miss.  91.2   0      0.000   ppb
31  8.261  1,3,5-Triclorobenzeno    Miss.  182.0  0      0.000   ppb
32  8.394  Naftaleno                Miss.  128.2  0      0.000   ppb
---------------------------------------
Unidentified Peaks
---------------------------------------
None
```

Figura 4.2 - Arquivo de dados brutos do equipamento GCMS02

4.2 Estrutura do compilador

Mesmo tendo sido projetadas duas classes, uma para cada gramática, o compilador desenvolvido utiliza uma estrutura procedural, e não orientada a objetos. Para o propósito de um compilador que utiliza uma pequena linguagem para extração de dados, a implementação em uma estrutura orientada a objetos adicionaria nesse momento uma complexidade desnecessária para a solução do problema.

4.3 Gramática

A gramática proposta para reconhecer o arquivo de dados brutos do equipamento GCMS01 é apresentada na Figura 4.3.

```
<amostra>        ::= <cab_amostra> <cab_parametros> <parametros>
<cab_amostra>    ::= <dados> <id_amostra> <data_analise> <fator_diluicao> <dados>
<dados>          ::= <dado> <dados>
                 |   <dado>
<dado>           ::= <id>
                 |   <arquivo>
                 |   '.'
                 |   ':'
<id_amostra>     ::= 'Sample' 'Name' ':' <id_am>
<data_analise>   ::= 'Acquisition' 'Date' ':' <data> <hora>
<fator_diluicao> ::= 'Dilution' 'Factor' ':' <numerico>
<cab_parametros> ::= 'Name' 'Calculated' 'Amount' 'Units'
<parametros>     ::= <parametro> <parametros>
                 |   <parametro>
<parametro>      ::= <nome_parametro> <valor_medido> <unidade>
<nome_parametro> ::= <id> <id>
                 |   <id>
<valor_medido>   ::= <numerico>
                 |   'N/A'
<unidade>        ::= <id>
<numerico>       ::= <inteiro>
                 |   <fracionario>
<id_am>          ::= <id>
                 |   <inteiro>
<arquivo>        ::= [A-Z]:('\'[A-Z]+)+[A-Z,a-z,_][A-Z,a-z,_,0-9]*'.'[A-Z,a-z,_,0-9]*
<inteiro>        ::= [0-9]+
<fracionario>    ::= [0-9]+'.'[0-9]+
<id>             ::= [A-Z,a-z,_][A-Z,a-z,_,0-9]*
<data>           ::= [0-9][0-9]'/'[0-9][0-9]'/'[0-9][0-9]
<hora>           ::= [0-9][0-9]':'[0-9][0-9]':'[0-9][0-9]
```

Figura 4.3 - Gramática do arquivo de dados brutos do equipamento GCMS01

Os símbolos não terminais dessa gramática estão representados entre os sinais ‘<’ e ‘>’. Cada um desses símbolos não terminais representa uma regra gramatical.

Os símbolos representados entre aspas simples e as expressões regulares são símbolos terminais, reconhecidos como *tokens* da linguagem.

A gramática proposta para reconhecer o arquivo de dados brutos do equipamento GCMS02 é apresentada na Figura 4.4.

```
<amostra>        ::= <cab_amostra> <cab_parametros> <parametros> <tracos>
<cab_amostra>    ::= 'Sample' <dados> <id_amostra> <dados> <data_analise> <dados>
<cab_parametros> ::= 'Peaks' ':' <inteiro> <entre_par>
<parametros>     ::= <parametro> <parametros>
                 |   <parametro>
<parametro>      ::= <inteiro> <fracionario> <nome_parametro> <tipo_res> <quant>
                     <inteiro> <fracionario> <id>
<dados>          ::= <dado> <dados>
                 |   <dado>
<dado>           ::= <id>
                 |   <tracos>
                 |   <arquivo>
                 |   <entre_par>
                 |   'Date'
                 |   'ID'
                 |   '.'
                 |   ':'
<id_amostra>     ::= 'Sample' 'ID' ':' <id_am>
<data_analise>   ::= 'Acquisition' 'Date' ':' <data> <hora> <turno>
<turno>          ::= 'AM'
                 |   'PM'
<nome_parametro> ::= <id> <id>
                 |   <id>
<id_am>          ::= <id>
                 |   <inteiro>
<tipo_res>       ::= 'Id' '.'
                 |   'Miss' '.'
                 |   'Fail'
<quant>          ::= <id>
                 |   <fracionario>
<arquivo>        ::= [A-Z]:('\'[A-Z]+)+[A-Z,a-z,_][A-Z,a-z,_,0-9]*'.'[A-Z,a-z,_,0-9]*
<inteiro>        ::= [0-9]+
<fracionario>    ::= [0-9]+'.'[0-9]+
<tracos>         ::= '-''-'+
<id>             ::= [A-Z,a-z,_][A-Z,a-z,_,0-9]*
<data>           ::= [0-9][0-9]'/'[0-9][0-9]'/'[0-9][0-9]
<hora>           ::= [0-9][0-9]':'[0-9][0-9]':'[0-9][0-9]
```

Figura 4.4 - Gramática do arquivo de dados brutos do equipamento GCMS02

Os símbolos não terminais dessa gramática estão representados entre os sinais ‘<’ e ‘>’. Cada um destes símbolos não terminais representa uma regra gramatical.

Os símbolos representados entre aspas simples e expressões regulares são símbolos terminais, reconhecidos como *tokens* da linguagem.

4.4 Analisador Léxico

O analisador léxico é o elemento do compilador que classifica as palavras ou símbolos do conjunto de dados brutos a ser processado. Ele é uma implementação de autômatos finitos de uma expressão regular. Sua execução consiste em varrer o conjunto de dados, caractere a caractere, e classificar a seqüência de caracteres encontrados de acordo com uma expressão regular. Caso o conjunto de caracteres lido satisfaça a condição de uma determinada expressão regular, tal *token* é classificado com um tipo que o identifica unicamente na gramática.

Para isso, definiu-se uma variável chamada “Token” do tipo *string* para armazenar o último *token* encontrado, e uma variável chamada “TipoToken” do tipo enumerado TTipoToken para armazenar a classificação desse último *token*. Cada elemento de TTipoToken é uma classificação de um *token* encontrado. A Tabela 4.1 lista os *tokens* reconhecidos pelo analisador léxico para a gramática proposta.

Tabela 4.1 - Lista dos *tokens* reconhecidos pelo analisador léxico

Tipo do token	Valor do token
T_ID	[A-Z,a-z,_][A-Z,a-z,_,0-9]*
T_INTEIRO	[0-9]+
T_FRACIONARIO	[0-9]+ '.' [0-9]+
T_DATA	[0-9][0-9] '/' [0-9][0-9] '/' [0-9][0-9]
T_HORA	[0-9][0-9] ':' [0-9][0-9] ':' [0-9][0-9]
T_IGUAL	'='
T_VEZES	'*'
T_MENOS	'-'
T_PONTO	'. '
T_VIRGULA	','
T_DOISPONTOS	': '
T_PONTOVIRGULA	'; '
T_TRACOSIMPLES	'-"-'+
T_TRACODUPLO	'="='+
T_TRACOVEZES	'*"*'+
T_ENTREPAR	'(' .* ')'
T_ABREPAR	'('
T_FECHAPAR	')'
T_ARQUIVO	[A-Z]':'('\'[A-Z]+)+[A-Z,a-z,_][A-Z,a-z,_,0-9]*'.'[A-Z,a-z,_,0-9]*
T_AM	'AM'
T_PM	'PM'
T_PEAKS	'Peaks'
T_SAMPLE	'Sample'
T_NAME	‘Name’
T_ACQUISITION	'Acquisition'

T_DATE	'Date'
T_DILUTION	'Dilution'
T_FACTOR	'Factor'
T_CALCULATED	'Calculated'
T_AMOUNT	'Amount'
T_UNITS	'Units'
T_MISS	'Miss'
T_FAIL	'Fail'
T_TXTID	'ID'
T_NA	'N/A'

A varredura do arquivo de dados brutos e classificação dos *tokens* é feita por uma função chamada PegaProximoToken. Essa função lê o arquivo de dados caractere a caractere, até encontrar uma seqüência de caracteres que satisfaça uma expressão regular. Quando isso ocorre, a função define os valores das variáveis “Token” e “TipoToken” e retorna. Caso a seqüência de caracteres lidos não seja reconhecida por nenhuma expressão regular, a variável “Token” recebe o conjunto de caracteres encontrado e a variável “TipoToken” recebe o valor T_ERROLEX. A Tabela 4.2 ilustra o reconhecimento e classificação de alguns *tokens*.

Tabela 4.2 - Exemplo de reconhecimento de *tokens*

Valor do token	Tipo do token
MS124371	T_ID
MS124371.RAW	T_ID
Instrument	T_ID
Report	T_ID
Calibration	T_ID
Calculation	T_ID
49.0+51.0+84.0	T_ID
1,3,5-Triclorobenzeno	T_ID
Benzo(k)fluoranteno	T_ID
2-Fluorbifenil	T_ID
112233	T_INTEIRO
112.233	T_FRACIONARIO
05/10/07	T_DATA
08:47:16	T_HORA
----------	T_TRACOSIMPLES
=========	T_TRACODUPLO
**********	T_TRACOVEZES
(Standard)	T_ENTREPAR
(# RT Compound Name Res Type Quan Íons Area Amount)	T_ENTREPAR
C:\VarianWS\Metodo\voc-rapido.mth	T_ARQUIVO
C:\star\data\maio\23.05.05\15815.run	T_ARQUIVO
C:\star\tph 2005\tphfpcw772.mth	T_ARQUIVO
'AM'	T_AM
'PM'	T_PM
'Peaks'	T_PEAKS

'Sample'	T_SAMPLE
‘Name’	T_NAME
'Acquisition'	T_ACQUISITION
'Date'	T_DATE
'Dilution'	T_DILUTION
'Factor'	T_FACTOR
'Calculated'	T_CALCULATED
'Amount'	T_AMOUNT
'Units'	T_UNITS
'Miss'	T_MISS
'Fail'	T_FAIL
'ID'	T_TXTID
'N/A'	T_NA

4.5 Analisador Sintático

O analisador sintático processa as regras gramaticais através do consumo de *tokens*, e verifica se esses *tokens* estão na seqüência correta de cada regra. Cada regra gramatical é uma *procedure* do analisador sintático, que consome os termos terminais e chama a *procedure* correspondente para os termos não terminais da gramática.

Para a análise sintática, foram criadas duas classes: TGCMS01 e TGCMS02.

4.5.1 A classe TGCMS01

A classe TGCMS01, ilustrada na Figura 4.5, foi construída para interpretar arquivos de dados brutos do equipamento GCMS01, conforme gramática apresentada anteriormente na Figura 4.3. A execução da análise sintática ocorre pela chamada da *procedure* Sintático de um objeto criado a partir do *constructor* Create dessa classe.

```
TGCMS01 = class
  FSaida: string;
  FNumeroErros: Integer;
private
  procedure Gramatica;
  procedure CabAmostra;
  procedure Dados;
  procedure Dado;
  procedure IdAmostra;
  procedure IdAm;
  procedure DataAnalise;
  procedure FatorDiluicao;
  procedure CabParametros;
  procedure Parametros;
  procedure Parametro;
  procedure NomeParametro;
  procedure ValorMedido;
  procedure Unidade;
  procedure Numerico;
public
  procedure Sintatico;
  constructor Create;
end;
```

Figura 4.5 - Classe TGCMS01

A variável `FNumErros` é um acumulador que guarda a quantidade de erros gramaticais encontrados durante a execução do analisador sintático. A variável `FSaida` é a variável utilizada pelo montador de código para guardar os dados processados, ou seja, o produto final da análise sintática.

4.5.2 A classe TGCMS02

A classe TGCMS02, ilustrada na Figura 4.6, foi construída para interpretar arquivos de dados brutos do equipamento GCMS02, conforme gramática apresentada anteriormente na Figura 4.4. A execução da análise sintática ocorre pela chamada da *procedure* Sintático de um objeto criado a partir do *constructor* Create dessa classe.

```
TGCMS02 = class
  FSaida: string;
  FNumeroErros: Integer;
private
  procedure Gramatica;
  procedure CabAmostra;
  procedure Dados;
  procedure Dado;
  procedure IdAmostra;
  procedure IdAm;
  procedure DataAnalise;
  procedure Turno;
  procedure CabParametros;
  procedure Parametros;
  procedure Parametro;
  procedure NomeParametro;
  procedure TipoResultado;
  procedure Quantificacao;
public
  procedure Sintatico;
  constructor Create;
end;
```

Figura 4.6 - Classe TGCMS02

A variável `FNumErros` é um acumulador que guarda a quantidade de erros gramaticais encontrados durante a execução do analisador sintático. A variável `FSaida` é a variável utilizada pelo montador de código para guardar os dados processados, ou seja, o produto final da análise sintática.

A Figura 4.7 ilustra o funcionamento da *procedure* FatorDiluicao, que processa a regra gramatical `<fator_diluicao> ::= 'Dilution' 'Factor' ':' <numerico>`. Esse código exige e consome os *tokens* 'Dilution', 'Factor' e ':', e depois chama a *procedure* Numérico, que processa a regra `<numerico>`. Caso algum dos *tokens* exigidos pela gramática não sejam encontrados na posição correta, o analisador sintático gera uma mensagem de erro por não ter encontrado o *token* esperado.

```
//-------------------------------------------------------------------//
// Regra esperada para esta procedure é:                             //
// <fator_diluicao> ::= 'Dilution' 'Factor' ':' <numerico>           //
//-------------------------------------------------------------------//

procedure TSintatico01.FatorDiluicao;
const
  NmRegra = '<fator_diluicao>';
begin
  if TipoToken = T_DILUTION then
    begin
      PegaProximoToken;
      if TipoToken = T_FACTOR then
        begin
          PegaProximoToken;
          if TipoToken = T_DOISPONTOS then
            begin
              PegaProximoToken;
              frmPrincipal.mmSaida.Lines.Add('Diluicao' + #9 + Token);
              Numerico;
            end
          else
            frmPrincipal.EscreveMensagemTokenEsperado(':', NmRegra);
        end
      else
        frmPrincipal.EscreveMensagemTokenEsperado('Factor', NmRegra);
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Dilution', NmRegra);
end;
```

Figura 4.7 - Código para interpretação da regra gramatical `<fator_diluicao>`

A *procedure* Numérico, chamada pela *procedure* FatorDiluicao, espera encontrar um *token* de um número inteiro ou fracionário, que são terminais. A Figura 4.8 mostra que caso a *procedure* encontre um desses *tokens*, ela apenas os consome. Do contrário, gera uma mensagem de erro.

```
//-----------------------------------------------//
// Regra esperada para esta procedure é:         //
// <numerico> ::= <inteiro>                      //
//            |   <fracionario>                   //
//-----------------------------------------------//

procedure TSintatico01.Numerico;
const
  NmRegra = '<numerico>';
begin
  if TipoToken in [T_INTEIRO, T_FRACIONARIO] then
    begin
      PegaProximoToken;
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado(
      '<inteiro> ou <fracionario>', NmRegra);
end;
```

Figura 4.8 - Código para interpretação da regra gramatical `<numerico>`

4.6 Montagem

Em um compilador, a etapa de montagem é responsável por gerar o produto final da compilação. No caso de compiladores de linguagens de programação clássicas, o montador gera arquivos em código de máquina para execução pelo processador. No caso específico de extração de dados, o montador gera um conjunto de dados organizados como produto da compilação, para que seja usado em uma etapa seguinte.

Para este projeto, o montador deve gerar um arquivo contendo um cabeçalho e uma lista de compostos encontrados na amostra com suas respectivas concentrações e unidades de medição. O cabeçalho deve conter a identificação da amostra, a data e hora de análise e opcionalmente a diluição utilizada na amostra. Já a lista de compostos analisados deve obrigatoriamente apresentar o nome do composto, a concentração encontrada e a unidade de medição. A Figura 4.9 ilustra o modelo de um arquivo de dados gerado pelo compilador após o processamento dos dados brutos.

```
'Amostra'              <tab>  <id_am>
'Data'                 <tab>  <data>
'Hora'                 <tab>  <hora>
'Diluicao'             <tab>  <numerico>
<nome_parametro_01>    <tab>  <valor_01>  <tab>  <unidade_01>
<nome_parametro_02>    <tab>  <valor_02>  <tab>  <unidade_02>
<nome_parametro_03>    <tab>  <valor_03>  <tab>  <unidade_03>
.
.
<nome_parametro_nn>    <tab>  <valor_nn>  <tab>  <unidade_nn>
```

Figura 4.9 - Modelo de um arquivo de dados gerado pelo compilador

A Figura 4.10 apresenta um trecho do resultado final da extração dos dados de um espectrômetro GCMS01.

```
Amostra          22548
Data             01/18/07
Hora             16:50:19
Diluicao         1.00
ACENAFTENO       5.282      PPB
2-FLUORBIFENIL   437.365    PPB
FLUORENO         N/A        PPB
ACENAFTILENO     0.925      PPB
CRISENO          N/A        PPB
FLUORANTENO      0.935      PPB
```

Figura 4.10 - Exemplo real de extração de dados

4.7 Dificuldades encontradas

As principais dificuldades encontradas no desenvolvimento do protótipo do compilador foram:

- definição de expressões regulares e autômatos finitos para o analisador léxico;

- definição das regras gramaticais para o analisador sintático;

Essas dificuldades ocorrem para os dois equipamentos estudados, devido à falta de estruturas organizadas nos arquivos de dados brutos, já que esses arquivos não são construídos tendo como objetivo a interpretação e extração de dados por um sistema automatizado. Dessa forma, os arquivos não possuem estruturas léxicas e gramaticais “elegantes”, exigindo um esforço bastante grande para interpretação e definição das regras.

4.7.1 Analisador Léxico

Como exemplo de dificuldade na definição de autômatos finitos para classificação dos *tokens*, é possível destacar os seguintes casos:

***Token* "49.0+51.0+84.0":** Esse *token* deve ser classificado como T_ID, e não como T_FRACIONARIO ou T_ERROLEX. A principal dificuldade nesse caso é que o *token* inicia exatamente como um número inteiro, depois aparentemente se apresenta como fracionário e por último é identificado como um identificador. Tem que se ter cuidado para não considerá-lo como uma seqüência de *tokens* T_FRACIONARIO, T_MAIS, T_FRACIONARIO, T_MAIS e T_FRACIONARIO.

***Token* "1,3,5-Triclorobenzeno":** Esse *token* deve ser identificado como T_ID, e não como T_FRACIONARIO ou T_ERROLEX. A principal dificuldade nesse caso é que o *token* inicia exatamente como um número inteiro, depois aparentemente se apresenta como fracionário. Em seguida apresenta outro separador decimal, o que caracterizaria erro léxico. Porém, por se tratar do nome de um composto químico, deve ser classificado como T_ID. Tem que se ter cuidado para não considerá-lo como uma seqüência de *tokens* T_ERROLEX, T_MENOS, e T_ID.

***Token* "c:\star\tph 2005\tphfpcw772.mth":** Esse *token* deve ser identificado como T_ARQUIVO e não como T_ID e T_ID, devido ao caractere de espaço entre "tph" e "2005". Existem três dificuldades nesse caso. A primeira é que o *token* deve obrigatoriamente iniciar com uma letra apenas, um caractere de dois pontos, um caractere de barra invertida e seguir uma seqüência de demais caracteres. A segunda dificuldade consiste em que obrigatoriamente, esse *token* deve terminar com uma seqüência de um ponto e pelo menos um caractere, para representar um nome de arquivo. A terceira dificuldade é que o caractere de espaço não representa nesse caso o final do *token*, como em todos os outros casos.

***Token* "c:\star\data\maio\23.05.05\15815.run":** Além das características descritas no caso anterior, esse *token* apresenta uma particularidade que consiste em o nome de uma pasta possuir ponto. Esse ponto não pode ser considerado o ponto que separa o nome do arquivo de sua extensão, pois isso poderia fazer com que o *scanner* identificasse o final do *token* antes de localizar o verdadeiro nome do arquivo. Se isso ocorresse, o *token* seria classificado como T_ARQUIVO seguido de um T_ID, mas deve ser identificado como T_ARQUIVO;

***Token* "Benzo(k)fluoranteno":** Esse *token* deve ser identificado como T_ID, e não como T_ID, T_ENTREPAR e T_ID. A principal dificuldade nesse caso é que o *token* inicia como um simples identificador, porém apresenta um texto entre parênteses. Isso poderia quebrar a leitura do *scanner*, gerando a classificação T_ID e T_ENTREPAR. Porém, por se tratar do nome de um composto químico, deve ser classificado inteiramente como T_ID. Assim, o *scanner* deve finalizar o lexema somente quando localizar um espaço, e não o texto entre parênteses.

***Token* "2-Fluorbifenil":** Esse *token* deve ser classificado como T_ID, e não como T_INTEIRO ou T_ERROLEX. A principal dificuldade nesse caso é que o *token* inicia exatamente como um número inteiro, depois um caractere de sinal de menos, seguido de um identificador. Ao encontrar o caractere de sinal de menos, o lexema não pode ser considerado um erro léxico. Também tem que se ter cuidado para não considerá-lo como uma seqüência de *tokens* T_INTEIRO, T_MENOS, T_ID ou T_ERROLEX;

Todas as dificuldades citadas acima foram resolvidas e o analisador léxico consegue agora reconhecer perfeitamente todos os *tokens* definidos para a linguagem.

4.7.2 Analisador Sintático

Os arquivos de dados brutos gerados pelos equipamentos analíticos não têm como propósito gerar um conjunto de dados estruturado para extração por um sistema externo automatizado. Dessa forma, as construções sintáticas desses arquivos apresentam formatos muitas vezes complicados de serem interpretados.

Um exemplo de falta de estrutura é o conjunto de dados que formam o cabeçalho do arquivo do equipamento GCMS02, apresentado na Figura 4.11. Nesse conjunto de dados é visível a quantidade de dados desnecessários para a extração, sendo que os dados importantes são apenas "Sample ID" e "Acquisition Date", destacados em negrito. Assim, foi necessário definir a regra sintática `<cab_amostra> ::= 'Sample' <dados> <id_amostra> <dados> <data_analise> <dados>`, onde os termos `<dados>` apenas consomem os *tokens* como se eles fossem comentários.

A dificuladade nessa tarefa consiste em que os *tokens* a serem ignorados não são delimitados por um simples identificador inicial e outro final, como é comum ocorrer em comentários de códigos fonte os delimitadores "/*" e "*/".

```
Sample Report (Standard)
---------------------------------------
MS Data File Information
---------------------------------------
Sample ID:  15357
Operator:   Luci Andrietta
Instrument ID:    Saturn 2100D
Last Calibration: 5/9/2005 7:36 AM
Acquisition Date: 5/11/2005 4:32 AM
Data File:  c:\varianws\data\maio\100505-noite\15357.sms
Calculation Date: 5/24/2005 2:40 PM
Method:     C:\VarianWS\Metodo\voc-rapido.mth
---------------------------------------
Target Compounds
---------------------------------------
```

Figura 4.11 - Cabeçalho do arquivo do equipamento GCMS02

Outro caso de dificuldade na definição de regras sintáticas ocorre no caso de nomes de compostos químicos. Um nome de composto pode ser formado por um único identificador, por exemplo, “Clorofórmio”, “Dibromofluorometano” ou “Tricloroeteno” ou ainda por uma seqüência de identificadores, como “Cloreto de vinila”, “Cloreto de metileno” ou “Dissulfeto de Carbono”. O problema dessa regra sintática é que não existe um *token* que indica o separador entre os identificadores, tampouco um *token* que identifica o final da seqüência.

Assim, a regra sintática `<nome_parametro>` ilustrada na Figura 4.12 consiste em consumir os *tokens* até que seja localizado um *token* diferente de T_ID, e o nome do parâmetro é a seqüência de identificadores encontrados.

```
<nome_parametro> ::= <id> <id>
                   |   <id>
```

Figura 4.12 - Regra sintática `<nome_parametro>`

Essa dificuldade de definição do nome do parâmetro ocorre tanto para o equipamento GCMS01 quanto para o equipamento GCMS02. A Figura 4.13 ilustra o código fonte para resolver essa questão. É possível perceber a recursividade através da chamada da *procedure* NomeParametro pela própria *procedure* NomeParametro, caso o *token* seguinte seja um T_ID.

```
//-----------------------------------------------------//
// Regra esperada para esta procedure é:               //
// <nome_parametro> ::= <id> <id>                      //
//                    |   <id>                         //
//-----------------------------------------------------//

procedure TGCMS01.NomeParametro;
begin
  if TipoToken = T_ID then
    begin
      // Montador
      FSaida := FSaida + ' ' + Token;

      PegaProximoToken;

      // Recursividade
      NomeParametro;
    end;
end;
```

Figura 4.13 - Código para regra sintática <nome_parametro>

Todas as dificuldades citadas acima foram resolvidas e o analisador sintático consegue agora reconhecer perfeitamente todas as regras gramaticais definidas.

4.8 Interface do protótipo do compilador

A interface do protótipo do compilador desenvolvido possui quatro botões e quatro janelas, conforme mostra a Figura 4.14. O botão com título “Abrir” abre uma janela para escolha do arquivo de dados brutos a importar. O botão com nome “Léxico” executa o analisador léxico para os dados brutos carregados anteriormente. Os botões “GCMS01” e “GCMS02” executam os analisadores sintáticos “GCMS01” e “GCMS02” respectivamente.

A janela dos dados brutos é onde são colocados os dados brutos gerados pelo equipamento analítico. A janela dos *tokens* é onde são listados os *tokens* encontrados pelo analisador léxico durante a interpretação dos dados brutos. A janela dos dados compilados é a área onde aparece o resultado final da compilação dos dados brutos, de acordo com a gramática escolhida. Essa janela dispõe de forma limpa e ordenada os dados extraídos. A janela de mensagens de erro mostra os erros sintáticos encontrados durante a interpretação do arquivo de dados brutos, caso ocorram.

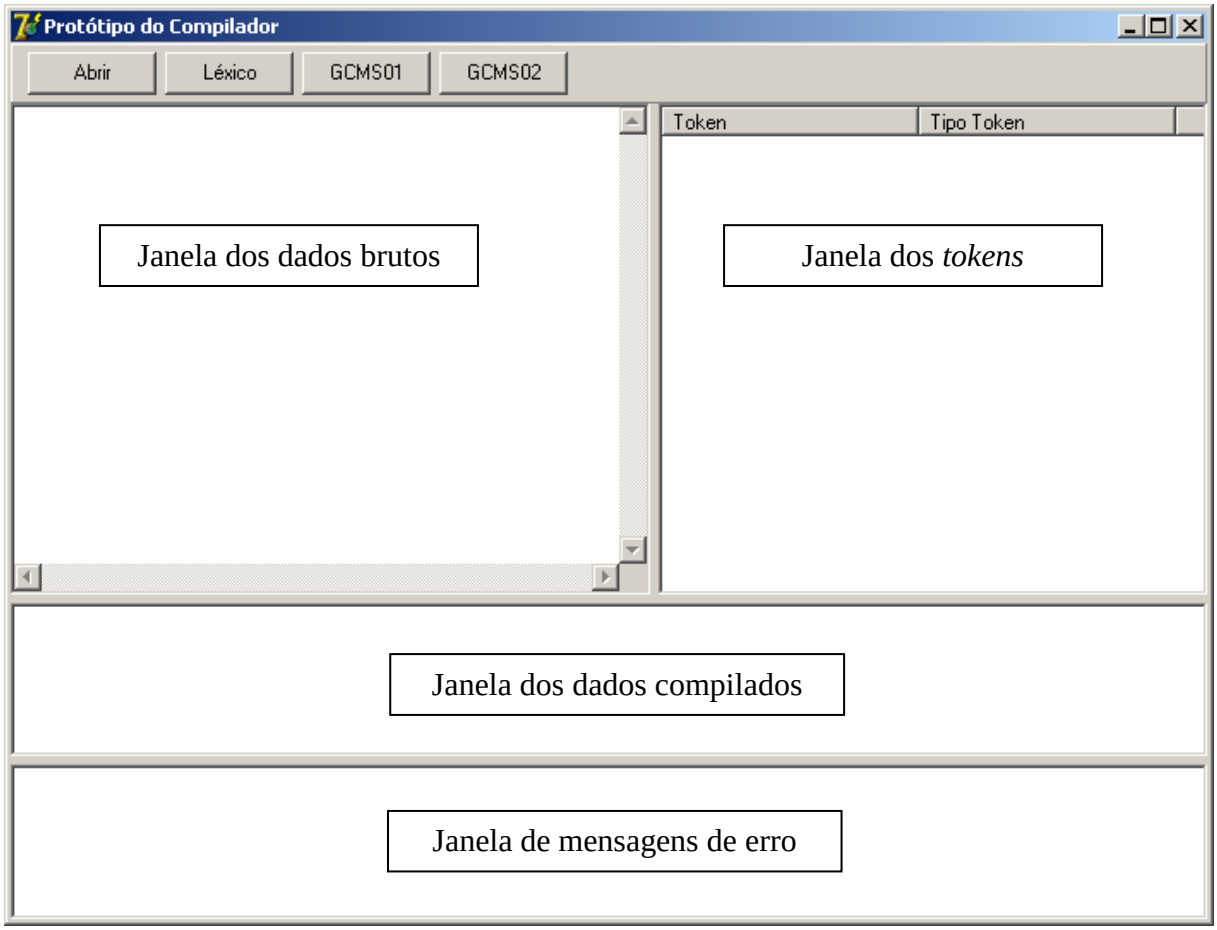

Figura 4.14 - Interface do protótipo do compilador

4.9 Utilização do protótipo do compilador

A utilização do protótipo do compilador consiste em apenas dois comandos:

Importação de um arquivo de dados brutos gerado pelo equipamento analítico, através do botão “Abrir”;

b) Interpretação dos dados brutos através do botão com o nome do respectivo equipamento analítico;

A Figura 4.15 ilustra a interface do compilador após a interpretação dos dados brutos de um equipamento GCMS01, enquanto a Figura 4.16 mostra a tela do compilador após a interpretação dos dados brutos de um equipamento GCMS02.

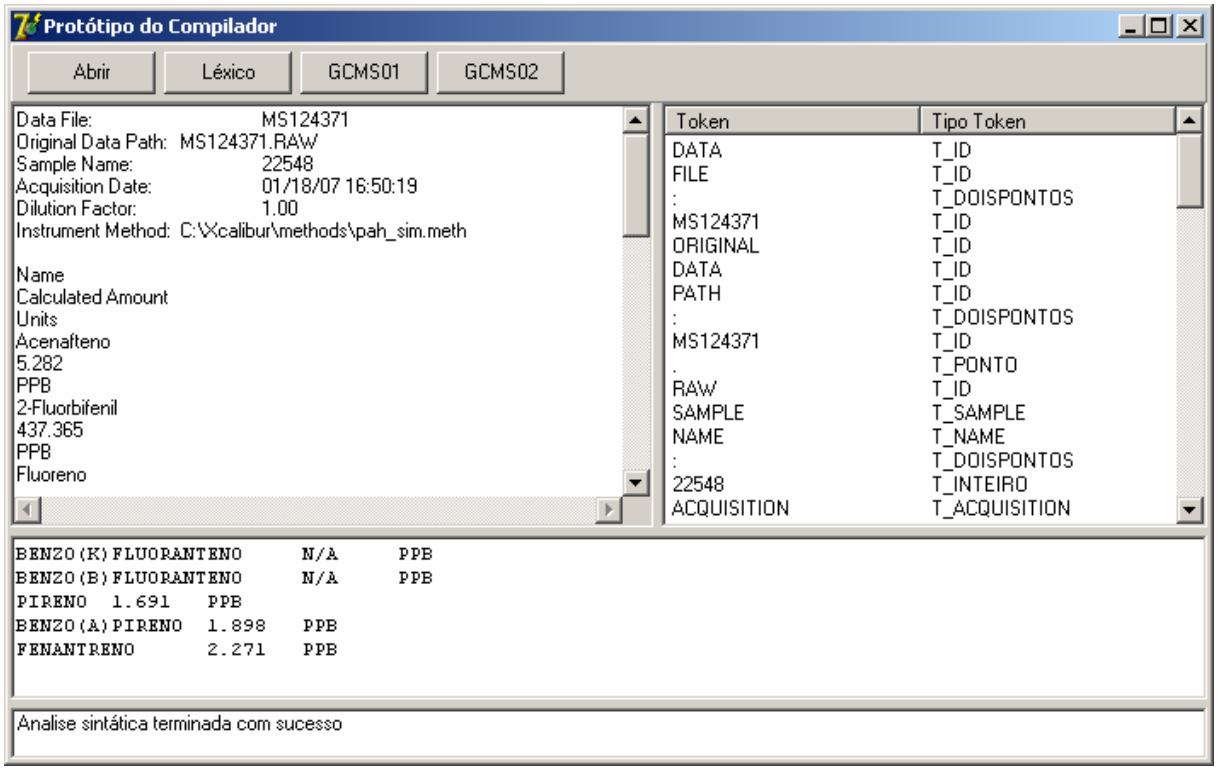

Figura 4.15 - Exemplo de interpretação para dados do GCMS01

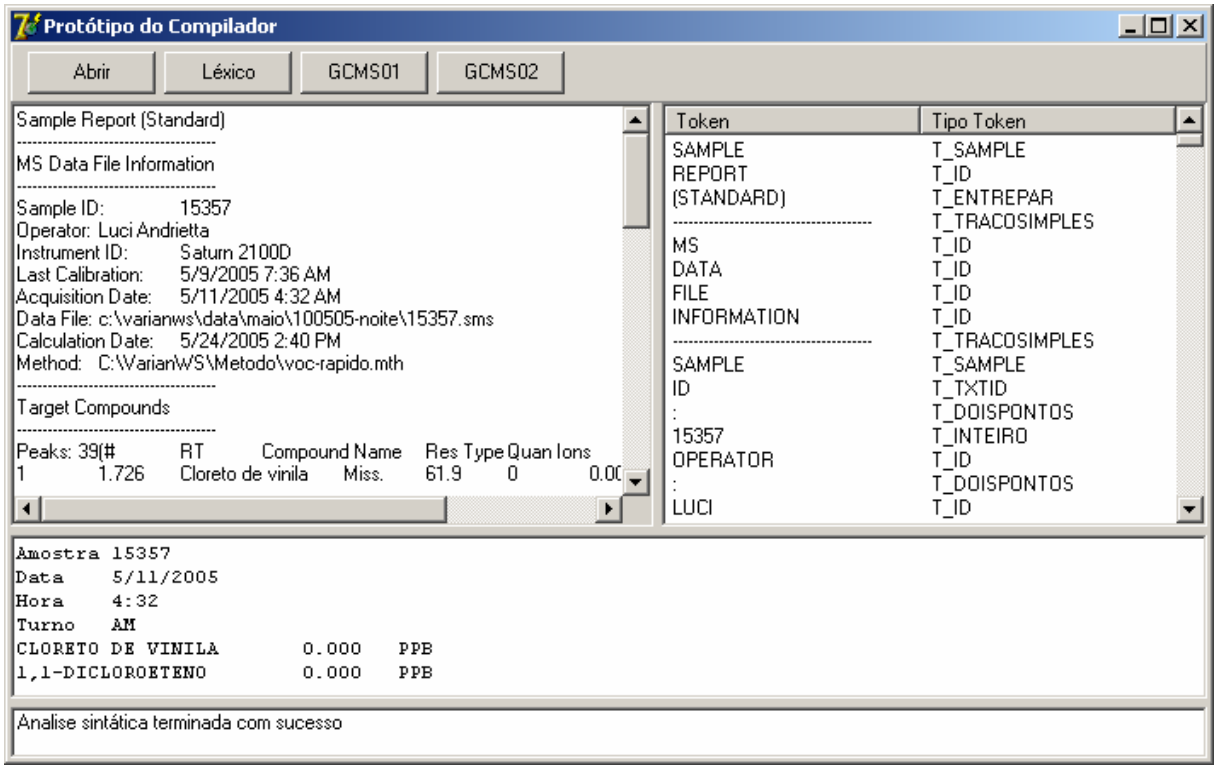

Figura 4.16 - Exemplo de interpretação para dados do GCMS02

5 EXTRAÇÃO DE DADOS

Para validar o compilador desenvolvido, foram feitos diversos testes de extração de dados para cada gramática. Como exemplo de resultados obtidos, é listado a seguir um conjunto de dados brutos, lista de *tokens* encontrados e dados finais processados para cada um dos equipamentos GCMS01 e GCMS02.

5.1 Dados brutos do equipamento GCMS01

```
Data File:              MS408032
Original Data Path:     MS408032.RAW
Sample Name:            PAH_1000_PPB
Acquisition Date:       05/18/07 07:05:03
Dilution Factor:        1.00
Instrument Method:      C:\Xcalibur\methods\pah_sim_0515-07.meth

Name
Calculated Amount
Units
Acenafteno
977.036
PPB
Antraceno
934.808
PPB
Benzo(a)antraceno
887.111
PPB
Indeno(1,2,3-cd)pireno
542.575
PPB
Benzo(g,h,i)perileno
638.330
PPB
2-Fluorbifenil
974.617
PPB
Dibenz(a,h)antraceno
451.649
PPB
Acenaftileno
966.286
PPB
```

```
Fluoreno
977.618
PPB
D10-Fenantreno
N/A
ppb
Fluoranteno
886.852
PPB
Criseno
894.540
PPB
D10-Acenafteno
N/A
ppb
Naftaleno
1018.848
PPB
D14-Terfenil
990.723
PPB
D12-Criseno
N/A
ppb
D12-Perileno
N/A
ppb
Benzo(b)fluoranteno
1130.915
PPB
Benzo(k)fluoranteno
833.264
PPB
D8-Naftaleno
N/A
ppb
Pireno
895.046
PPB
Benzo(a)pireno
835.573
PPB
Fenantreno
957.783
PPB
```

5.2 *Tokens* dos dados brutos do equipamento GCMS01

Tabela 5.1 - *Tokens* dos dados brutos do equipamento GCMS01

Tipo do token	Valor do token
DATA	T_ID
FILE	T_ID
:	T_DOISPONTOS
MS408032	T_ID
ORIGINAL	T_ID
DATA	T_ID
PATH	T_ID

:	T_DOISPONTOS
MS408032	T_ID
.	T_PONTO
RAW	T_ID
SAMPLE	T_SAMPLE
NAME	T_NAME
:	T_DOISPONTOS
PAH_1000_PPB	T_ID
ACQUISITION	T_ACQUISITION
DATE	T_DATE
:	T_DOISPONTOS
05/18/07	T_DATA
07:05:03	T_HORA
DILUTION	T_DILUTION
FACTOR	T_FACTOR
:	T_DOISPONTOS
1.00	T_FRACIONARIO
INSTRUMENT	T_ID
METHOD	T_ID
:	T_DOISPONTOS
C:\XCALIBUR\METHODS\PAH_SIM_0515-07.METH	T_ARQUIVO
NAME	T_NAME
CALCULATED	T_CALCULATED
AMOUNT	T_AMOUNT
UNITS	T_UNITS
ACENAFTENO	T_ID
977.036	T_FRACIONARIO
PPB	T_ID
ANTRACENO	T_ID
934.808	T_FRACIONARIO
PPB	T_ID
BENZO(A)ANTRACENO	T_ID
887.111	T_FRACIONARIO
PPB	T_ID
INDENO(1,2,3-CD)PIRENO	T_ID
542.575	T_FRACIONARIO
PPB	T_ID
BENZO(G,H,I)PERILENO	T_ID
638.330	T_FRACIONARIO
PPB	T_ID
2-FLUORBIFENIL	T_ID
974.617	T_FRACIONARIO
PPB	T_ID
DIBENZ(A,H)ANTRACENO	T_ID
451.649	T_FRACIONARIO
PPB	T_ID
ACENAFTILENO	T_ID
966.286	T_FRACIONARIO

PPB	T_ID
FLUORENO	T_ID
977.618	T_FRACIONARIO
PPB	T_ID
D10-FENANTRENO	T_ID
N/A	T_NA
PPB	T_ID
FLUORANTENO	T_ID
886.852	T_FRACIONARIO
PPB	T_ID
CRISENO	T_ID
894.540	T_FRACIONARIO
PPB	T_ID
D10-ACENAFTENO	T_ID
N/A	T_NA
PPB	T_ID
NAFTALENO	T_ID
1018.848	T_FRACIONARIO
PPB	T_ID
D14-TERFENIL	T_ID
990.723	T_FRACIONARIO
PPB	T_ID
D12-CRISENO	T_ID
N/A	T_NA
PPB	T_ID
D12-PERILENO	T_ID
N/A	T_NA
PPB	T_ID
BENZO(B)FLUORANTENO	T_ID
1130.915	T_FRACIONARIO
PPB	T_ID
BENZO(K)FLUORANTENO	T_ID
833.264	T_FRACIONARIO
PPB	T_ID
D8-NAFTALENO	T_ID
N/A	T_NA
PPB	T_ID
PIRENO	T_ID
895.046	T_FRACIONARIO
PPB	T_ID
BENZO(A)PIRENO	T_ID
835.573	T_FRACIONARIO
PPB	T_ID
FENANTRENO	T_ID
957.783	T_FRACIONARIO
PPB	T_ID
DATA	T_ID
FILE	T_ID
:	T_DOISPONTOS

MS408032	T_ID

5.3 Resultado final do processamento do equipamento GCMS01

Tabela 5.2 - Cabeçalho do processamento do equipamento GCMS01

Informação	Valor
Amostra	PAH_1000_PPB
Data	05/18/07
Hora	07:05:03
Diluição	1.00

Tabela 5.3 - Parâmetros do processamento do equipamento GCMS01

Nome do parâmetro analisado	Valor	Unidade
ACENAFTENO	977.036	PPB
ANTRACENO	934.808	PPB
BENZO(A)ANTRACENO	887.111	PPB
INDENO(1,2,3-CD)PIRENO	542.575	PPB
BENZO(G,H,I)PERILENO	638.330	PPB
2-FLUORBIFENIL	974.617	PPB
DIBENZ(A,H)ANTRACENO	451.649	PPB
ACENAFTILENO	966.286	PPB
FLUORENO	977.618	PPB
D10-FENANTRENO	N/A	PPB
FLUORANTENO	886.852	PPB
CRISENO	894.540	PPB
D10-ACENAFTENO	N/A	PPB
NAFTALENO	1018.848	PPB
D14-TERFENIL	990.723	PPB
D12-CRISENO	N/A	PPB
D12-PERILENO	N/A	PPB
BENZO(B)FLUORANTENO	1130.915	PPB
BENZO(K)FLUORANTENO	833.264	PPB
D8-NAFTALENO	N/A	PPB
PIRENO	895.046	PPB
BENZO(A)PIRENO	835.573	PPB
FENANTRENO	957.783	PPB

5.4 Avaliação da extração de dados do equipamento GCMS01

No exemplo de extração de dados ilustado acima, obteve-se a extração exata de todas as informações desejadas. Para validar a gramática e o compilador desenvolvido, realizou-se a interpretação e extração de dados de 47 diferentes aquivos gerados pelo equipamento GCMS01. Todas as informações previstas foram extraídas com sucesso, o que resultou em precisão de 100%.

5.5 Dados brutos do equipamento GCMS02

```
Sample Report (Standard)
---------------------------------------
MS Data File Information
---------------------------------------
Sample ID:                      controle svoc's
Operator:
Instrument ID:
Last Calibration:             6/15/2007 9:00 AM
Acquisition Date:             10/26/2007 11:44 AM
Data File:                     c:\saturnws\analises 2007\svoc's\outubro
2007\controle svoc's_10-26-2007.sms
Calculation Date:             10/31/2007 9:21 AM
Method:                           C:\SaturnWS\Metodos\Pah's_13_06_07.mth
---------------------------------------
Target Compounds
---------------------------------------
Peaks: 25(
#     RT    Compound Name     Res Type    Quan Ions   Area  Amount)
1   4.523  1,4-Dichlorobenzene-D4  Id.   150      28333  1.000    ug
2   5.959  Naphthalene-D8          Id.   136      77865  1.000    ug
3   8.712  Acenaphthene-d10        Id.   162      7551   1.000    ug
4   12.072 FenantrenoD10           Id.   188      54113  1.000    ug
5   21.231 Criseno-D12             Id.   240      38116  1.000    ug
6   27.872 Perylene-D12            Id.   264      4227   1.000    ug
7   5.104  NitrobenzenoD5          Id.   82+128   134199 2.789    ug
8   5.986  Naphthalene             Id.   128      231738 2.533    ug
9   7.529  1,1-Biphenyl            Id.   172      188918 3.671    ug
10  8.453  Acenaphthylene          Id.   152      211391 15.724   ug
11  8.775  Acenaphthene            Id.   153      140414 15.470   ug
12  9.819  Fluorene                Id.   165+166  277676 2.828    ug
13  12.136 Phenanthrene            Id.   178      183832 2.919    ug
14  12.272 Anthracene              Id.   178      188970 3.152    ug
15  15.567 Fluoranthene            Id.   202      176860 3.101    ug
16  16.253 Pyrene                  Id.   202      182404 3.249    ug
17  16.896 p-Terphenyl-d14         Id.   244      55492  1.278    ug
18  21.177 Benz[a]anthracene       Id.   228      90795  2.579    ug
19  21.359 Chrysene                Id.   228      123251 2.989    ug
20  26.253 BENZO B FLUORANTENO     Id.   252      15829  0.997    ug
21  26.378 BENZO K FLUORANTENO     Id.   252      16521  0.822    ug
22  27.611 BENZO A PIRENO          Id.   252      12277  0.524    ug
23  32.086 Indeno[1,2,3-cd]pyrene  Miss. 276+138  0      0.000    ug
24  32.078 Dibenz[a,h]anthracene   Id.   278+138  1233   0.969    ug
---------------------------------------
Unidentified Peaks
---------------------------------------
None
```

5.6 *Tokens* dos dados brutos do equipamento GCMS02

Tabela 5.4 - *Tokens* dos dados brutos do equipamento GCMS02

Tipo do token	Valor do token
SAMPLE	T_SAMPLE
REPORT	T_ID
(STANDARD)	T_ENTREPAR
---	T_TRACOSIMPLES
MS	T_ID

DATA	T_ID
FILE	T_ID
INFORMATION	T_ID
----------	T_TRACOSIMPLES
SAMPLE	T_SAMPLE
ID	T_TXTID
:	T_DOISPONTOS
CONTROLE	T_ID
SVOC'S	T_ID
OPERATOR	T_ID
:	T_DOISPONTOS
INSTRUMENT	T_ID
ID	T_TXTID
:	T_DOISPONTOS
LAST	T_ID
CALIBRATION	T_ID
:	T_DOISPONTOS
6/15/2007	T_DATA
9:00	T_HORA
AM	T_AM
ACQUISITION	T_ACQUISITION
DATE	T_DATE
:	T_DOISPONTOS
10/26/2007	T_DATA
11:44	T_HORA
AM	T_AM
DATA	T_ID
FILE	T_ID
:	T_DOISPONTOS
C:\SATURNWS\ANALISES 2007\SVOC'S\OUTUBRO 2007\CONTROLE SVOC'S_10-26-2007.SMS	T_ARQUIVO
CALCULATION	T_ID
DATE	T_DATE
:	T_DOISPONTOS
10/31/2007	T_DATA
9:21	T_HORA
AM	T_AM
METHOD	T_ID
:	T_DOISPONTOS
C:\SATURNWS\METODOS\PAH'S_13_06_07.MTH	T_ARQUIVO
----------	T_TRACOSIMPLES
TARGET	T_ID
COMPOUNDS	T_ID
----------	T_TRACOSIMPLES
PEAKS	T_PEAKS
:	T_DOISPONTOS
25	T_INTEIRO
(# RT Compound Name Res Type Quan Ions Area Amount)	T_ENTREPAR
1	T_INTEIRO

4.523	T_FRACIONARIO
1,4-DICHLOROBENZENE-D4	T_ID
ID	T_TXTID
.	T_PONTO
150	T_INTEIRO
28333	T_INTEIRO
1.000	T_FRACIONARIO
UG	T_ID
2	T_INTEIRO
5.959	T_FRACIONARIO
NAPHTHALENE-D8	T_ID
ID	T_TXTID
.	T_PONTO
136	T_INTEIRO
77865	T_INTEIRO
1.000	T_FRACIONARIO
UG	T_ID
3	T_INTEIRO
8.712	T_FRACIONARIO
ACENAPHTHENE-D10	T_ID
ID	T_TXTID
.	T_PONTO
162	T_INTEIRO
7551	T_INTEIRO
1.000	T_FRACIONARIO
UG	T_ID
4	T_INTEIRO
12.072	T_FRACIONARIO
FENANTRENOD10	T_ID
ID	T_TXTID
.	T_PONTO
188	T_INTEIRO
54113	T_INTEIRO
1.000	T_FRACIONARIO
UG	T_ID
5	T_INTEIRO
21.231	T_FRACIONARIO
CRISENO-D12	T_ID
ID	T_TXTID
.	T_PONTO
240	T_INTEIRO
38116	T_INTEIRO
1.000	T_FRACIONARIO
UG	T_ID
6	T_INTEIRO
27.872	T_FRACIONARIO
PERYLENE-D12	T_ID
ID	T_TXTID
.	T_PONTO

264	T_INTEIRO
4227	T_INTEIRO
1.000	T_FRACIONARIO
UG	T_ID
7	T_INTEIRO
5.104	T_FRACIONARIO
NITROBENZENOD5	T_ID
ID	T_TXTID
.	T_PONTO
82+128	T_ID
134199	T_INTEIRO
2.789	T_FRACIONARIO
UG	T_ID
8	T_INTEIRO
5.986	T_FRACIONARIO
NAPHTHALENE	T_ID
ID	T_TXTID
.	T_PONTO
128	T_INTEIRO
231738	T_INTEIRO
2.533	T_FRACIONARIO
UG	T_ID
9	T_INTEIRO
7.529	T_FRACIONARIO
1,1-BIPHENYL	T_ID
ID	T_TXTID
.	T_PONTO
172	T_INTEIRO
188918	T_INTEIRO
3.671	T_FRACIONARIO
UG	T_ID
10	T_INTEIRO
8.453	T_FRACIONARIO
ACENAPHTHYLENE	T_ID
ID	T_TXTID
.	T_PONTO
152	T_INTEIRO
211391	T_INTEIRO
15.724	T_FRACIONARIO
UG	T_ID
11	T_INTEIRO
8.775	T_FRACIONARIO
ACENAPHTHENE	T_ID
ID	T_TXTID
.	T_PONTO
153	T_INTEIRO
140414	T_INTEIRO
15.470	T_FRACIONARIO
UG	T_ID

12	T_INTEIRO
9.819	T_FRACIONARIO
FLUORENE	T_ID
ID	T_TXTID
.	T_PONTO
165+166	T_ID
277676	T_INTEIRO
2.828	T_FRACIONARIO
UG	T_ID
13	T_INTEIRO
12.136	T_FRACIONARIO
PHENANTHRENE	T_ID
ID	T_TXTID
.	T_PONTO
178	T_INTEIRO
183832	T_INTEIRO
2.919	T_FRACIONARIO
UG	T_ID
14	T_INTEIRO
12.272	T_FRACIONARIO
ANTHRACENE	T_ID
ID	T_TXTID
.	T_PONTO
178	T_INTEIRO
188970	T_INTEIRO
3.152	T_FRACIONARIO
UG	T_ID
15	T_INTEIRO
15.567	T_FRACIONARIO
FLUORANTHENE	T_ID
ID	T_TXTID
.	T_PONTO
202	T_INTEIRO
176860	T_INTEIRO
3.101	T_FRACIONARIO
UG	T_ID
16	T_INTEIRO
16.253	T_FRACIONARIO
PYRENE	T_ID
ID	T_TXTID
.	T_PONTO
202	T_INTEIRO
182404	T_INTEIRO
3.249	T_FRACIONARIO
UG	T_ID
17	T_INTEIRO
16.896	T_FRACIONARIO
P-TERPHENYL-D14	T_ID
ID	T_TXTID

.	T_PONTO
244	T_INTEIRO
55492	T_INTEIRO
1.278	T_FRACIONARIO
UG	T_ID
18	T_INTEIRO
21.177	T_FRACIONARIO
BENZ[A]ANTHRACENE	T_ID
ID	T_TXTID
.	T_PONTO
228	T_INTEIRO
90795	T_INTEIRO
2.579	T_FRACIONARIO
UG	T_ID
19	T_INTEIRO
21.359	T_FRACIONARIO
CHRYSENE	T_ID
ID	T_TXTID
.	T_PONTO
228	T_INTEIRO
123251	T_INTEIRO
2.989	T_FRACIONARIO
UG	T_ID
20	T_INTEIRO
26.253	T_FRACIONARIO
BENZO	T_ID
B	T_ID
FLUORANTENO	T_ID
ID	T_TXTID
.	T_PONTO
252	T_INTEIRO
15829	T_INTEIRO
0.997	T_FRACIONARIO
UG	T_ID
21	T_INTEIRO
26.378	T_FRACIONARIO
BENZO	T_ID
K	T_ID
FLUORANTENO	T_ID
ID	T_TXTID
.	T_PONTO
252	T_INTEIRO
16521	T_INTEIRO
0.822	T_FRACIONARIO
UG	T_ID
22	T_INTEIRO
27.611	T_FRACIONARIO
BENZO	T_ID
A	T_ID

PIRENO	T_ID
ID	T_TXTID
.	T_PONTO
252	T_INTEIRO
12277	T_INTEIRO
0.524	T_FRACIONARIO
UG	T_ID
23	T_INTEIRO
32.086	T_FRACIONARIO
INDENO[1,2,3-CD]PYRENE	T_ID
MISS	T_MISS
.	T_PONTO
276+138	T_ID
0	T_INTEIRO
0.000	T_FRACIONARIO
UG	T_ID
24	T_INTEIRO
32.078	T_FRACIONARIO
DIBENZ[A,H]ANTHRACENE	T_ID
ID	T_TXTID
.	T_PONTO
278+138	T_ID
1233	T_INTEIRO
0.969	T_FRACIONARIO
UG	T_ID
----------	T_TRACOSIMPLES
UNIDENTIFIED	T_ID
PEAKS	T_PEAKS
----------	T_TRACOSIMPLES
NONE	T_ID

5.7 Resultado final do processamento do equipamento GCMS02

Tabela 5.5 - Cabeçalho do processamento do equipamento GCMS02

Informação	Valor
Amostra	CONTROLE
Data	10/26/2007
Hora	11:44
Turno	AM

Tabela 5.6 - Parâmetros do processamento do equipamento GCMS02

Nome do parâmetro analisado	Valor	Unidade
1,4-DICHLOROBENZENE-D4	1.000	UG
NAPHTHALENE-D8	1.000	UG
ACENAPHTHENE-D10	1.000	UG
FENANTRENOD10	1.000	UG

CRISENO-D12	1.000	UG
PERYLENE-D12	1.000	UG
NITROBENZENOD5	2.789	UG
NAPHTHALENE	2.533	UG
1,1-BIPHENYL	3.671	UG
ACENAPHTHYLENE	15.724	UG
ACENAPHTHENE	15.470	UG
FLUORENE	2.828	UG
PHENANTHRENE	2.919	UG
ANTHRACENE	3.152	UG
FLUORANTHENE	3.101	UG
PYRENE	3.249	UG
P-TERPHENYL-D14	1.278	UG
BENZ[A]ANTHRACENE	2.579	UG
CHRYSENE	2.989	UG
BENZO B FLUORANTENO	0.997	UG
BENZO K FLUORANTENO	0.822	UG
BENZO A PIRENO	0.524	UG
INDENO[1,2,3-CD]PYRENE	0.000	UG
DIBENZ[A,H]ANTHRACENE	0.969	UG

5.8 Avaliação da extração de dados do equipamento GCMS02

No exemplo de extração de dados ilustado acima, obteve-se a extração exata de todas as informações desejadas. Para validar a gramática e o compilador desenvolvido, realizou-se a interpretação e extração de dados de 42 diferentes aquivos gerados pelo equipamento GCMS02. Todas as informações previstas foram extraídas com sucesso, o que resultou em precisão de 100%.

CONCLUSÃO

O objetivo principal deste trabalho concentrou-se no estudo de compiladores para serem usados na interpretação de dados brutos e extração de informações, e não para serem usados na sua maneira mais usual, ou seja, geradores de códigos de máquinas.

A parte teórica deste estudo afirma a possibilidade de se desenvolver um compilador aplicado à extração de informações a partir de um conjunto de dados brutos, embora o uso clássico de compiladores consiste em leitura de uma linguagem de programação de alto nível e estruturada e geração de código de máquina executável.

A utilização de técnicas de compiladores para a extração de informações a partir de dados brutos é plenamente viável, pois permite desenvolver uma solução elegante e de alto nível com relativa baixa complexidade para um problema muito comum em laboratórios de análises ambientais.

O estudo sobre rotinas de laboratórios de análises ambientais e a detecção da necessidade de automatização da leitura de resultados analíticos ocorreu durante dez anos, em trabalhos diários com automação de diversos processos em dezenas de laboratórios de controle de qualidade nas regiões sul e sudeste do Brasil.

O desenvolvimento do protótipo do compilador para extração das informações dos dados brutos contidos em relatórios de análises de equipamentos analíticos comprova a viabilidade técnica da solução proposta para o problema analisado.

As dezenas de testes de extração de dados, realizados com vários arquivos de dados brutos ratifica a versatilidade, flexibilidade e exatidão com que os dados são interpretados e extraídos. Após os ajustes dos analisadores léxico e sintático, o nível de falhas ocorridas na interpretação dos dados foi inexistente.

Existe uma real possibilidade de evolução neste trabalho para aumentar sua flexibilidade e ampliar a aplicação da solução para arquivos de dados brutos de outros equipamentos analíticos. Uma maneira de se fazer isso é transformar a aplicação atual, de um simples compilador com analisador léxico e sintático embutidos no código fonte, para um gerador de compiladores. A definição das expressões regulares dos *tokens* e definição das regras gramaticais podem ser feitos em arquivos externos, com possibilidade de definição das regras pelos próprios usuários da aplicação.

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, Alfred V. Aho; SETHI, Ravi; ULLMAN, Jeffrey D. **COMPILADORES: princípios, técnicas e ferramentas.** Rio de Janeiro: LTC, 1995.

FISCHER, Charles N.; LEBLANC, Richard J. **Crafting a compiler with C.** California: Benjamin Cummings, 1991.

GRUNE, Dick et. al. **Projeto moderno de compiladores: implementação e aplicações.** Rio de Janeiro: Campus, 2001.

KAPLAN, Randy M. **Constructing language processors for little languages.** New York: John Wiley, 1994.

NETO, João J. **Introdução à compilação.** Rio de Janeiro: LTC, 1987.

PITTMAN, Thomas; PETERS, James. **The art of compiler design: theory and practice.** New Jersey: Prentice-hall, 1992.

RANGEL, José Lucas. **Material didático relativo à disciplina de Compiladores.** 1999. Disponível em: <http://www-di.inf.puc-rio.br/~rangel/comp.html> Acesso em: 12 de maio de 2007.

SEBESTA, Robert W. **Conceitos de linguagens de programação**. Porto Alegre: Bookman, 2000.

TORRES, Gabriel. **Hardware: curso completo** - 4. ed. 2001.

APÊNDICE A - CÓDIGO FONTE

Arquivo Protótipo.dpr

```
program Prototipo;

uses
  Forms,
  principal in 'principal.pas' {frmPrincipal},
  uLexico in 'uLexico.pas',
  uGCMS01 in 'uGCMS01.pas',
  uGCMS02 in 'uGCMS02.pas';

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TfrmPrincipal, frmPrincipal);
  Application.Run;
end.
```

Arquivo principal.pas

```
unit principal;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, ComCtrls, ClipBrd;

type
  TfrmPrincipal = class(TForm)
    pnlSuperior: TPanel;
    btnAbrir: TButton;
    btnLexico: TButton;
    btnGCMS01: TButton;
    btnGCMS02: TButton;
    spt01: TSplitter;
    spt02: TSplitter;
    spt03: TSplitter;
    mmDadosBrutos: TMemo;
    lvwTokens: TListView;
    lbMensagens: TListBox;
    mmSaida: TMemo;
    odDadosBrutos: TOpenDialog;
    procedure btnLexicoClick(Sender: TObject);
    procedure btnGCMS01Click(Sender: TObject);
    procedure btnGCMS02Click(Sender: TObject);
    procedure btnAbrirClick(Sender: TObject);
    procedure lvwTokensKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure lbMensagensKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure EscreveToken;
    procedure EscreveMensagem(AMsg: string);
    procedure EscreveMensagemTokenEsperado(ATokenEsperado, ANmRegra: string);
  end;

var
  frmPrincipal: TfrmPrincipal;

implementation

uses uLexico, uGCMS01, uGCMS02;

{$R *.dfm}

procedure TfrmPrincipal.btnLexicoClick(Sender: TObject);
begin
  lvwTokens.Clear;
  lbMensagens.Clear;
  InicializaLexico(mmDadosBrutos.Lines.Text);

  while PegaProximoToken do ;
end;

procedure TfrmPrincipal.EscreveMensagem(AMsg: string);
begin
  lbMensagens.Items.Add(AMsg)
end;

procedure TfrmPrincipal.EscreveMensagemTokenEsperado(ATokenEsperado,
 ANmRegra: string);
begin
```

```
  EscreveMensagem(
    'Esperava "' + ATokenEsperado +
    '" mas encontrei "' + Token +
    '" em "' + ANmRegra + '"');
end;

procedure TfrmPrincipal.EscreveToken;
begin
  with lvwTokens.Items.Add do
    begin
      Caption := Token;
      SubItems.Add(TokenToStr(TipoToken));
    end;
end;

procedure TfrmPrincipal.btnGCMS01Click(Sender: TObject);
var
  GCMS01: TGCMS01;
begin
  mmSaida.Clear;
  lvwTokens.Clear;
  lbMensagens.Clear;
  InicializaLexico(mmDadosBrutos.Lines.Text);

  GCMS01 := TGCMS01.Create;
  GCMS01.Sintatico;
  FreeAndNil(GCMS01);
end;

procedure TfrmPrincipal.btnGCMS02Click(Sender: TObject);
var
  GCMS02: TGCMS02;
begin
  mmSaida.Clear;
  lvwTokens.Clear;
  lbMensagens.Clear;
  InicializaLexico(mmDadosBrutos.Lines.Text);

  GCMS02 := TGCMS02.Create;
  GCMS02.Sintatico;
  FreeAndNil(GCMS02);
end;

procedure TfrmPrincipal.btnAbrirClick(Sender: TObject);
begin
  if odDadosBrutos.Execute then
    mmDadosBrutos.Lines.LoadFromFile(odDadosBrutos.FileName);
end;

procedure TfrmPrincipal.lvwTokensKeyDown(Sender: TObject; var Key: Word;
 Shift: TShiftState);
var
  i: Integer;
  str: string;
begin
  if UpCase(Char(Key)) = 'C' then
    if Shift = [ssCtrl] then
      begin
        for i := 0 to lvwTokens.Items.Count - 1 do
          str := str +
            lvwTokens.Items[i].Caption + #9 +
            lvwTokens.Items[i].SubItems[0] + #10;
          Clipboard.AsText := str;
      end;
end;

procedure TfrmPrincipal.lbMensagensKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
```

```
begin
  if UpCase(Char(Key)) = 'C' then
    if Shift = [ssCtrl] then
      Clipboard.AsText := lbMensagens.Items.Text;
end;

end.
```

Arquivo principal.dfm

```
object frmPrincipal: TfrmPrincipal
  Left = 158
  Top = 104
  Width = 844
  Height = 480
  Caption = 'Protótipo do Compilador'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  Position = poScreenCenter
  WindowState = wsMaximized
  PixelsPerInch = 96
  TextHeight = 13
  object spt01: TSplitter
    Left = 511
    Top = 35
    Width = 5
    Height = 229
    Align = alRight
  end
  object spt03: TSplitter
    Left = 0
    Top = 358
    Width = 836
    Height = 5
    Cursor = crVSplit
    Align = alBottom
  end
  object spt02: TSplitter
    Left = 0
    Top = 264
    Width = 836
    Height = 5
    Cursor = crVSplit
    Align = alBottom
  end
  object pnlSuperior: TPanel
    Left = 0
    Top = 0
    Width = 836
    Height = 35
    Align = alTop
    TabOrder = 0
    object btnLexico: TButton
      Left = 90
      Top = 5
      Width = 75
      Height = 25
      Caption = 'Léxico'
      TabOrder = 1
      OnClick = btnLexicoClick
    end
    object btnGCMS01: TButton
      Left = 170
      Top = 5
      Width = 75
      Height = 25
      Caption = 'GCMS01'
      TabOrder = 2
      OnClick = btnGCMS01Click
    end
    object btnGCMS02: TButton
```

```
    Left = 250
    Top = 5
    Width = 75
    Height = 25
    Caption = 'GCMS02'
    TabOrder = 3
    OnClick = btnGCMS02Click
  end
  object btnAbrir: TButton
    Left = 10
    Top = 5
    Width = 75
    Height = 25
    Caption = 'Abrir'
    TabOrder = 0
    OnClick = btnAbrirClick
  end
end
object mmDadosBrutos: TMemo
  Left = 0
  Top = 35
  Width = 511
  Height = 229
  Align = alClient
  ScrollBars = ssBoth
  TabOrder = 1
end
object lvwTokens: TListView
  Left = 516
  Top = 35
  Width = 320
  Height = 229
  Align = alRight
  Columns = <
    item
      Caption = 'Token'
      Width = 150
    end
    item
      Caption = 'Tipo Token'
      Width = 150
    end>
  TabOrder = 2
  ViewStyle = vsReport
  OnKeyDown = lvwTokensKeyDown
end
object lbMensagens: TListBox
  Left = 0
  Top = 363
  Width = 836
  Height = 90
  Align = alBottom
  ItemHeight = 13
  TabOrder = 4
  OnKeyDown = lbMensagensKeyDown
end
object mmSaida: TMemo
  Left = 0
  Top = 269
  Width = 836
  Height = 89
  Align = alBottom
  Font.Charset = ANSI_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Courier New'
  Font.Style = []
  ParentFont = False
```

```
    TabOrder = 3
  end
  object odDadosBrutos: TOpenDialog
    DefaultExt = '*.*'
    Left = 10
    Top = 45
  end
end
```

Arquivo uLexico.pas

```
unit uLexico;

interface

uses SysUtils;

type
  TTipoToken = (
    T_UNKNOWN,
    T_ERROLEX,
    T_FIMARQUIVO,
    T_ID,
    T_INTEIRO,
    T_FRACIONARIO,
    T_DATA,
    T_HORA,
    T_IGUAL,
    T_VEZES,
    T_MENOS,
    T_PONTO,
    T_VIRGULA,
    T_DOISPONTOS,
    T_PONTOVIRGULA,
    T_TRACOSIMPLES,
    T_TRACODUPLO,
    T_TRACOVEZES,
    T_ENTREPAR,
    T_ABREPAR,
    T_FECHAPAR,
    T_ARQUIVO,
    T_COMENTARIO,
    T_AM,
    T_PM,
    T_PEAKS,
    T_SAMPLE,
    T_NAME,
    T_ACQUISITION,
    T_DATE,
    T_DILUTION,
    T_FACTOR,
    T_CALCULATED,
    T_AMOUNT,
    T_UNITS,
    T_MISS,
    T_FAIL,
    T_TXTID,
    T_NA);

function RemoveAcento(AChr: Char): Char;
function TokenToStr(ATipoToken: TTipoToken): string;
function PegaProximoToken(APrint: Boolean = True): Boolean;

procedure ProcessaData;
procedure ProcessaHora;

procedure MoveLookAhead;
procedure InicializaLexico(AFonte: string);

var
  Token: string;
  TipoToken: TTipoToken;

implementation

uses Classes, principal;
```

```
var
  LookAhead: Char;
  Ponteiro: Integer;
  TextoFonte: string;

function RemoveAcento(AChr: Char): Char;
const
  ComAcento = 'àâêôûãõáéíóúçüÀÂÊÔÛÃÕÁÉÍÓÚÇÜ';
  SemAcento = 'aaeouaoaeioucuAAEOUAOAEIOUCU';
begin
  if Pos(AChr, ComAcento) > 0 then
    Result := SemAcento[Pos(AChr, ComAcento)]
  else
    Result := AChr;
end;

function TokenToStr(ATipoToken: TTipoToken): string;
begin
  case TipoToken of
    T_UNKNOWN      : Result := 'T_UNKNOWN';
    T_ERROLEX      : Result := 'T_ERROLEX';
    T_FIMARQUIVO   : Result := 'T_FIMARQUIVO';
    T_ID           : Result := 'T_ID';
    T_INTEIRO      : Result := 'T_INTEIRO';
    T_FRACIONARIO  : Result := 'T_FRACIONARIO';
    T_DATA         : Result := 'T_DATA';
    T_HORA         : Result := 'T_HORA';
    T_IGUAL        : Result := 'T_IGUAL';
    T_VEZES        : Result := 'T_VEZES';
    T_MENOS        : Result := 'T_MENOS';
    T_PONTO        : Result := 'T_PONTO';
    T_VIRGULA      : Result := 'T_VIRGULA';
    T_DOISPONTOS   : Result := 'T_DOISPONTOS';
    T_PONTOVIRGULA : Result := 'T_PONTOVIRGULA';
    T_TRACOSIMPLES : Result := 'T_TRACOSIMPLES';
    T_TRACODUPLO   : Result := 'T_TRACODUPLO';
    T_TRACOVEZES   : Result := 'T_TRACOVEZES';
    T_ENTREPAR     : Result := 'T_ENTREPAR';
    T_ABREPAR      : Result := 'T_ABREPAR';
    T_FECHAPAR     : Result := 'T_FECHAPAR';
    T_ARQUIVO      : Result := 'T_ARQUIVO';
    T_COMENTARIO   : Result := 'T_COMENTARIO';
    T_AM           : Result := 'T_AM';
    T_PM           : Result := 'T_PM';
    T_PEAKS        : Result := 'T_PEAKS';
    T_SAMPLE       : Result := 'T_SAMPLE';
    T_NAME         : Result := 'T_NAME';
    T_ACQUISITION  : Result := 'T_ACQUISITION';
    T_DATE         : Result := 'T_DATE';
    T_DILUTION     : Result := 'T_DILUTION';
    T_FACTOR       : Result := 'T_FACTOR';
    T_CALCULATED   : Result := 'T_CALCULATED';
    T_AMOUNT       : Result := 'T_AMOUNT';
    T_UNITS        : Result := 'T_UNITS';
    T_MISS         : Result := 'T_MISS';
    T_FAIL         : Result := 'T_FAIL';
    T_TXTID        : Result := 'T_TXTID';
    T_NA           : Result := 'T_NA';
  else
    Result := 'T_UNKNOWN';
  end;
end;

function PegaProximoToken(APrint: Boolean = True): Boolean;
begin
  Token := '';
  TipoToken := T_UNKNOWN;
```

```
// Pula espacos
  while LookAhead in [' ', #9, #10, #12, #13] do
    MoveLookAhead;

  if LookAhead in['A'..'Z', 'µ'] then
    begin
      Token := LookAhead;
      MoveLookAhead;

// Identifica N/A
      if (Token = 'N') and (LookAhead = '/') then
        begin
          Token := Token + LookAhead;
          MoveLookAhead;

          if LookAhead = 'A' then
            begin
              Token := Token + LookAhead;
              MoveLookAhead;
              TipoToken := T_NA;
            end
          else
            begin
              Token := Token + LookAhead;
              MoveLookAhead;
              TipoToken := T_ERROLEX;
            end
        end
      else
        begin
// Identifica caminho e nome de arquivo
          if LookAhead = ':' then
            begin
              Token := Token + LookAhead;
              MoveLookAhead;

              if LookAhead = '\' then
                begin
                  Token := Token + LookAhead;
                  MoveLookAhead;
                end;

// Consome até encontrar extensão do arquivo
                while not (LookAhead in['.', #0]) do
                  begin
                    Token := Token + LookAhead;
                    MoveLookAhead;
                  end;

                if LookAhead = '.' then
                  begin
                    while not (LookAhead in[' ', #0, #9, #10, #12, #13]) do
                      begin
                        Token := Token + LookAhead;
                        MoveLookAhead;
                      end;
                    TipoToken := T_ARQUIVO;
                  end
                else
// Extensão não encontrada
                  begin
                    while not (LookAhead in[' ', #0, #9, #10, #12, #13]) do
                      begin
                        Token := Token + LookAhead;
                        MoveLookAhead;
                      end;

                    TipoToken := T_ERROLEX;
```

```
                end;
            end
          else
            begin
              while not (LookAhead in[':', '.', ' ', #0, #9, #10, #12, #13]) do
                begin
                  Token := Token + LookAhead;
                  MoveLookAhead;
                end;

              if Token = 'AM'           then TipoToken := T_AM else
              if Token = 'PM'           then TipoToken := T_PM else
              if Token = 'PEAKS'        then TipoToken := T_PEAKS else
              if Token = 'SAMPLE'       then TipoToken := T_SAMPLE else
              if Token = 'NAME'         then TipoToken := T_NAME else
              if Token = 'ACQUISITION'  then TipoToken := T_ACQUISITION else
              if Token = 'DATE'         then TipoToken := T_DATE else
              if Token = 'DILUTION'     then TipoToken := T_DILUTION else
              if Token = 'FACTOR'       then TipoToken := T_FACTOR else
              if Token = 'CALCULATED'   then TipoToken := T_CALCULATED else
              if Token = 'AMOUNT'       then TipoToken := T_AMOUNT else
              if Token = 'UNITS'        then TipoToken := T_UNITS else
              if Token = 'MISS'         then TipoToken := T_MISS else
              if Token = 'FAIL'         then TipoToken := T_FAIL else
              if Token = 'ID'           then TipoToken := T_TXTID else
                                             TipoToken := T_ID;
            end;
        end;
    end
  else
// Identifica início de números
    if LookAhead in ['0'..'9'] then
      begin
        Token := LookAhead;
        MoveLookAhead;

        if LookAhead in ['0'..'9'] then
          begin
            Token := Token + LookAhead;
            MoveLookAhead;
          end;

// Identifica data e hora
        if LookAhead in ['/', ':'] then
          begin
            if LookAhead = '/' then
              ProcessaData;

            if LookAhead = ':' then
              ProcessaHora;
          end
        else
          begin
// Identifica números
            while LookAhead in ['0'..'9'] do
              begin
                Token := Token + LookAhead;
                MoveLookAhead;
              end;

            if LookAhead in ['.', ','] then
              begin
                Token := Token + LookAhead;
                MoveLookAhead;

                while LookAhead in ['0'..'9'] do
                  begin
                    Token := Token + LookAhead;
```

```
              MoveLookAhead;
            end;

          if LookAhead in [' ', '(', #0, #9, #10, #12, #13] then
            TipoToken := T_FRACIONARIO
          else
            begin
              while not (LookAhead in[' ', '(', #0, #9, #10, #12, #13]) do
                begin
                  Token := Token + LookAhead;
                  MoveLookAhead;
                end;
              TipoToken := T_ID;
            end;
        end
      else
        if LookAhead in [' ', '(', #0, #9, #10, #12, #13] then
          TipoToken := T_INTEIRO
        else
          begin
            while not (LookAhead in [' ', '(', #0, #9, #10, #12, #13]) do
              begin
                Token := Token + LookAhead;
                MoveLookAhead;
              end;
            TipoToken := T_ID;
          end;
    end;
  end
else
  begin
    case LookAhead of
      '.': TipoToken := T_PONTO;
      ':': TipoToken := T_DOISPONTOS;
      ';': TipoToken := T_PONTOVIRGULA;
      ',': TipoToken := T_VIRGULA;
      '(':
        begin
          Token := Token + LookAhead;
          MoveLookAhead;

          while not (LookAhead in [')', #0]) do
            begin
              Token := Token + LookAhead;
              MoveLookAhead;
            end;

          if LookAhead = ')' then
            begin
              Token := Token + LookAhead;
              MoveLookAhead;
              TipoToken := T_ENTREPAR;
            end
          else
            TipoToken := T_ERROLEX;
        end;
      '-':
        begin
          Token := Token + LookAhead;
          MoveLookAhead;

          if LookAhead <> '-' then
            TipoToken := T_MENOS
          else
            begin
              while LookAhead = '-' do
                begin
                  Token := Token + LookAhead;
```

```
            MoveLookAhead;
          end;

        if LookAhead in [' ', #0, #9, #10, #12, #13] then
          TipoToken := T_TRACOSIMPLES
        else
          TipoToken := T_ERROLEX;
      end;
  end;
'=':
  begin
    Token := Token + LookAhead;
    MoveLookAhead;

    if LookAhead <> '=' then
      TipoToken := T_IGUAL
    else
      begin
        while LookAhead = '=' do
          begin
            Token := Token + LookAhead;
            MoveLookAhead;
          end;

        if LookAhead in [' ', #0, #9, #10, #12, #13] then
          TipoToken := T_TRACODUPLO
        else
          TipoToken := T_ERROLEX;
      end;
  end;
'*':
  begin
    Token := Token + LookAhead;
    MoveLookAhead;

    if LookAhead <> '*' then
      TipoToken := T_VEZES
    else
      begin
        while LookAhead = '*' do
          begin
            Token := Token + LookAhead;
            MoveLookAhead;
          end;

        if LookAhead in [' ', #0, #9, #10, #12, #13] then
          TipoToken := T_TRACOVEZES
        else
          TipoToken := T_ERROLEX;
      end;
  end;
else
  begin
    TipoToken := T_ERROLEX;
    Token := Token + LookAhead;
    MoveLookAhead;

    while not (LookAhead in [' ', #0, #9, #10, #12, #13]) do
      begin
        Token := Token + LookAhead;
        MoveLookAhead;
      end;
  end;
end;

if Token = '' then
  begin
    Token := LookAhead;
```

```
            MoveLookAhead;
          end;
      end;

  Result := Trim(Token) <> '';

  if Result = False then
    TipoToken := T_FIMARQUIVO;

  if APrint then
    frmPrincipal.EscreveToken;
end;

procedure InicializaLexico(AFonte: string);
begin
  LookAhead := ' ';
  Ponteiro := 0;
  TextoFonte := AFonte;
end;

procedure MoveLookAhead;
begin
  if Ponteiro < Length(TextoFonte) then
    begin
      Inc(Ponteiro);
      LookAhead := UpCase(RemoveAcento(TextoFonte[Ponteiro]));
    end
  else
    LookAhead := #0;
end;

procedure ProcessaData;
begin
// Já entra com 1 ou 2 dígitos

// Primeiro divisor de data
  Token := Token + LookAhead;
  MoveLookAhead;

  if LookAhead in ['0'..'9'] then
    begin
      Token := Token + LookAhead;
      MoveLookAhead;

      if LookAhead in ['0'..'9'] then
        begin
          Token := Token + LookAhead;
          MoveLookAhead;
        end;

// Segundo divisor de data
      if LookAhead = '/' then
        begin
          Token := Token + LookAhead;
          MoveLookAhead;

          if LookAhead in ['0'..'9'] then
            begin
              Token := Token + LookAhead;
              MoveLookAhead;

              if LookAhead in ['0'..'9'] then
                begin
                  Token := Token + LookAhead;
                  MoveLookAhead;

// Ano com dois dígitos
                  if LookAhead in [' ', #0, #9, #10, #12, #13] then
```

```
                    TipoToken := T_DATA
                  else
                    if LookAhead in ['0'..'9'] then
                      begin
                        Token := Token + LookAhead;
                        MoveLookAhead;

                        if LookAhead in ['0'..'9'] then
                          begin
                            Token := Token + LookAhead;
                            MoveLookAhead;

// Ano com quatro dígitos
                            if LookAhead in [' ', #0, #9, #10, #12, #13] then
                              TipoToken := T_DATA
                            else
                              TipoToken := T_ERROLEX;
                          end
                        else
                          TipoToken := T_ERROLEX;
                      end
                    else
                      TipoToken := T_ERROLEX;
                end
              else
                while not (LookAhead in [' ', #0, #9, #10, #12, #13]) do
                  begin
                    Token := Token + LookAhead;
                    MoveLookAhead;
                  end;
            end
          else
            while not (LookAhead in [' ', #0, #9, #10, #12, #13]) do
              begin
                Token := Token + LookAhead;
                MoveLookAhead;
              end;
        end
      else
        while not (LookAhead in [' ', #0, #9, #10, #12, #13]) do
          begin
            Token := Token + LookAhead;
            MoveLookAhead;
          end;
    end
  else
    while not (LookAhead in [' ', #0, #9, #10, #12, #13]) do
      begin
        Token := Token + LookAhead;
        MoveLookAhead;
      end;

  if TipoToken = T_UNKNOWN then
    TipoToken := T_ERROLEX;
end;

procedure ProcessaHora;
begin
// Já entra com 1 ou 2 dígitos

// Primeiro divisor de hora
  Token := Token + LookAhead;
  MoveLookAhead;

  if LookAhead in ['0'..'9'] then
    begin
      Token := Token + LookAhead;
      MoveLookAhead;
```

```
      if LookAhead in ['0'..'9'] then
        begin
          Token := Token + LookAhead;
          MoveLookAhead;
        end;

      if LookAhead in [' ', #0, #9, #10, #12, #13] then
        TipoToken := T_HORA
      else
// Segundo divisor de hora
        if LookAhead = ':' then
          begin
            Token := Token + LookAhead;
            MoveLookAhead;

            if LookAhead in ['0'..'9'] then
              begin
                Token := Token + LookAhead;
                MoveLookAhead;

                if LookAhead in ['0'..'9'] then
                  begin
                    Token := Token + LookAhead;
                    MoveLookAhead;
                  end;

                if LookAhead in [' ', #0, #9, #10, #12, #13] then
                  TipoToken := T_HORA
                else
                  while not (LookAhead in [' ', #0, #9, #10, #12, #13]) do
                    begin
                      Token := Token + LookAhead;
                      MoveLookAhead;
                    end;
              end
            else
// Terceiro bloco não numérico
              while not (LookAhead in [' ', #0, #9, #10, #12, #13]) do
                begin
                  Token := Token + LookAhead;
                  MoveLookAhead;
                end;
          end
        else
// Não encontrou segundo divisor de hora
          while not (LookAhead in [' ', #0, #9, #10, #12, #13]) do
            begin
              Token := Token + LookAhead;
              MoveLookAhead;
            end;
    end
  else
// Segundo bloco não numérico
    while not (LookAhead in [' ', #0, #9, #10, #12, #13]) do
      begin
        Token := Token + LookAhead;
        MoveLookAhead;
      end;

  if TipoToken = T_UNKNOWN then
    TipoToken := T_ERROLEX;
end;

end.
```

Arquivo uGCMS01.pas

```
unit uGCMS01;

interface

uses SysUtils;

type

  TGCMS01 = class
    FSaida: string;
    FNumeroErros: Integer;
  private
    { Private declarations }
    procedure Gramatica;
    procedure CabAmostra;
    procedure Dados;
    procedure Dado;
    procedure IdAmostra;
    procedure IdAm;
    procedure DataAnalise;
    procedure FatorDiluicao;
    procedure CabParametros;
    procedure Parametros;
    procedure Parametro;
    procedure NomeParametro;
    procedure ValorMedido;
    procedure Unidade;
    procedure Numerico;
  public
    { Public declarations }
    procedure Sintatico;
    constructor Create;
  end;

implementation

uses uLexico, principal;

constructor TGCMS01.Create;
begin
  inherited Create;
end;

procedure TGCMS01.Sintatico;
begin
  Gramatica;

  if FNumeroErros = 0 then
    frmPrincipal.EscreveMensagem('Analise sintática terminada com sucesso')
  else
    frmPrincipal.EscreveMensagem(
      'Sintaxe não reconhecida. ' + IntToStr(FNumeroErros) +
      ' erros encontrados');
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <amostra> ::= <cab_amostra> <cab_parametros> <parametros>                 //
//---------------------------------------------------------------------------//
procedure TGCMS01.Gramatica;
const
  NmRegra = '<amostra>';
begin
  FNumeroErros := 0;
  PegaProximoToken;
```

```
  CabAmostra;
  CabParametros;
  Parametros;
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <cab_amostra> ::= <dados> <id_amostra>                                  //
//                   <data_analise> <fator_diluicao> <dados>               //
//-------------------------------------------------------------------------//
procedure TGCMS01.CabAmostra;
const
  NmRegra = '<cab_amostra>';
begin
  Dados;
  IdAmostra;
  DataAnalise;
  FatorDiluicao;
  Dados;
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <dados> ::= <dado> <dados>                                              //
//         |   <dado>                                                      //
//-------------------------------------------------------------------------//
procedure TGCMS01.Dados;
const
  NmRegra = '<dados>';
begin
  Dado;

  if TipoToken in [T_ID, T_ARQUIVO, T_PONTO, T_DOISPONTOS] then
    begin
      Dados;
    end;
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <dado> ::= <id>                                                         //
//        |   <arquivo>                                                    //
//        |   '.'                                                          //
//        |   ':'                                                          //
//-------------------------------------------------------------------------//
procedure TGCMS01.Dado;
const
  NmRegra = '<dado>';
begin
  if TipoToken in [T_ID, T_ARQUIVO, T_PONTO, T_DOISPONTOS] then
    PegaProximoToken
  else
    frmPrincipal.EscreveMensagemTokenEsperado(
      '<id>, <arquivo>, ''.'' ou '':''', NmRegra);
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <id_amostra> ::= 'Sample' 'Name' ':' <id_am>                            //
//-------------------------------------------------------------------------//
procedure TGCMS01.IdAmostra;
const
  NmRegra = '<id_amostra>';
begin
  if TipoToken = T_SAMPLE then
    begin
      PegaProximoToken;
      if TipoToken = T_NAME then
```

```
        begin
          PegaProximoToken;
          if TipoToken = T_DOISPONTOS then
            begin
              PegaProximoToken;
              IdAm;
            end
          else
            frmPrincipal.EscreveMensagemTokenEsperado(':', NmRegra);
        end
      else
        frmPrincipal.EscreveMensagemTokenEsperado('Name', NmRegra);
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Sample', NmRegra);
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <id_am> ::= <id>                                                          //
//         |   <inteiro>                                                     //
//---------------------------------------------------------------------------//
procedure TGCMS01.IdAm;
const
  NmRegra = '<id_am>';
begin
  if TipoToken in [T_ID, T_INTEIRO] then
    begin
      frmPrincipal.mmSaida.Lines.Add('Amostra' + #9 + Token);
      PegaProximoToken;
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('<id> ou <inteiro>', NmRegra);
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <data_analise> ::= 'Acquisition' 'Date' ':' <data> <hora>                 //
//---------------------------------------------------------------------------//
procedure TGCMS01.DataAnalise;
const
  NmRegra = '<data_analise>';
begin
  if TipoToken = T_ACQUISITION then
    begin
      PegaProximoToken;
        if TipoToken = T_DATE then
          begin
            PegaProximoToken;
              if TipoToken = T_DOISPONTOS then
                begin
                  PegaProximoToken;
                  if TipoToken = T_DATA then
                    begin
                      frmPrincipal.mmSaida.Lines.Add('Data' + #9 + Token);
                      PegaProximoToken;
                        if TipoToken = T_HORA then
                          begin
                            frmPrincipal.mmSaida.Lines.Add('Hora' + #9 + Token);
                            PegaProximoToken;
                          end
                        else
                          frmPrincipal.EscreveMensagemTokenEsperado(
                            '<hora>', NmRegra);
                    end
                  else
                    frmPrincipal.EscreveMensagemTokenEsperado(
                      '<data>', NmRegra);
```

```
              end
            else
              frmPrincipal.EscreveMensagemTokenEsperado(':', NmRegra);
          end
        else
          frmPrincipal.EscreveMensagemTokenEsperado('Date', NmRegra);
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Acquisition', NmRegra);
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <fator_diluicao> ::= 'Dilution' 'Factor' ':' <numerico>                   //
//---------------------------------------------------------------------------//
procedure TGCMS01.FatorDiluicao;
const
  NmRegra = '<fator_diluicao>';
begin
  if TipoToken = T_DILUTION then
    begin
      PegaProximoToken;
      if TipoToken = T_FACTOR then
        begin
          PegaProximoToken;
          if TipoToken = T_DOISPONTOS then
            begin
              PegaProximoToken;
              frmPrincipal.mmSaida.Lines.Add('Diluicao' + #9 + Token);
              Numerico;
            end
          else
            frmPrincipal.EscreveMensagemTokenEsperado(':', NmRegra);
        end
      else
        frmPrincipal.EscreveMensagemTokenEsperado('Factor', NmRegra);
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Dilution', NmRegra);
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <cab_parametros> ::= 'Name' 'Calculated' 'Amount' 'Units'                 //
//---------------------------------------------------------------------------//
procedure TGCMS01.CabParametros;
const
  NmRegra = '<cab_parametros>';
begin
  if TipoToken = T_NAME then
    begin
      PegaProximoToken;
      if TipoToken = T_CALCULATED then
        begin
          PegaProximoToken;
          if TipoToken = T_AMOUNT then
            begin
              PegaProximoToken;
              if TipoToken = T_UNITS then
                begin
                  PegaProximoToken;
                end
              else
                frmPrincipal.EscreveMensagemTokenEsperado('Units', NmRegra);
            end
          else
            frmPrincipal.EscreveMensagemTokenEsperado('Amount', NmRegra);
        end
```

```
      else
        frmPrincipal.EscreveMensagemTokenEsperado('Calculated', NmRegra);
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Name', NmRegra);
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <parametros> ::= <parametro> <parametros>                               //
//               |   <parametro>                                           //
//-------------------------------------------------------------------------//
procedure TGCMS01.Parametros;
begin
  Parametro;

  if TipoToken = T_ID then
    begin
      Parametros;
    end;
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <parametro> ::= <nome_parametro> <valor_medido> <unidade>               //
//-------------------------------------------------------------------------//
procedure TGCMS01.Parametro;
begin
  FSaida := '';
  NomeParametro;
  FSaida := FSaida + #9 + Token;
  ValorMedido;
  FSaida := FSaida + #9 + Token;
  Unidade;
  frmPrincipal.mmSaida.Lines.Add(Trim(FSaida));
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <nome_parametro> ::= <id> <id>                                          //
//                   |   <id>                                              //
//-------------------------------------------------------------------------//
procedure TGCMS01.NomeParametro;
begin
  if TipoToken = T_ID then
    begin
      FSaida := FSaida + ' ' + Token;
      PegaProximoToken;
      NomeParametro;
    end;
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <valor_medido> ::= <numerico>                                           //
//                 |   'N/A'                                               //
//-------------------------------------------------------------------------//
procedure TGCMS01.ValorMedido;
begin
  if TipoToken = T_NA then
    PegaProximoToken
  else
    Numerico;
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <unidade> ::= <id>                                                      //
```

```
//---------------------------------------------------------------------------//
procedure TGCMS01.Unidade;
const
  NmRegra = '<unidade>';
begin
  if TipoToken = T_ID then
    begin
      PegaProximoToken;
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('<id>', NmRegra);
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <numerico> ::= <inteiro>                                                  //
//            |   <fracionario>                                              //
//---------------------------------------------------------------------------//
procedure TGCMS01.Numerico;
const
  NmRegra = '<numerico>';
begin
  if TipoToken in [T_INTEIRO, T_FRACIONARIO] then
    begin
      PegaProximoToken;
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado(
      '<inteiro> ou <fracionario>', NmRegra);
end;

end.
```

Arquivo uGCMS02.pas

```
unit uGCMS02;

interface

uses SysUtils;

type

  TGCMS02 = class
    FSaida: string;
    FNumeroErros: Integer;
  private
    { Private declarations }
    procedure Gramatica;
    procedure CabAmostra;
    procedure Dados;
    procedure Dado;
    procedure IdAmostra;
    procedure IdAm;
    procedure DataAnalise;
    procedure Turno;
    procedure CabParametros;
    procedure Parametros;
    procedure Parametro;
    procedure NomeParametro;
    procedure TipoResultado;
    procedure Quantificacao;
  public
    { Public declarations }
    procedure Sintatico;
    constructor Create;
  end;

implementation

uses uLexico, principal;

constructor TGCMS02.Create;
begin
  inherited Create;
end;

procedure TGCMS02.Sintatico;
begin
  Gramatica;

  if FNumeroErros = 0 then
    frmPrincipal.EscreveMensagem('Analise sintática terminada com sucesso')
  else
    frmPrincipal.EscreveMensagem(
      'Sintaxe não reconhecida. ' + IntToStr(FNumeroErros) +
      ' erros encontrados');
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <cab_amostra> ::= 'Sample' <dados> <id_amostra>                           //
//                   <dados> <data_analise> <dados>                          //
//---------------------------------------------------------------------------//
procedure TGCMS02.CabAmostra;
const
  NmRegra = '<cab_amostra>';
begin
  if TipoToken = T_SAMPLE then
    begin
      PegaProximoToken;
```

```
      Dados;
      IdAmostra;
      Dados;
      DataAnalise;
      Dados;
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Sample', NmRegra);
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <cab_parametros> ::= 'Peaks' ':' <inteiro> <entre_par>                    //
//---------------------------------------------------------------------------//
procedure TGCMS02.CabParametros;
const
  NmRegra = '<cab_parametros>';
begin
  if TipoToken = T_PEAKS then
    begin
      PegaProximoToken;
      if TipoToken = T_DOISPONTOS then
        begin
          PegaProximoToken;
          if TipoToken = T_INTEIRO then
            begin
              PegaProximoToken;
              if TipoToken = T_ENTREPAR then
                begin
                  PegaProximoToken;
                end
              else
                frmPrincipal.EscreveMensagemTokenEsperado(
                  '<entre_par>', NmRegra);
            end
          else
            frmPrincipal.EscreveMensagemTokenEsperado('<inteiro>', NmRegra);
        end
      else
        frmPrincipal.EscreveMensagemTokenEsperado(':', NmRegra);
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Peaks', NmRegra);
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <dado> ::= <id>                                                           //
//        |   <data>                                                         //
//        |   <hora>                                                         //
//        |   <tracos>                                                       //
//        |   <arquivo>                                                      //
//        |   <entre_par>                                                    //
//        |   'Date'                                                         //
//        |   'AM'                                                          //
//        |   'PM'                                                          //
//        |   '.'                                                           //
//        |   ':'                                                           //
//---------------------------------------------------------------------------//
procedure TGCMS02.Dado;
const
  NmRegra = '<dado>';
begin
  if TipoToken in [
       T_ID, T_TXTID, T_TRACOSIMPLES, T_ARQUIVO, T_ENTREPAR, T_PONTO,
       T_DOISPONTOS, T_DATE, T_DATA, T_HORA, T_AM, T_PM] then
    PegaProximoToken
  else
```

```
    frmPrincipal.EscreveMensagemTokenEsperado(
      '<id>, <data>, <hora>, <tracos>, <arquivo>, <entre_par>, ' +
      'ID, AM, PM, Date, ''.'' ou '':''', NmRegra);
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <dados> ::= <dado> <dados>                                              //
//         |   <dado>                                                      //
//-------------------------------------------------------------------------//
procedure TGCMS02.Dados;
const
  NmRegra = '<dados>';
begin
  Dado;

  if TipoToken in [
       T_ID, T_TXTID, T_TRACOSIMPLES, T_ARQUIVO, T_ENTREPAR, T_PONTO,
       T_DOISPONTOS, T_DATE, T_DATA, T_HORA, T_AM, T_PM] then
    begin
      Dados;
    end;
end;

//-------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                   //
// <data_analise> ::= 'Acquisition' 'Date' ':' <data> <hora> <turno>       //
//-------------------------------------------------------------------------//
procedure TGCMS02.DataAnalise;
const
  NmRegra = '<data_analise>';
begin
  if TipoToken = T_ACQUISITION then
    begin
      PegaProximoToken;
        if TipoToken = T_DATE then
          begin
            PegaProximoToken;
              if TipoToken = T_DOISPONTOS then
                begin
                  PegaProximoToken;
                  if TipoToken = T_DATA then
                    begin
                      frmPrincipal.mmSaida.Lines.Add('Data' + #9 + Token);
                      PegaProximoToken;
                        if TipoToken = T_HORA then
                          begin
                            frmPrincipal.mmSaida.Lines.Add('Hora' + #9 + Token);
                            PegaProximoToken;
                            Turno;
                          end
                        else
                          frmPrincipal.EscreveMensagemTokenEsperado(
                            '<hora>', NmRegra);
                    end
                  else
                    frmPrincipal.EscreveMensagemTokenEsperado(
                      '<data>', NmRegra);
                end
              else
                frmPrincipal.EscreveMensagemTokenEsperado(':', NmRegra);
          end
        else
          frmPrincipal.EscreveMensagemTokenEsperado('Date', NmRegra);
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Acquisition', NmRegra);
end;
```

```
//-----------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                       //
// <amostra> ::= <cab_amostra> <cab_parametros> <parametros> <tracos>          //
//-----------------------------------------------------------------------------//
procedure TGCMS02.Gramatica;
const
  NmRegra = '<amostra>';
begin
  FNumeroErros := 0;
  PegaProximoToken;

  CabAmostra;
  CabParametros;
  Parametros;

  if TipoToken = T_TRACOSIMPLES then
    begin
      PegaProximoToken;
      if TipoToken = T_ID then
        begin
          PegaProximoToken;
          if TipoToken = T_PEAKS then
            begin
              PegaProximoToken;
                if TipoToken = T_TRACOSIMPLES then
                  begin
                    PegaProximoToken;
                    if TipoToken = T_ID then
                      begin
                        PegaProximoToken;
                      end
                    else
                      frmPrincipal.EscreveMensagemTokenEsperado(
                        '<id>', NmRegra);
                  end
                else
                  frmPrincipal.EscreveMensagemTokenEsperado(
                    '<tracos>', NmRegra);
            end
          else
            frmPrincipal.EscreveMensagemTokenEsperado('Peaks', NmRegra);
        end
      else
        frmPrincipal.EscreveMensagemTokenEsperado('<id>', NmRegra);
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('<tracos>', NmRegra);
end;

//-----------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                       //
// <id_am> ::= <id>                                                            //
//         |   <inteiro>                                                      //
//-----------------------------------------------------------------------------//
procedure TGCMS02.IdAm;
const
  NmRegra = '<id_am>';
begin
  if TipoToken in [T_ID, T_INTEIRO] then
    begin
      frmPrincipal.mmSaida.Lines.Add('Amostra' + #9 + Token);
      PegaProximoToken;
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('<id> ou <inteiro>', NmRegra);
end;
```

```
//-----------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                       //
// <id_amostra> ::= 'Sample' 'ID' ':' <id_am>                                  //
//-----------------------------------------------------------------------------//
procedure TGCMS02.IdAmostra;
const
  NmRegra = '<id_amostra>';
begin
  if TipoToken = T_SAMPLE then
    begin
      PegaProximoToken;
      if TipoToken = T_TXTID then
        begin
          PegaProximoToken;
          if TipoToken = T_DOISPONTOS then
            begin
              PegaProximoToken;
              IdAm;
            end
          else
            frmPrincipal.EscreveMensagemTokenEsperado(':', NmRegra);
        end
      else
        frmPrincipal.EscreveMensagemTokenEsperado('ID', NmRegra);
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Sample', NmRegra);
end;

//-----------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                       //
// <nome_parametro> ::= <id> <id>                                              //
//                    |   <id>                                                 //
//-----------------------------------------------------------------------------//
procedure TGCMS02.NomeParametro;
begin
  if TipoToken = T_ID then
    begin
      FSaida := FSaida + ' ' + Token;
      PegaProximoToken;
      NomeParametro;
    end;
end;

//-----------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                       //
// <parametro> ::= <inteiro> <fracionario> <nome_parametro>                    //
//                 <tipo_res> <quant> <inteiro> <fracionario> <id>             //
//-----------------------------------------------------------------------------//
procedure TGCMS02.Parametro;
const
  NmRegra = '<parametro>';
begin
  if TipoToken = T_INTEIRO then
    begin
      PegaProximoToken;
      if TipoToken = T_FRACIONARIO then
        begin
          PegaProximoToken;
          FSaida := '';
          NomeParametro;
          TipoResultado;
          Quantificacao;
          if TipoToken = T_INTEIRO then
            begin
              PegaProximoToken;
              if TipoToken = T_FRACIONARIO then
                begin
```

```
                FSaida := FSaida + #9 + Token;
                PegaProximoToken;
                if TipoToken = T_ID then
                  begin
                    FSaida := FSaida + #9 + Token;
                    frmPrincipal.mmSaida.Lines.Add(Trim(FSaida));
                    PegaProximoToken;
                  end
                else
                  frmPrincipal.EscreveMensagemTokenEsperado('<id>', NmRegra);
              end
            else
              frmPrincipal.EscreveMensagemTokenEsperado(
                '<inteiro>', NmRegra);
          end
        else
          frmPrincipal.EscreveMensagemTokenEsperado('<fracionario>', NmRegra);
      end
    else
      frmPrincipal.EscreveMensagemTokenEsperado('<fracionario>', NmRegra);
  end
else
  frmPrincipal.EscreveMensagemTokenEsperado('<inteiro>', NmRegra);
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <parametros> ::= <parametro> <inteiro> <parametros>                       //
//              |   <parametro>                                              //
//---------------------------------------------------------------------------//
procedure TGCMS02.Parametros;
const
  NmRegra = '<parametros>';
begin
  Parametro;

  if TipoToken = T_INTEIRO then
    begin
      Parametros;
    end;
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <quant> ::= <id>                                                          //
//         |   <inteiro>                                                    //
//         |   <fracionario>                                                //
//---------------------------------------------------------------------------//
procedure TGCMS02.Quantificacao;
const
  NmRegra = '<quant>';
begin
  if TipoToken in [T_ID, T_INTEIRO, T_FRACIONARIO] then
    begin
      PegaProximoToken;
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('<id>, <inteiro> ou <fracionario>',
NmRegra);
end;

//---------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                     //
// <tipo_res> ::= 'Id' '.'                                                   //
//            |   'Miss' '.'                                                 //
//            |   'Fail'                                                     //
//---------------------------------------------------------------------------//
procedure TGCMS02.TipoResultado;
```

```
const
  NmRegra = '<tipo_res>';
begin
  if TipoToken in[T_TXTID, T_MISS, T_FAIL] then
    if TipoToken in[T_TXTID, T_MISS] then
      begin
        PegaProximoToken;
        if TipoToken = T_PONTO then
          begin
            PegaProximoToken;
          end
        else
          frmPrincipal.EscreveMensagemTokenEsperado('''.''', NmRegra);
      end
    else
      PegaProximoToken
  else
    frmPrincipal.EscreveMensagemTokenEsperado('Id, Miss ou Fail', NmRegra);
end;

//-----------------------------------------------------------------------------//
// Regra esperada para esta procedure é:                                       //
// <turno> ::= 'AM'                                                            //
//           |   'PM'                                                          //
//-----------------------------------------------------------------------------//
procedure TGCMS02.Turno;
const
  NmRegra = '<turno>';
begin
  if TipoToken in[T_AM, T_PM] then
    begin
      frmPrincipal.mmSaida.Lines.Add('Turno' + #9 + Token);
      PegaProximoToken;
    end
  else
    frmPrincipal.EscreveMensagemTokenEsperado('AM ou PM', NmRegra);
end;

end.
```