Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

SCC0206 - Introdução à Compilação
Professor Thiago S. Pardo

LALG

Uma implementação simplificada de Pascal

Parte I

Especificações da Linguagem e Implementação do Analisador Léxico

Projeto de Curso Desenvolvido pelos Alunos

Ubiratan F. Soares (5634292)
Ulisses F. Soares (5377365)
Rafael Pillon Almeida (5634775)

São Carlos, 10 de abril de 2012

Introdução

LALG é uma linguagem de programação derivada do Pascal, proposta como projeto prático da disciplina de Introdução à Compilação. Trata-se, portanto, de uma linguagem procedimental, dotada de um conjunto mínimo de funcionalidades que torne possível a sua utilização enquanto linguagem de alto nível e viável a sua implementação em um curso de um semestre.

A linguagem LALG possui em sua forma mínima a definição de programa e de bloco, além da gramática de expressões e gramática de atributos herdadas da linguagem Pascal. Além disso, também vêm do Pascal a definição de procedimentos, bem como a passagem e lista de parâmetros dos mesmos. Não há o suporte há funções na proposta minimalista da linguagem.

Demais aspectos da linguagem foram decididos pelo grupo e serão apresentados na sequência.

Especificações Gerais do Compilador

Tratamos a seguir as especificações gerais do Compilador da linguagem *LALG*. Essa especificação está de acordo com a proposta da docente, sendo subdividida nos itens que se seguem.

1) Sobre a Finalidade do Compilador

O Compilador a ser desenvolvido se destina a compilar programas escrito na linguagem **LALG** a ser implementada em suas diversas etapas ao longo da disciplina de Introdução à Compilação. Assim, o compilador será desenvolvido com a finalidade didática, para de modo a apresentar aos alunos os conceitos da área de Compiladores e ensinar técnicas que abordem as principais fases processo de tradução.

2) Sobre os Requisitos de Velocidade do Compilador

A eficiência do processo de compilação não é o foco da implementação em questão. Dessa forma, heurísticas de *parsing* eficiente, uso de estruturas de dados auxiliares otimizadas, algoritmos para geração de representações intermediárias eficientes e outros não serão explorados nesse trabalho.

A finalidade do projeto proposto com a LALG é de construir um software de intuito didático, com tempo de desenvolvimento adequado. Assim, serão utilizadas ferramentas para automatizar partes do processo, ficando em segundo plano o compromisso com desempenho. Além disso, a intenção é que o compilador gere código transportável, uma vez que a linguagem-objeto pode eventualmente ser transportada para alguma máquina de execução hipotética.

Finalmente, deve-se destacar que o número de passos que será utilizado no processo de tradução é um item que de certa forma agrega eficiência ao processo a ser implementado.

3) Sobre os Requisitos de Velocidade do Código-Objeto

Sendo esse um projeto de caráter didático, não pretende-se alcançar requisitos de desempenho do código-objeto gerado pelo Compilador *LALG.* Assim, a geração de código se dará para a linguagem de máquina qualquer, sem compromissos de desempenho para execução.

4) Sobre a Máquina Hospedeira do Compilador

O compilador a ser implementado para essa linguagem é do tipo cruzado (*Cross Compiler*), de maneira que o hardware de execução representado pela máquina virtual onde residirá o código-objeto é distinto do ambiente onde reside o tradutor. Mais detalhes no item 6.

5) Sobre o Software de Sistema para a Execução de Programas

O ambiente de sistema para a execução do compilador *LALG* é multi-plataforma. A implementação usará, dentre outras ferramentas, o *compiler compiler* **JavaCC**, que executa em ambiente **Java** e gera código nessa mesma linguagem. Dessa forma, qualquer sistema que possua compatibilidade com a versão ***1.6_x*** do ***JRE (Java Runtime Engine)*** ou outra versão mais atual poderá compilar programas na linguagem *LALG*.

Além disso, assume-se que esse Compilador estará acessível por interface de linha de comando e aceitará como entrada arquivos de dados na extensão ***.pas*** que identificarão os códigos-fonte e produzirá o código-objeto apropriado.

6) Sobre o Ambiente de Execução da Máquina-Objeto

Ainda não está definido o ambiente de execução dos código-objeto gerado pelo compilador da *LALG*.

7) Sobre o Grau de Complexidade da Linguagem-Fonte

A linguagem *LALG* é simples do ponto de vista da complexidade. Além de suportar somente dois tipos nativos de dados, três instruções de fluxo de controle, não suportar ponteiros e funções, ela é completamente procedimental, como as linguagem-mãe Pascal. Além disso, as rotinas de entrada e saída são tratadas como nativas da linguagem, de maneira que não é necessário a tradução de um comando de ES a ser interceptado por uma biblioteca ou rotina do sistema.

Por esses e outros aspectos, a *LALG* pode ser considerada de complexidade simples.

8) Sobre o Número de Passos no Processo de Compilação

O compilador será implementado com um tradutor de um passo. Dessa forma, o código-objeto será gerado conforme o código-fonte em *LALG* for processado pelo compilador, sem retrocessos para otimizações. Dessa forma, o processo de tradução ganha em eficiência, ainda que não sejam introduzidos ganhos de performance com otimizações globais ou locais, uma vez que as fases intermediárias são entrelaçadas. Essa abordagem será dirigida por sintaxe[1].

9) Sobre as Técnicas de Implementação

Para auxiliar no desenvolvimento desse trabalho, conforme comentado no item 6, foi adotada a ferramenta de *compiler compiler* **JavaCC**. Essa ferramenta oferece suporte à duas fases do processo de compilação : a análise léxica (*tokenizer*) e a análise sintática (*parser*).

Uma vez que a ferramenta JavaCC gera código de compilador em linguagem Java, o restante do compilador *LALG* também será implementado nessa mesma linguagem. A especificação léxica se dá segundo expressões reguladores estendidas, de acordo com a entrada padrão JavaCC.

Por outro lado, a Análise Sintática será feita de forma descendente recursiva, de forma que a gramática da linguagem *LALG* descrita em EBNF ou alguma variante de notação de gramáticas pode ser facilmente adaptada para uma especificação aceitável por esse *compiler compiler*.

A implementação dos demais passos do processo de tradução (Análise Semântica, Geração de Código e afins) será decidida ao longo do desenvolvimento desse trabalho.

Especificações Léxicas

Para a implementação do analisador léxico, primeira etapa do processo de compilação, seguem-se algumas considerações.

Conjunto de Palavras Reservadas

As palavras reservadas são os símbolos imediatos a serem reconhecidos na elaboração de qualquer linguagem de programação, de maneira que a LALG possui um conjunto de poucas palavras reservadas. Na etapa de análise léxica agora implementada, não há distinção entre essas palavras e identificadores bem formadas, mas essa poderá ser incorporada no futuro mediante vantagens de inserção antecipada na tabela de palavras reservadas.

A tabela de palavras reservadas da LALG segue abaixo :

Palavras Reservadas	program	procedure	var
integer	real	begin	end
if	then	else	while
do	repeat	until	const

Constantes Permitidas

Como a *LALG* só admite dois tipos primitivos de dados, só são permitidas atribuições constantes reais e inteiras. Os inteiros serão de 16 bits, o que implica no maior inteiro valendo **+32767** e o menor assumindo **-32768**. Números além desse escopo não serão reconhecidos como válidos pelo *tokenizer*.

Já para os números reais, os valores em 16 bits correspondem ao delineados pelo padrão **IEEE 754**, com o maior valor correpondendo a **+3.4e38** e o menor a **-3.4e38.**

Caracteres Não-Imprimíveis e Símbolos Especiais

Os caracteres **"/n"**, **"/t"**, **"LF"**, **"CR"** e outros de controle de linhas utilizados em editores de texto ASCII não serão imprimidos na etapa de análise léxica. Além disso, são tratados como símbolos especiais os seguinte caracteres listado na tabela a seguir. A LALG não suporta caracteres fora da codificação ASCII.

.	**;**	**>**	**>=**	**=**
<	**<=**	**:=**	**{**	**}**
+	**-**	*****	**,**	

Tipos e Tratamento dos Comentários

A *LALG* suporta comentários multi-linha. Um trecho comentado pode ser obtido iniciando o excerto com os delimitadores **"{"** e **"}"** , assim como na linguagem Pascal.

A implementação do analisador léxico permite que a *LALG* tenha suporte a comentários aninhados. Exemplos de uso desse tipos de comentário e sua validação pelo *tokenizer* estão nos arquivos de implementação entregues no email da disciplina.

Tamanho Máximo e Formação dos Identificadores

A *LALG* suporta identificadores formados por até 32 caracteres. Esses caracteres podem ser letras minúsculas ou maiúsculas e números, começando sempre por uma letra. O analisador léxico implementado é capaz de invalidar identificadores que utilizem caracteres fora desse cojunto (por exemplo, **"@"**), mas não é capaz de invalidar um identificador mal formado no caso de começar por número (por exemplo, **1a**).

Por fim, a *LALG* será implementada como *case sensitive*, de maneira que nenhum tratamento em relação ao uso de caixa alta ou baixa foi implementado em nível léxico.

Comentários sobre a Implementação do Analisador Léxico

Para a implementação do analisador léxico foi utilizada a versão mais recente do JavaCC (**javacc-5.0**), além do ambiente **Java 1.6.0_31**.

No intuito de auxiliar a correção dessa etapa do projeto, foram desenvolvidos *scripts* de ambiente UNIX para compilação e execução automática de testes.

Para a compilação e geração do arquivo **.jar** executável em Java, foi desenvolvido um script com base na ferramenta Apache Ant. As instruções são detalhadas a seguir.

Configurando o Apache ANT

Caso não possua o Apache ANT em seu sistema, navegue pelo terminal até a raiz do diretório **lalg-lexical** e coloque o executável do ANT no seu ambiente (exige JRE 1.6+) :

- Linux / Unix / Mac OSX

```
-> export PATH=./lib/apache-ant-1.8.2/bin:$PATH
```

Usuários das versões Mac OSX 10.6 e 10.7 já possuem o Apache ANT instalado caso tenham o módulo "*Developer Tools*" em seu ambiente. Caso não possuam, o procedimento é o mesmo descrito acima.

Executando as tarefas

A partir da raiz de ***lalg-lexical***, os seguintes comandos são válidos

Menu de opções

```
-> ant
```

Gerar os arquivos com JavaCC

```
-> ant parser
```

Esses arquivos serão colocados em *lalg-lexical/src/javacc-generated*

Gerar os arquivos Java compilados

```
-> ant build
```

Esses arquivos serão colocados em *lalg-lexical/build*.

Gerar o arquivo .jar executável do compilador

```
-> ant jar
```

O executável será colocado em *lalg-lexical/bin*.

Limpar arquivos compilados, classes e diretórios criados

```
-> ant clean
```

Executando testes em bateria

Para executar os testes em bateria, no diretório *lalg-lexical/tests* existe um *script* Shell capaz de ler todos os arquivos de extensão **.pas** localizados no diretório *lalg-lexical/tests/samples* e utilizar o binário localizado previamente gerado em *lalg-lexical/bin* para executar o analisador léxico.

Os resultados serão um arquivo de texto simples para cada arquivo de código-fonte, com os símbolos reconhecidos e eventuais apontamentos de erros léxicos que estarão localizados em *lalg-lexical/tests/results.*

Referências

1. **Thiago Salgueiro Pardo** - Notas de Aula. São Carlos, 2012.
2. **T. Kowaltowski** - *Implementação de Linguagens de Programação*. Guanabara Dois, 1983
3. **A. V. Aho, M. S. Lam, J. D. Ullman** - *Compilers: Principles, Techniques, and Tools (2nd edition).* Addison Wesley, 2006
4. **Thiago S. Pardo** - *Notas de Aula*. São Carlos, 2010.
5. **A. W. Appel and J. Palsberg** - *Modern Compiler Implementation in Java (2nd edition)*. Cambridge University Press, 2002