Teoria e Implementação de Linguagens Computacionais

Especificação do projeto da disciplina (2006.1)

Versão de 23/08/2006

André Santos

Introdução

O objetivo do projeto é proporcionar aos alunos da disciplina de *Teoria e Implementação de Linguagens de Computacionais* a experiência de criar um compilador completo, compreendendo as fases mais comuns do processo de compilação: análise léxica/sintática, análise semântica e geração de código.

A descrição abaixo propõe um projeto a ser implementado na disciplina.

Descrição

O projeto consiste na criação do analisador léxico, sintático e semântico da linguagem abaixo, mini-pascal. Além disso, os alunos deverão construir um gerador de código para tal linguagem, tendo como alvo a JVM (Java Virtual Machine) ou a IL (intermediate language) do ambiente .NET da Microsoft.

Entregas e Avaliação

O projeto deverá ser entregue conforme o calendário da disciplina.

O projeto será avaliado conforme a tabela abaixo:

Análise Léxica e Sintática	3,0
Análise Semântica (tipos e escopo)	3,0
Geração de Código	4,0
Total	10,0

Printed with FinePrint - purchase at www.fineprint.com

Convenções

A equipe deverá entregar *todos os fontes do projeto* assim como quaisquer *arquivos de especificação das ferramentas utilizadas* para o monitor responsável por esta. Todo este conteúdo deverá estar compactado em um arquivo .zip e enviado por email. Além de todos os fontes e arquivos de especificação adicionais, deverão estar presentes no .zip os seguintes itens:

- *readme.txt* : com instruções detalhadas de como instalar/compilar o projeto. É importante que estas instruções sejam claras e fáceis de seguir, pois se o monitor não conseguir instalar/compilar o projeto, ele não poderá dar uma nota a equipe.

- *releaseNotes.txt* : contendo a descrição detalhada do que a atual versão do projeto suporta ou não suporta, assim como variações da especificação original. Esse arquivo será útil na correção do projeto, pois através dele as deficiências da versão a ser corrigida poderão ser analisadas isoladamente, já que um *bug* quando reportado no *releaseNotes.txt* ajudará o monitor a corrigir o projeto de forma mais justa, analisando o problema isoladamente e evitando que tal deficiência impacte sobre outra parte do projeto.

OBSERVAÇÕES SOBRE A ENTREGA DO PROJETO

- O projeto deve estar em um arquivo zip, de modo que quando se usar o "extract here" no windows explorer o compilador esteja no diretório projeto_if688/src/projeto/Compilador.java

- Para rodar o compilador deve ser executado o comando:

java projeto.Compilador NomeDoArquivo DiretórioSaida

(esse diretório saída só vai ser útil na geração de código, ele vai conter o assembler gerado)

Elementos SINTÁTICOS

Notas sobre a Notação

Os colchetes '[' ']' indicam que o elemento sintático é opcional.

Ex: No elemento que representa as seções de declaração DeclSection, nos informa que o usuário pode não declarar variável alguma.

O '*' indica que existe ZERO ou MAIS declarações do elemento que o antecede.

O '+' indica que existe UM ou MAIS declarações do elemento que o antecede.

Printed with FinePrint - purchase at www.fineprint.com

O '|' indica que APENAS um dos elementos vai ser aplicado a regra.

Sintaxe

Program -> [**VarSection**] **FunctionDecl***

Block -> '{' **StmtList** '}'

Type -> CHAR | INT | BOOL | STRING | VOID

VarSection -> VAR (**VarDecl**)+

VarDecl -> **Type IdentList** ';'

IdentList -> IDENT (**Seq_IdentList**)*

Seq_IdentList -> ',' IDENT

Expression -> **SimpleExpression ComplexExpression***

ComplexExpression -> **RelOp SimpleExpression**

SimpleExpression -> **Term ComplexTerm***

ComplexTerm -> **AddOp Term**

Term -> **Factor ComplexFactor***

ComplexFactor -> **MulOp Factor**

Factor -> IDENT '(' [**Expression**] ')'
-> IDENT
-> INT_LITERAL
-> CHAR_LITERAL
-> STRING_LITERAL
-> BOOLEAN_LITERAL
-> '(' **Expression** ')'

RelOp -> '>' | '<' | '==' | ' !='

Printed with FinePrint - purchase at www.fineprint.com

AddOp -> '+' | '-' | OR

MulOp -> '*' | '/' | AND

Statement -> IDENT '(' [**Expression**] ')'

-> IDENT '=' **Expression**

-> PRINTLN '(' **Expression** ')'

-> PRINT '(' **Expression** ')'

-> '{' **StmtList** '}'

-> IF **Expression Statement** ELSE **Statement**

-> WHILE **Expression Statement**

-> FOR '(' **Statement** ';' **Statement** ';' **Statement** ')' **Statement**

-> RETURN **Expression**

StmtList -> **SimpleStatement***

SimpleStatement -> **Statement** ';'

FunctionDecl -> **FunctionHeading Block**

FunctionHeading -> FUNCTION **Type** IDENT '(' [**FormalParam**] ')'

FormalParm -> **Type** IDENT

Observações

A linguagem é case insensitive!

São considerados tokens todos os elementos acima que contém todas letras maiúsculas ou está entre aspas simples. Ex: IF é token, ')' é o token fecha parênteses.

IDENT -> Contém Letras maiúsculas e minúsculas, e caractere underscore '_'. Contém números também, mas estes não podem iniciar o token.

STRING_LITERAL -> Conjunto de caracteres entre aspas simples (é diferente de java que tem uma string denotada com aspas duplas).

Printed with FinePrint - purchase at www.fineprint.com

CHAR_LITERAL -> um caracter entre aspas simples.

PRINT -> ‘Print’, Recebe uma String e imprime na tela

PRINTLN -> ‘Println’, Recebe uma String e imprime na tela colocando \n depois.

INT_LITERAL -> Inteiro literal

Printed with FinePrint - purchase at www.fineprint.com