Universidade Federal de Campina Grande – UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Sistemas e Computação – DSC

Disciplina: Compiladores

Professora: Franklin Ramalho

Relatório da Fase 1 do Projeto de Compilador para a linguagem OurJava

Equipe

Adalberto Teixeira

Andrey Menezes

Augusto Queiroz

Campina Grande-PB, Setembro/2011

CONTEÚDO

INTRODUÇÃO

Nesse relatório descrevemos os resultados e o caminho seguido por nossa equipe durante a fase inicial da criação do compilador para a linguagem *OurJava*, uma linguagem baseada em Java mas com reduzida expressividade e abrangência.

OurJava é uma linguagem composta pelas seguintes estruturas da linguagem Java original[1]:

- Funções e procedimentos
- Comandos condicionais
- Comandos iterativos
- Comando de atribuição
- Expressões aritméticas e booleanas
- Expressões relacionais
- Literais
- Arranjos
- Chamada de funções e procedimentos
- Seqüenciadores de escape

Trivialmente, percebe-se que toda a parte de Orientação a Objetos foi retirada da linguagem Java Original, como também lançamento e tratamento de exceções e concorrência.

ANALISADOR LÉXICO

O analisador léxico foi inicialmente desenvolvido com alguns padrões simples, mas só após termos a gramática bem definida pudemos iterativamente melhorá-la e completá-la.

Logo abaixo expomos o código completo e final do analisador léxico na notação da ferramenta Flex[2].

O cabeçalho em comentários:

```
/*
** Projeto Compiladores - 2011.2 **
** UFCG - DSC                   **
** Prof.: Franklin Ramalho      **
** Equipe:                      **
**     Adalberto Teixeira       **
**     Andrey Menezes                **
**     Augusto Macedo                **
** Ling. Fonte: Java                  **
*/
```

As declarações iniciais:

```
%{
#include "tokens.h"
#include <string.h>
```

[1] Quando falamos de Java Original nos referimos à gramática que está disponível no site: ...
[2] http://www.cs.princeton.edu/~appel/modern/c/software/flex/flex_toc.html

```
extern int column; // importa variavel do sintatico
extern int line; // importa variavel do sintatico
extern char* yytext;
void moveCol();
void moveLine();
%}
```

A definição dos padrões:

```
col_delimit  [ \r\t]
line_delimit [\n]
id              {letter_no_digit}+({letter})*
letter          [A-Za-z_0-9]
letter_no_digit [A-Za-z_]
digit        [0-9]
non_zero_digit      [1-9]
int_literal  {signal}?{non_zero_digit}+{digit}*{int_type_suf}?
int_type_suf [lL]
signal              [+-]
float_literal{signal}?{digit}+.{digit}+{exp_indicator}?{float_sufix}?
double_literal      {signal}?{digit}+.{digit}+{exp_indicator}?{double_sufix}?
exp_indicator[eE]{signal}?{digit}+
float_sufix  [fF]
double_sufix [dDfF]
hex_literal  0{hex_id}{hex_digit}*
hex_id       [xX]
hex_digit    [0-9A-F]
oct_literal  0{oct_digit}*
oct_digit    [0-7]
equal_oper   "=="|"!="
rel_oper     "<"|">"|"<="|">="|"instanceof"
shifts       "<<"|">>"|">>>"
logic_assig  "&="|"^="|"|="
arith_assig  "*="|"/="|"+="|"-="|"%="
shift_assig  "<<="|">>="|">>>="
char_literal ['][^']{1}[']
string_literal      ["][^"]*["]


%%

{col_delimit}+                  {moveCol();}
{line_delimit}                  {moveLine();}
"//".*{line_delimit}            {moveLine();}
"/*"([^*]|[*]+[^/])*[*]+[/]     {moveCol();}
";"                         {moveCol();return PT_VIRGULA;}
"{"                         {moveCol();return BEG;}
"}"                         {moveCol();return END;}
"["                         {moveCol();return OPEN_COLC;}
"]"                         {moveCol();return CLOSE_COLC;}
"("                         {moveCol();return OPEN_PAREN;}
")"                         {moveCol();return CLOSE_PAREN;}
","                         {moveCol();return VIRGULA;}
"."                         {moveCol();return POINT;}
":"                         {moveCol();return TWO_POINTS;}
"+"                         {moveCol();return PLUS;}
"-"                         {moveCol();return MINUS;}
"*"                         {moveCol();return MULT;}
"/"                         {moveCol();return DIV;}
"%"                         {moveCol();return MOD;}
"="                         {moveCol();return EQUAL;}
"?"                         {moveCol();return QUESTION_MARK;}
"||"                        {moveCol();return OR_LOGIC;}
"&&"                        {moveCol();return AND_LOGIC;}
"++"                        {moveCol();return INCREMENT;}
"--"                        {moveCol();return DECREMENT;}
"!"                         {moveCol();return NOT;}
"~"                         {moveCol();return NOT_BIT;}
```

```
"|"                     {moveCol();return OR;}
"^"                     {moveCol();return OR_EXC;}
"&"                     {moveCol();return AND;}
"if"                    {moveCol();return IF;}
"else"                  {moveCol();return ELSE;}
"while"                 {moveCol();return WHILE;}
"new"                   {moveCol();return NEW;}
"do"                    {moveCol();return DO;}
"for"                   {moveCol();return FOR;}
"break"                 {moveCol();return BREAK;}
"continue"              {moveCol();return CONTINUE;}
"goto"                  {moveCol();return GOTO;}
"return"                {moveCol();return RETURN;}
"case"                  {moveCol();return CASE;}
"switch"                {moveCol();return SWITCH;}
"default"               {moveCol();return DEFAULT;}
"transient"             {moveCol();return TRANSIENT;}
"volatile"              {moveCol();return VOLATILE;}
"final"                       {moveCol();return FINAL;}
"class"                       {moveCol();return CLASS;}
"static"                {moveCol();return STATIC;}
"void"                  {moveCol();return VOID;}
"public"                {moveCol();return PUBLIC;}
"main"                  {moveCol();return MAIN;}
"args"                  {moveCol();return ARGS;}
"int"                   {moveCol();return TYPE_INT;}
"short"                       {moveCol();return TYPE_SHORT;}
"long"                  {moveCol();return TYPE_LONG;}
"byte"                  {moveCol();return TYPE_BYTE;}
"boolean"               {moveCol();return TYPE_BOOL;}
"float"                       {moveCol();return TYPE_FLOAT;}
"double"                {moveCol();return TYPE_DOUBLE;}
"char"                  {moveCol();return TYPE_CHAR;}
"String"                {moveCol();return TYPE_STRING;}
"true"                  {moveCol();yylval.strval = strdup(yytext);return
LITERAL;}
"false"                       {moveCol();yylval.strval =
strdup(yytext);return LITERAL;}
"null"                  {moveCol();yylval.strval = strdup(yytext);return
LITERAL;}
{char_literal}                {moveCol();yylval.strval =
strdup(yytext);return LITERAL;}
{string_literal}        {moveCol();yylval.strval = strdup(yytext);return
LITERAL;}
{logic_assig}           {moveCol();yylval.strval = strdup(yytext);return
LOGIC_ASSIGN;}
{shift_assig}           {moveCol();yylval.strval = strdup(yytext);return
SHIFT_ASSIGN;}
{arith_assig}           {moveCol();yylval.strval = strdup(yytext);return
ARITH_ASSIGN;}
{equal_oper}            {moveCol();yylval.strval = strdup(yytext);return
EQUALOP;}
{rel_oper}              {moveCol();yylval.strval = strdup(yytext);return
RELOP;}
{shifts}                {moveCol();yylval.strval = strdup(yytext);return
SHIFTS;}
{int_literal}                 {moveCol();yylval.strval =
strdup(yytext);return LITERAL;}
{hex_literal}                 {moveCol();yylval.strval =
strdup(yytext);return LITERAL;}
{oct_literal}           {moveCol();yylval.strval = strdup(yytext);return
LITERAL;}
{float_literal}         {moveCol();yylval.strval = strdup(yytext);return
LITERAL;}
{double_literal}        {moveCol();yylval.strval = strdup(yytext);return
LITERAL;}
{id}                    {moveCol();yylval.strval = strdup(yytext);return
ID;}
```

```
%%
```

As rotinas utilizadas durante o reconhecimento:

```
int yywrap(){
   return 1;
}


void moveLine(){
   /**segue para proxima linha e reinicia a coluna**/
   line++;
   column = 1;
}

void moveCol(){
   /**segue para proxima coluna**/
   column += yyleng;
}
```

ANALISADOR SINTÁTICO

Para desenvolvermos o analisador sintático realizamos a sequência de atividades a seguir:

Retirada das Recursões à esquerda: inicialmente aplicamos o algoritmo geral visto em sala para retirar todas as recursões detectadas visualmente.

Fatoração: Fatoramos algumas regras também aplicando o algoritmo visto em sala e presente no livro texto.

Checamos a gramática com relação à *Ambigüidade* e vimos que a mesma já tratara esse problema. A questão da *Precedência de Operadores* também estava tratada dado que os níveis de precedência da linguagem já estavam especificados em n+1 regras de produção (para mais detalhes basta ver as regras das Expressions).

Nesse ponto do projeto alcançamos a gramática do OurJava que foi entregue ao professor na atividade de sala. Logo após, partimos para a implementação do compilador utilizando a linguagem reconhecida pela ferramenta *Bison*[3]. Essa etapa foi bem trabalhosa, dado que encontramos várias inconsistências na gramática de Java original e ainda tivemos que adaptá-la para a nova linguagem. Apesar de tudo, aprendemos bastante.

Vários erros e warnings foram reportados ao compilarmos a gramática da forma que foi entregue ao professor, com a ajuda dos arquivos de saída do Bison pudemos observar a evolução da compilação do código do analisador sintático de *OurJava* e depurá-lo.

[3] http://www.gnu.org/s/bison/

Ao término dessa fase, remontamos a planilha com todas as modificações realizadas na gramática de Java para alcançar a gramática de OurJava (ver planilha em anexo).