Universidade de Coimbra
Faculdade de Ciências e Tecnologia

Projecto de Compiladores

Compliador de PJava

Trabalho realizado pelos alunos:
2008114843 Ricardo Pinto Lopes
2008115099 Rui Chicória

Meta 1

Nesta primeira meta, desenvolveu-se o ficheiro lex, a gramática da linguagem, a especificação da sintaxe abstracta, construção da sua árvore e a análise sintática. Parte da linguagem PJava está já implementada, faltando ainda alguns pormenores como o do while, o switch ou a chamada de métodos.

Por opção, deixou-se o terceiro campo do ciclo for a aceitar apenas uma expressão. Isto foge à norma, por ser também possível colocar outros elementos, como prints ou chamadas de funções, e ser possível colocar mais do que uma instrução, separando-as por uma vírgula. Optou-se por simplificar este campo devido à falta de tempo para implementar todas as funcionalidades, e por assim já ser possível captar o essencial da utilização deste ciclo. Também por opção, implementou-se apenas ifs sem elses, devido igualmente a uma questão de gestão de tempo e de procurar explorar o máximo de situações interessantes possíveis face à natureza da linguagem PJava.

Para a obtenção das estruturas para representar a Árvore de Sintaxe Abstracta, recorreu-se ao método usado na ficha 6a. Assim, para cada nova funcionalidade implementada na gramática, criou-se uma nova estrutura no ficheiro structures.h, e as funções correspondentes no ficheiro functions.c. Desta forma, o programa tem uma lista de is_static, que corresponde à lista de todos os atributos e métodos do código em PJava, e cada um desses is_static vai conter atribuições, no caso de se tratarem de atributos, ou argumentos e statements, no caso de se tratarem de métodos.

Eis a gramática actual (com as funções a ser chamadas omitidas, por uma questão de legibilidade):

```
initclass:      CLASS VAR '{' statics '}'
        ;

statics:        statics static
        |       static
        ;

static:         STATIC attribution ';'
        |       STATIC declaration
        |       method
        ;

declaration:    type attributions ';'
        ;
```

```
attributions:   attributions ',' attribution
        |               attribution
        ;

attribution:    VAR
        |       VAR '=' expression
        ;

method:         STATIC type VAR '(' args ')' '{' statements '}'
        |       STATIC type VAR '(' args ')' '{' '}'
        |       STATIC type VAR '(' ')' '{' statements '}'
        |       STATIC type VAR '(' ')' '{' '}'
        ;

args:           args ',' arg
        |       arg
        ;

arg:                    type VAR
        ;

type:           INT
        |       STRING
        |       VOID
        |       FLOAT
        |       DOUBLE
        |       BOOLEAN
        |       CHAR
        ;

statements:     statements statement
        |       statement
        ;

statement:      declaration
        |       attribution ';'
        |       print
        |       if
        |       while
        |       for
        ;


print:                  PRINT '(' expression ')' ';'
```

```
        |       PRINTLN '(' expression ')' ';'
        ;

expression:     infix_expression
        |       unary_expression
        |       NUMBER
        |       VAR
        ;

infix_expression:       expression '+' expression
        |               expression '-' expression
        |               expression '*' expression
        |               expression '/' expression
        |               expression '+"+'
        |               expression '-"-'
        ;

unary_expression:       '-' expression   %prec UMINUS
        ;

b_expression:           b_expression AND b_expression
        |                       b_expression OR b_expression
        |                       '!' b_expression  %prec UMINUS
        |               expression EQUALS expression
        |                       expression DIFERENT expression
        |                       expression GREATER expression
        |               expression LESSER expression
        |               expression GREATEQ expression
        |                       expression LESSEQ expression
        |                       TRUE
        |                       FALSE
        ;

if:             IF '(' b_expression ')' '{' statements '}'
        |       IF '(' b_expression ')' statement
        ;

while:          WHILE '(' b_expression ')' '{' statements '}'
        |       WHILE '(' b_expression ')' statement
        ;

for:            FOR '(' for_first_camp b_expression ';' expression ')' '{' statements '}'
        |       FOR '(' for_first_camp b_expression ';' expression ')' statement
        ;
```

```
for_first_camp:         attributions ';'
        |                       declaration
        ;
```