SELMA

Vonk, J
s0132778
Matenweg 75-201

Florisson, M
s000000
Box Calslaan xx-30

June 27, 2011

Contents

1 Inleiding

Korte beschrijving van de practicumopdracht.

2 Beknopte beschrijving

van de programmeertaal (maximaal een A4-tje).

3 Problemen en oplossingen

uitleg over de wijze waarop je de problemen die je bent tegenge- komen bij het maken van de opdracht hebt opgelost (maximaal twee A4-tjes).

4 Syntax, context-beperkingen en semantiek

4.1 Lexer - terminals

Om de code te kunnen parsen zal deze eerst door de lexer moeten gaan. Hier definieren wij een aantal terminal symbolen. Dit is een eindige set van een aantal symbolen of woorden, de lexer zal deze herkennen. Mits ze in de juiste volgorde worden gebruikt krijg je taalconstructies die de parser vervolgens weer begrijpt. We hebben een aantal speciale terminals die zijn opgebouwd uit meerdere karakters bijvoorbeeld. Deze vormen de lexicon. En een drietal terminals zonder textuele vorm. Deze zijn enkel voor de interne boekhouding van de parser.

```
CHARV           APOSTROPHE LETTER APOSTROPHE;
BOOLEAN         (TRUE | FALSE);
ID              LETTER (LETTER | DIGIT)*;
NUMBER          DIGIT+;

DIGIT           ('0'..'9');
LOWER           ('a'..'z');
UPPER           ('A'..'Z');
LETTER          (LOWER | UPPER);
TRUE            'true';
FALSE           'false';

UMIN;
UPLUS;
COMPOUND;
```

Verder zijn er nog de 'gewone' terminals. Te verdelen in keywords, tokens en operators. Keywords geven aan dat er een bepaalde actie gedaan wordt, zoals een variabele declareren of een if statement. Tokens zijn er om de taal iets meer structuur te geven, denk aan comma's tussen de variabelen. En operators zijn bewerkingen die je kunt uitvoeren op 1 of meer expressies.

Tokens		Keywords	
COLON	':';	PRINT	'print';
SEMICOLON	';';	READ	'read';
LPAREN	'(';	VAR	'var';
RPAREN	')';	CONST	'const';
LCURLY	'{';	INT	'integer';
RCURLY	'}';	BOOL	'boolean';
COMMA	',';	CHAR	'character';
EQ	'=';	BEGIN	'begin';
APOSTROPHE	'''';	END	'end.';
		IF	'if';
		THEN	'then';
		ELSE	'else';
		FI	'fi';
		WHILE	'while';
Operators		DO	'do';
		OD	'od';
NOT	'!';	PROC	'procedure';
MULT	'*';	FUNC	'function';
DIV	'/';		
MOD	'%';		
PLUS	'+';		
MINUS	'-';		
RELS	'<';		
RELSE	'<=';		
RELGE	'>=';		
RELG	'>';		
RELE	'==';		
RELNE	'<>';		
AND	'&&';		
OR	'||';		
BECOMES	':=';		

4.2 Basis

De basis van het programma geeft een aantal restricties op aan de taal. Allereerst is er het programma, dit bestaat uit een (zeer grote) compoundexpression waarna het programma stopt (End Of File). Deze wordt hier herschreven. Een compoundexpression is uiteindelijk opgebouwd uit een serie declaraties en statements, gescheiden door een semicolon. Hier is te zien dat het programma uit minimaal 1 expressie bestaat, dat declaraties en expressies door elkaar gebruikt mogen worden en dat het laatste statement in een programma altijd een expressie is.

```
program
        : compoundexpression EOF
                -> ^(BEGIN compoundexpression END)
        ;

compoundexpression
        : cmp -> ^(COMPOUND cmp)
        ;

cmp
  : ((declaration SEMICOLON!)* expression SEMICOLON!)+
        ;
```

4.3 Declaraties en types

SELMA kent twee soorten declaraties, variabelen en constanten. SELMA staat toe om per declaratie meerdere identifiers te definieren. Bij de declaratie dien je het type van de te declareren waarde mee te geven. En bij een constante dien je uiteraard een waarde mee te geven.

4.3.1 Syntax

```
declaration
        : VAR identifier (COMMA identifier)* COLON type
                -> ^(VAR type identifier)+
        | CONST identifier (COMMA identifier)* COLON type
                EQ unsignedConstant
                -> ^(CONST type unsignedConstant identifier)+
        ;

type
        : INT
        | BOOL
        | CHAR
        ;
```

4.3.2 Context

Het gegeven type dient bij de constante overeen te komen met het type van de gegeven waarde.

Identifiers mogen niet eerder gedeclareerd zijn, in de huidige of bovenliggende scope.

4.3.3 Semantiek

Er zal ruimte gereserveerd worden voor de variabele en het adres wordt onthouden. Voor een constante geldt hetzelfde behalve dat dan ook direct de desbetreffende waarde op dat adres wordt gezet. Op het moment dat elders in het programma een verwijzing is naar deze gedeclareerde dan zal deze variabele of constante geladen worden.

4.3.4 Voorbeeld

```
var i, x: integer;
const c: char = 'g';
const b,t: boolean = true;
```

5 Vertaalregels

voor de taal, d.w.z. de transformaties waaruit blijkt op welke wijze een opeenvolging van symbolen die voldoet aan een produktieregel wordt omgezet in een opeenvol- ging van TAM-instructies. Vertaalregels zijn de code templates van hoofdstuk 7 van Watt & Brown.

6 Beschrijving van Java programmatuur

Beknopte bespreking van de extra Java klassen die u gedefinieerd heeft voor uw compiler (b.v. symbol table management, type checking, code generatie, error handling, etc.). Geef ook aan welke informatie in de AST-nodes opgeslagen wordt.

7 Testplan en -resultaten

Bespreking van de correctheids-tests aan de hand van de criteria zoals deze zijn beschreven in het A.5 van deze appendix. Aan de hand van deze criteria moet een verzameling test-programmas in het taal geschreven worden die de juiste werking van de vertaler en interpreter controleren. Tot deze test-set behoren behalve correcte programmas die de verschillende taalconstructies testen, ook programmas met syntactische, semantische en run-time fouten. Alle uitgevoerde tests moeten op de CD-R aanwezig zijn; van een testprogramma moet de uitvoer in de appendix opgenomen worden (zie onder).

8 Conclusies

9 Appendix

9.1 ANTLR Lexer specificatie

Specificatie van de invoer voor de ANTLR scanner generator, d.w.z. de token-definities van het taaltje.

9.2 ANTLR Parser specificatie

Specificatie van de invoer voor de parser generator, d.w.z. de structuur van de taal en de wijze waarop de AST gegenereerd wordt

9.3 Alle ANTLR TreeParser specificaties

Waarschijnlijk zult u (tenminste) twee tree parsers gebruiken: een context checker en een code generator.

9.4 Invoer- en uitvoer van een uitgebreid testprogramma

Van een correct en uitgebreid test- programma (met daarin alle features van uw programmeertaal) moet worden bijgevoegd: de listing van het oorspronkelijk programma, de listing van de gegenereerde TAM-code (be- standsnaam met extensie .tam) en een of meer executie voorbeelden met in- en uitvoer waaruit de juiste werking van de gegenereerde code blijkt.