

SELMA

Vonk, J
s0132778
Matenweg 75-201

Florisson, M
s0165972
Box Calslaan 60-30

July 5, 2011

Contents

1	Inleiding	5
2	Beknopte beschrijving	6
3	Problemen en oplossingen	7
3.1	Taalconstructie	7
3.2	Checker	7
3.3	Compiler	7
3.4	Randcode	8
4	Syntax, context-beperkingen en semantiek	9
4.1	Lexer - terminals	9
4.2	De basis - Programma	11
4.3	Expression_statement	11
4.4	Declaraties en types	11
4.4.1	Syntax	11
4.4.2	Context	12
4.4.3	Semantiek	12
4.4.4	Voorbeeld	12
4.5	Functiedeclaratie	12
4.5.1	Syntax	13
4.5.2	Context	13
4.5.3	Semantiek	13
4.5.4	Voorbeeld	13
4.6	Expressies - assignment	14
4.6.1	Syntax	14
4.6.2	Context	14
4.6.3	Semantiek	14
4.6.4	Voorbeeld	14
4.7	Expressies - OR	14
4.7.1	Syntax	15
4.7.2	Context	15
4.7.3	Semantiek	15
4.7.4	Voorbeeld	15
4.8	Expressies - AND	15
4.8.1	Syntax	16
4.8.2	Context	16
4.8.3	Semantiek	16
4.8.4	Voorbeeld	16
4.9	Expressies - Relaties	16
4.9.1	Syntax	16
4.9.2	Context	17
4.9.3	Semantiek	17
4.9.4	Voorbeeld	17

4.10	Expressies - plus en minus	17
4.10.1	Syntax	17
4.10.2	Context	17
4.10.3	Semantiek	17
4.10.4	Voorbeeld	18
4.11	Expressies - delen en vermenigvuldigen	18
4.11.1	Syntax	18
4.11.2	Context	18
4.11.3	Semantiek	18
4.11.4	Voorbeeld	18
4.12	Expressies - unaries	18
4.12.1	Syntax	19
4.12.2	Context	19
4.12.3	Semantiek	19
4.12.4	Voorbeeld	19
4.13	Expressies - toplevel	19
4.13.1	Syntax	20
4.13.2	Context	20
4.13.3	Semantiek	20
4.13.4	Voorbeeld	20
4.14	Unsigned constants	20
4.14.1	Syntax	20
4.14.2	Context	21
4.14.3	Semantiek	21
4.14.4	Voorbeeld	21
4.15	Identifier	21
4.15.1	Syntax	21
4.15.2	Context	22
4.15.3	Semantiek	22
4.15.4	Voorbeeld	22
4.16	Read	22
4.16.1	Syntax	22
4.16.2	Context	22
4.16.3	Semantiek	22
4.16.4	Voorbeeld	23
4.17	Print	23
4.17.1	Syntax	23
4.17.2	Context	23
4.17.3	Semantiek	23
4.17.4	Voorbeeld	23
4.18	If	23
4.18.1	Syntax	23
4.18.2	Context	24
4.18.3	Semantiek	24
4.18.4	Voorbeeld	24
4.19	While	24

4.19.1	Syntax	24
4.19.2	Context	24
4.19.3	Semantiek	24
4.19.4	Voorbeeld	24
4.20	Functieaanroep	25
4.20.1	Syntax	25
4.20.2	Context	25
4.20.3	Semantiek	25
4.20.4	Voorbeeld	25
4.21	Closed expression	25
4.21.1	Syntax	25
4.21.2	Context	26
4.21.3	Semantiek	26
4.21.4	Voorbeeld	26
4.22	Closed compoundexpression	26
4.22.1	Syntax	26
4.22.2	Context	26
4.22.3	Semantiek	26
4.22.4	Voorbeeld	26
5	Vertaalregels	27
5.1	Run	27
5.1.1	Program	27
5.2	Execute	28
5.2.1	ExpressionStatement	28
5.2.2	while	29
5.3	Evaluate	29
5.3.1	Compound Expression	29
5.3.2	if then else expression	29
5.3.3	Identifier	30
5.3.4	Integer Literal	30
5.3.5	Character Literal	30
5.3.6	Boolean Literals	30
5.3.7	Arithmetic, AND en OR	30
5.3.8	Relational operators	31
5.3.9	Unary Plus and Minus	31
5.3.10	NOT	32
5.3.11	Assignment	32
5.3.12	Print	32
5.3.13	Read	33
5.4	Elaborate	33

6	Beschrijving van Java programmatuur	34
6.1	main - SELMA	34
6.2	SELMAException	34
6.3	SELMATreeAdaptor	34
6.4	SELMATree	34
6.5	SymbolTable	35
6.5.1	SymbolTableException	36
6.6	IDEntry	36
6.7	CheckerEntry	36
6.8	CompilerEntry	36
7	Testplan en -resultaten	37
8	Conclusies	40
9	Appendix	41
9.1	ANTLR Lexer & Parser specificatie	41
9.2	ANTLR Checker specificatie	46
9.3	ANTLR Codegenerator specificatie	53
9.4	ANTLR Codegenerator Stringtemplate specificatie	60
9.5	Invoer- en uitvoer van een uitgebreid testprogramma	68
9.5.1	SELMA-code van pasen	69
9.5.2	Jasmin-code van pasen	72

1 Inleiding

Voor vertalerbouw dient als eindopdracht een eigen taal geschreven te worden. Deze taal dient een expression-language te zijn, dit is een taal die geen statements, maar enkel expressies kent. Alles wat je dus aanroept zal een waarde teruggeven.

Voor deze zelfbedachte taal dient een parser en lexer geschreven te worden, een checker en een compiler. Hierbij dient een verslag met een uitgebreide beschrijving van de taal en een goede kijk op hoe alles onder de motorkap werkt. Ook moet er een bewijs worden geleverd dat de taal werkt, dit kan door een testprogramma te schrijven dat tamelijk uitgebreid is en te kijken of dit werkt naar behoren. (exhaustive testing)

Hoe uitgebreid de te definiëren taal wordt is aan de studenten zelf - dit is echter terug te zien in het te behalen cijfer.

Voor onze taal, SELMA, hebben wij gekozen voor het volgende:

Klopt?

- Basic Expression Language
- If- & while-statements
- Ondersteunen van functies
- Compileren naar JVM-code in plaats van TAM-code

Onze taal heet SELMA. Een naam aan een taal geven is lastig, zo waren er een aantal andere opties zoals: SMEF of Taal voor Vertalerbouw (TV). SELMA staat voor Simpel Expression Language. Nu moest de afkorting wat meer zeggen dus kozen we voor de meisjesnaam SELMA, alleen maar omdat een afkorting vinden voor SELDERIE wel heel veel werk is.

Gelukkig heet onze taal dus geen SELDERIE, maar SELMA:

Waarbij de MA voor Minor Adjustments stond, we hebben inmiddels zoveel werk eraan gehad dat "Minor" dat geen eer meer aan doet.

Dus met gepaste trots presenteren wij u SELMA:

Simple Expression Language Met Augurk

Vanaf nu enkel nog naar te verwijzen als SELMA.

2 Beknopte beschrijving

Onze taal is gemaakt naar de gegeven instructies van de practicumhandleiding en alles is of een expressie of declaration in deze taal. Bij sommige expressies is het echter niet mogelijk een resultaat te geven, hier kunnen die expressies niet anders dan een void-resultaat retourneren, wat ze effectief een statement maakt. De structuur van de taal en de keywords lijken qua layout op een hybride tussen C en Pascal.

De volledige taal is LL(1) wij hebben hierdoor vooral tijdens het ontwerpen goed moeten nadenken hoe we de taal zo logisch mogelijk opbouwden zodat de parser er mee uit de voeten kon. Eventueel is er de mogelijkheid om lokaal 1 stap verder te kijken, wij hebben dit echter niet nodig gehad omdat wij voldoende keywords hebben gebruikt, zoals voor een functie een @ zetten - en we in de parser bewust rekening hebben gehouden met de LL(1) limitatie.

SELMA compileerde in eerste instantie naar TAM, op de cd is een fragment van deze code te zien. We hebben echter besloten dat het mooier was om JVM te gebruiken, niet zo zeer uit praktisch oogpunt, maar meer omdat JVM-bytecode ook door "echte" talen wordt gebruikt en omdat het een pluspunt is in de eindbeoordeling.

Op het moment dat we besloten om te schakelen waren we blij dat we hadden gekozen voor het gebruik van stringTemplates bij de codegeneratie, dit heeft ons wat werk gescheeld. En technisch gezien zouden we zo een extra compiler naar TAM-code erbij kunnen doen, aangezien er geen andere reden is dan "omdat het kan" hebben we onszelf die moeite bespaard.

Lees verder - of probeer eens een testprogramma te compileren in SELMA - om te leren hoe de vork nou precies in de steel zit met deze taal.

- *Mark & Jeroen*

NB: Aangezien CD's lang niet zo hip zijn als wat het internet heeft te bieden is ons gehele werk óók te vinden op github:

<http://github.com/markflorisson88/selma/>

3 Problemen en oplossingen

Tijdens het maken van de taal zijn we uiteraard af en toe tegen problemen aangelopen. Nu hebben wij tijdens het practicum de calc-taal al ontwikkeld dus we hadden al wat handigheid met ANTLR - en ANTLR's soms wat aparte foutmeldingen.

3.1 Taalconstructie

Wat ons is opgevallen is dat je van te voren goed moet specificeren hoe je taal er uit moet zien. Door bijvoorbeeld onze keuze om overal SEMICOLON's achter te zetten - wat op zich logisch is - kregen we soms wat onwennige taalconstructies. Zo dien je ook een semicolon na een functiedeclaratie te zetten, want het is een declaratie - en ook een semicolon na een if-expressie voelt wat ongebruikelijk. Omdat echter alles een expressie is in deze taal vonden we het passend hier geen uitzonderingen op te gaan maken door sommige expressies niet met een semicolon af te sluiten.

De eis om een taal LL(1) te maken heeft echter niet echt problemen opgeleverd, behalve dat we de declaratie voor een assignment op een andere plek wouden doen in eerste instantie (onder expressies-toplevel), hierbij was echter met LL(1) geen onderscheid te maken tussen een identifier en een assignment.

Een ander punt waar het onderscheid moeilijk was, waren functies. Deze zijn namelijk niet te onderscheiden van identifiers, tot je een haakje-openen na een identifier ziet. We hebben overwogen om een lokale forward-lookup te gebruiken, dit hebben we echter snel bestempeld als "slim valsspelen" en we hebben een '@' voor alle functie-aanroepen gezet. Klinkt ook mooi, aangezien je ook daadwerkelijk verwijst naar een stuk eerder gedefinieerde code.

Soms was het noodzakelijk om een stevige herschrijfregel te gebruiken, om in de checker en compiler wat meer gemak te hebben. Zo hebben we UMIN en EXPRESSION_STATEMENT toegevoegd. En hebben we vormen zoals (ID (COMMA ID)* COLON type) naar $\hat{(ID\ TYPE)}_+$ omgeschreven.

3.2 Checker

De checker heeft vrijwel geen problemen opgeleverd, aangezien onze randcode - Java-helperclasses etc. - gewoon netjes aansloot en een hoop werk uit handen nam.

Waar we wel consequent tegenaan liepen waren de wat vage manieren waarop er data uit de boom te halen is. Dollartekens voor Tokens, of juist niet, het was soms wat onduidelijk.

3.3 Compiler

In eerste instantie is de compiler in TAM geschreven, toen het echter een project van 2 werd in plaats van 1, is er besloten om een tandje bij te zetten en SELMA

in een wat algemener geaccepteerde code-vorm te compilen: JVM.

Mark, wat barft aan JV

En barfen stringtemplat

3.4 Randcode

De randcode is deels gebaseerd op de symboltables gebruikt tijdens het practicum en neemt een hoop werk uit handen. De symboltable-entries zijn per onderdeel (checker, compiler) anders. Dit omdat we merkten dat er soms te weinig gegevens waren over declaraties.

In de boom zelf konden we ook niet genoeg info kwijt, vandaar dat we een extension op de normale Tree hebben gemaakt, SELMATree, waarin is op te slaan wat het type is van elke expressie en of er variabele onderdelen in een expressie zitten.

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 definiëren 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 zestal 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;	
BEGIN;	
END;	
COMPOUND;	
EXPRESSION_STATEMENT;	

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 . '
UNDERSCORE	' _ '	IF	' if '
		THEN	' then '
		ELSE	' else '
		FI	' fi '
		WHILE	' while '
		DO	' do '
		OD	' od '
		FUNCDEF	' function '
		FUNCRETURN	' return '
		FUNCTION	' @ '
Operators			
NOT	' ! '		
MULT	' * '		
DIV	' / '		
MOD	' % '		
PLUS	' + '		
MINUS	' - '		
RELS	' < '		
RELSE	' < = '		
RELGE	' > = '		
RELG	' > '		
RELE	' = = '		
RELNE	' < > '		
AND	' & & '		
OR	' '		
BECOMES	' = = '		

4.2 De basis - Programma

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_statement? SEMICOLON! )+
    ;
```

4.3 Expression_statement

Dit is een speciale tussenstap voor de interne boekhouding. Na elke semicolon zal de mogelijk resterende waarde van de stack worden gepopped. Dit maakt dat er niet aan het eind van ons programma een hoop troep op de stack staat. Voorwaarde is wel dat er wordt bijgehouden wanneer een expression van het type void is, dan hoeft er namelijk niet gepopped te worden.

```
expression_statement
    : expression -> ^(EXPRESSION_STATEMENT expression)
    ;
```

4.4 Declaraties en types

SELMA kent twee soorten waarden-declaraties, variabelen en constanten. SELMA staat toe om per declaratie meerdere identifiers te definiëren. 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.

4.4.1 Syntax

```
declaration
//      : VAR^ identifier (COMMA! identifier)* COLON! type
//      | CONST^ identifier (COMMA! identifier)* COLON! type EQ!
//      unsignedConstant
```

```

      : VAR identifier (COMMA identifier)* COLON type
        -> ^(VAR type identifier)+
      | CONST identifier (COMMA identifier)* COLON type EQ
        unsignedConstant
        -> ^(CONST type unsignedConstant identifier)+
      | FUNCDEF^ identifier LPAREN! (funcpars SEMICOLON!)* RPAREN
        ! funcbody
      ;
funcpars : identifier (COMMA identifier)* COLON type -> (identifier
type)+;
type
  : INT
  | BOOL
  | CHAR
  ;

```

4.4.2 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.

4.4.3 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.

4.4.4 Voorbeeld

```

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

```

4.5 Functiedeclaratie

SELMA kent ook nog een functie declaratie. Deze valt logischerwijs ook onder de declaraties. De declaratie van een functie dient altijd voor het gebruik te komen. Een functie kan als een soort procedure worden gebruikt door geen return-type op te geven. Het return-type wordt dan automatisch void. Dit hebben we express gedaan, we willen het namelijk altijd een functie noemen, aangezien procedures niet echt een plek hebben binnen een expressietaal.

4.5.1 Syntax

```
      | FUNCDEF^ identifier LPAREN! (funcpars SEMICOLON!)* RPAREN
      ! funcbody
funcbody
: COLON type LCURLY compoundexpression FUNCRETUR
  expression SEMICOLON RCURLY -> ^(FUNCRETUR type
  compoundexpression expression)
| LCURLY! compoundexpression RCURLY!
;
```

4.5.2 Context

- De naam van de functie moet uniek zijn als functienaam, er mag wel een variabele of constante bestaan met die naam.
- De opgegeven identifiers moeten allemaal een andere naam hebben, ze hoeven echter niet uniek te zijn binnen het programma aangezien ze in een aparte scope staan.
- Het type van de expressie na het returntype dient hetzelfde te zijn als type.

4.5.3 Semantiek

Het adres waar deze functie staat wordt opgeslagen. Daarna komt de code van de functie. Aan het einde van de functie zal eventueel een result op de stack worden gezet en wordt het adres dat aan het begin is gegeven aangeroepen om weer terug te komen op de plek waar de functie wordt aangeroepen.

4.5.4 Voorbeeld

```
function foo() {
    6*7;
}
function foo(awesome, less: boolean; bar: integer): integer {
    var i: integer;

    if (awesome;) then
        i := 42;
    else
        i := 2;
    fi;

    return i;
}
```

4.6 Expressies - assignment

De expressies zijn ingedeeld in verschillende niveaus, dit om te zorgen dat ze in de juiste volgorde worden uitgevoerd. Zo willen we dat $6+3*12$ niet 108 is maar 42, niet alleen om dat 42 een mooier getal is, maar voornamelijk omdat het fijn is als de taal voldoet aan de conventionele rekenregels.

Het hoogste niveau is de assignment.

4.6.1 Syntax

```
expression
    : expr_assignment
    ;

expr_assignment
    : expr_arithmetic (BECOMES^ expression)?
    ;
```

4.6.2 Context

- `expr_arithmetic` moet een identifier worden, in het eind, aangezien dat het enige is waaraan je een waarde kunt toekennen
- deze identifier moet dan verwijzen naar een geldige variabele
- het type van `expression` en `expression_arithmetic` moet hetzelfde zijn
- `expression` is van het type van `expr_assignment`
- `expr_assignment` is van het type van `expr_arithmetic`

4.6.3 Semantiek

De waarde van `expression` zal worden toegekend aan het linker deel van de assignment. Tevens gaat de waarde van de hele expressie op de stack, zo is er een assignment met meerdere identifiers mogelijk.

4.6.4 Voorbeeld

```
7*6;
foo := 7*6;
foo := bar := 7*6;
```

4.7 Expressies - OR

De Of-operator is de laagste operator in het rijtje, vandaar dat deze bovenin de structuur zit.

NB: `expr_all` staat voor "expression arithmetic level 1"

4.7.1 Syntax

```
expr_arithmetic
: expr_all
;

expr_all                                     //
    expression arithmetic level 1
    : expr_al2 (OR^ expr_al2)*
    ;
```

4.7.2 Context

- Als `expr_al1` enkel uit 1 `expr_al2` bestaat dan zijn er geen restricties
- In de andere gevallen dienen alle `expr_al2` van het type boolean te zijn.
- het type van `expr_arithmetic` is het type van `expr_al1`
- als `exp_1 == expr_al2` dan is het type van `expr_al1` het type van `expr_al2`
- als `exp_1 != expr_al2` dan is het type van `expr_al1` een boolean

4.7.3 Semantiek

De eerste `expr_al2` zal op de stack worden gezet. Hierna wordt er telkens een `expr_al2` erbij gezet. De OR-operatie zal worden aangeroepen en het resultaat blijft op de stack zijn. Als er nog een `expr_al2` is dan zal deze ook op de stack worden gezet en wordt de OR-operatie opnieuw aangeroepen. Aldoende blijft er uiteindelijk 1 waarde op de stack staan.

4.7.4 Voorbeeld

```
7*6;
true OR false;
true OR false OR foo;
```

4.8 Expressies - AND

Hier wordt de AND-expressie beschreven. Net zoals bij de OR-expressie is het mogelijk nul tot veel AND-operatoren achter elkaar te plakken. De AND-expressie is een niveau hoger dan de OR-expressie en zal dus eerder worden uitgevoerd.

Het is eventueel mogelijk later in de compiler om een AND eerder af te breken aangezien als er een false in het rijtje zit het resultaat altijd false is. Wij hebben deze optimalisatie er nog niet inzitten, dit omdat sommige expressies ongeacht de eerdere expressies uitgevoerd dienen te worden, denk bijvoorbeeld aan een `READ()`-statement dat anders niet uitgevoerd zou worden.

4.8.1 Syntax

```
expr_al2
: expr_al3 (AND^ expr_al3)*
;
```

4.8.2 Context

- Als `expr_al2` enkel uit 1 `expr_al3` bestaat dan zijn er geen restricties
- In de andere gevallen dienen alle `expr_al3` van het type boolean te zijn.
- als `exp_2 == expr_al3` dan is het type van `expr_al2` het type van `expr_al3`
- als `exp_2 != expr_al3` dan is het type van `expr_al2` een boolean

4.8.3 Semantiek

Hetzelfde als bij het OR-statement. De waardes zullen op de stack geladen worden en er zal telkens een AND-operatie op 2 waardes worden uitgevoerd. De resulterende waarde is weer geschikt voor bijvoorbeeld nog een AND-operatie.

4.8.4 Voorbeeld

```
7*6;
foo AND bar;
foo AND false AND bar;
```

4.9 Expressies - Relaties

Hier worden bijna alle comperatoren afgehandeld. Het is belangrijk dat er in de checker goed wordt gekeken of de types van de linker en rechterzijde compatible zijn.

4.9.1 Syntax

```
expr_al3
: expr_al4 ((RELS|RELSE|RELG|RELGE|RELE|RELNE)^
  expr_al4)*
;
```

4.9.2 Context

- alle `expr_al4` dienen van hetzelfde type te zijn
- bij een operatie tussen twee `expr_al4` anders dan `RELE` & `RELNE` dient `expr_al4` een integer te zijn.
- als `exp_3 == expr_al4` dan is het type van `expr_al3` het type van `expr_al4`
- als `exp_3 != expr_al4` dan is het type van `expr_al3` een boolean

4.9.3 Semantiek

Vergelijkbaar met andere binaire operatoren zoals `AND` en `OR`, er zullen waardes op de stack worden gezet en de operatie zal 1 waarde achterlaten op de stack.

4.9.4 Voorbeeld

```
5 > 6;  
true == false;  
5 == 42;
```

4.10 Expressies - plus en minus

Hier zijn we aangeland bij de eerder genoemde `6+3*12`, plus en minus zit 1 niveau lager dan de vermenigvuldigingen.

4.10.1 Syntax

```
expr_al4  
    : expr_al5 ((PLUS|MINUS) ^ expr_al5)*  
    ;
```

4.10.2 Context

- als er minimaal 1 operatie wordt uitgevoerd dan dient `expr_al5` een integer te zijn
- als `exp_4 == expr_al5` dan is het type van `expr_al4` het type van `expr_al5`
- als `exp_4 != expr_al5` dan is het type van `expr_al4` een integer

4.10.3 Semantiek

Wederom een binaire operatie. Let op, de unaire plus en minus komen nog. Dus `5-6` zal de tweede minus niet hier worde opgevangen.

4.10.4 Voorbeeld

```
foo := 5;  
foo := 5 + 37;  
10 + 50 - 18;
```

4.11 Expressies - delen en vermenigvuldigen

Naast delen en vermenigvuldigen is het ook mogelijk een modulus te nemen. Wat wellicht is opgevallen bij het bovenstaande, is dat het mogelijk is om enkel een som in de code te zetten. Dit vinden wij prima, echter moet daarbij wel de resulterende waarde gepopped worden als die niet meer gebruikt wordt.

4.11.1 Syntax

```
expr_al5  
: expr_al6 ((MULT|DIV|MOD) ^ expr_al6)*  
;
```

4.11.2 Context

- als er minimaal 1 operatie wordt uitgevoerd dan dient `expr_al6` een integer te zijn
- als `exp_5 == expr_al6` dan is het type van `expr_al5` het type van `expr_al6`
- als `exp_5 != expr_al6` dan is het type van `expr_al5` een integer

4.11.3 Semantiek

Hetzelfde als bij optellen. Goed om te weten is dat de geretourneerde waarde een integer is, dus er zal worden afgerond.

4.11.4 Voorbeeld

```
foo := 6;  
foo := 6*7;  
foo := 21*6%84;
```

4.12 Expressies - unaries

Hier wordt gekeken of de expressie eventueel een NOT-, PLUS- of MIN-operator voor zich heeft staan. Om later verwarring te voorkomen zullen PLUS en MIN vervangen worden door speciale terminals, zijnde UMIN en UPLUS. UPLUS zou eventueel weg kunnen worden gelaten aangezien `+x==x`. Als er geen operator voor de expressie staat dan is `expr_al6` gewoon een `expr_al7`

4.12.1 Syntax

```
expr_al6
: PLUS expr_al7
  -> ^(UPLUS expr_al7)
| MINUS expr_al7
  -> ^(UMIN expr_al7)
| NOT expr_al7
  -> ^(NOT expr_al7)
| expr_al7
;
```

4.12.2 Context

- expr_al7 dient bij PLUS expr_al7 een integer te zijn
- expr_al7 dient bij MIN expr_al7 een integer te zijn
- expr_al7 dient bij NOT expr_al7 een boolean te zijn
- het type van expr_al6 het type van expr_al7

4.12.3 Semantiek

Bij UMIN zal $\text{expr_al6} == - \text{expr_al7}$

Bij UPLUS zal $\text{expr_al6} == \text{expr_al7}$

Bij NOT zal $\text{expr_al6} == ! \text{expr_al7}$

4.12.4 Voorbeeld

```
one := +1;
evil := -42;
foo := not foobar;
```

4.13 Expressies - toplevel

Op het hoogste niveau kan een expressie bestaan uit een semi-statement zoals een if-expressie of een print-expressie, of het kan een identifier of waarde zijn, of het kan een aparte (compound)expressie binnen haken zijn. Zoals je ziet stond in eerste de assignment hier. Maar aangezien het meest linkerdeel van een assignment een identifier is kan op L=1 geen onderscheid worden gemaakt tussen identifier of een assignment. Vandaar dat een assignment bij expr_al1 is gedefinieerd.

4.13.1 Syntax

```
expr_al7
: unsignedConstant
| identifier
// | expr_assignment //can be identifier
| expr_read
| expr_print
| expr_if
| expr_while
| expr_closedcompound
| expr_closed
| expr_funccall
;
```

4.13.2 Context

- expr_al7 is van hetzelfde type als de gegeven expressie of waarde.

4.13.3 Semantiek

Dit is enkel een lijst van mogelijke expressies en waardes en dus zal er in de compiler enkel deze expressie of waarde op stack hebben staan, maar wordt er geen operatie op uitgevoerd.

4.13.4 Voorbeeld

```
foo;
42;
(foo bar);
```

4.14 Unsigned constants

Uiteraard bied onze taal ook de mogelijkheid aan om constanten te gebruiken zonder deze eerst te moeten declareren. Oftewel, je kunt gewoon nummers gebruiken bijvoorbeeld.

4.14.1 Syntax

```
unsignedConstant
: boolval
| charval
| intval
;

intval
: NUMBER
;
```

```

boolval
    : BOOLEAN
    ;

charval
    : CHARV
    ;

CHARV
    : APOSTROPHE (LETTER|UNDERSCORE) APOSTROPHE
    ;

```

4.14.2 Context

- unsignedconstant is van het type van de gegeven waarde
- boolval is een boolean type
- charval is een char
- intval is een integer

4.14.3 Semantiek

De desbetreffende waarde wordt op de stack gezet.

4.14.4 Voorbeeld

```

'Y';
42;
true;

```

4.15 Identifier

Een identifier van een bestaande variabele of constante in de huidige of een hogere scope.

4.15.1 Syntax

```

identifier
    : ID
    ;

ID
    : LETTER (LETTER | DIGIT)*
    ;

```

4.15.2 Context

- De identifier dient te verwijzen naar een geldige variabele of constante
- Het type is het type van de variabele of declaratie waar de identifier naar verwijst.

4.15.3 Semantiek

Er zal een commando aangeroepen worden om de waarde uit het geheugen te laden. Deze waarde wordt dan op de stack gezet. Bij constanten gebeurt dit ook. Eventueel zou je ook de constante zelf al kunnen neerzetten op de stack, dit scheelt weer wat werk voor de processor. Dit doen wij echter niet momenteel.

4.15.4 Voorbeeld

```
Answer42;
```

4.16 Read

Om contact te hebben met de buitenwereld kan onze taal lezen en schrijven naar de standard-out.

4.16.1 Syntax

```
expr_read  
  : READ^ LPAREN! identifier (COMMA! identifier)* RPAREN!  
  ;
```

4.16.2 Context

- Identifier dient te verwijzen naar een geldige identifier
- De ingelezen waarde dient van het zelfde type als identifier te zijn
- Als er 1 identifier is opgegeven dan geeft read de gelezen waarde/type terug
- Als er meer dan 1 identifier wordt ingelezen dan is het returntype void

4.16.3 Semantiek

Het read-commando wordt aangeroepen en de waarde wordt van de standard-out gelezen en op de stack gezet. Vervolgens wordt die waarde opgeslagen in de variabele.

4.16.4 Voorbeeld

```
read(foo);  
read(foo, bar);
```

4.17 Print

De taal heeft ook de mogelijkheid om dat wat er bijvoorbeeld berekend is naar buiten te communiceren.

4.17.1 Syntax

```
expr_print  
  : PRINT LPAREN expression (COMMA expression)* RPAREN  
    -> ^(PRINT expression+)  
  ;
```

4.17.2 Context

- -

4.17.3 Semantiek

De waarde van de expressie staat op de stack. Vervolgens wordt deze netjes naar het scherm uitgevoerd. Afhankelijk van het type zal dat anders gebeuren.

4.17.4 Voorbeeld

```
print(42);  
print('4', '2');
```

4.18 If

Om keuzes in het programma mogelijk te maken zal er een conditioneel statement nodig zijn, het IF-statement is een dergelijk statement. Een ELSE-deel is optioneel.

4.18.1 Syntax

```
expr_if  
  : IF^ compoundexpression THEN compoundexpression (ELSE  
    compoundexpression)? FI!  
  ;
```


4.18.2 Context

- De eerste compoundexpression moet een boolean-type retourneren
- De if retourneert een type void

retourtype

4.18.3 Semantiek

Als de waarde binnen het ifstatement waar is dan zal de eerste compoundexpression worden uitgevoerd (na de then). Anders zal de andere compoundexpression worden uitgevoerd, mits deze is gedeclareerd.

4.18.4 Voorbeeld

```
if true; then i := 42; fi
if false; then i := 0; else i:=42; fi
```

4.19 While

De while zal net zolang een blok code uitvoeren tot een gegeven expressie waar is.

4.19.1 Syntax

```
expr_while
: WHILE^ compoundexpression DO compoundexpression OD
;
```

4.19.2 Context

- De eerste compoundexpression moet een boolean-type retourneren
- De while retourneert een type void

4.19.3 Semantiek

De tweede compoundexpression zal worden uitgevoerd tot de eerste compoundexpression waar is. Het kan zijn dat de tweede compoundexpression nooit wordt uitgevoerd dus.

4.19.4 Voorbeeld

```
while false; do
\\ this is not gonna be executed
tru := false;
od
```

```
while foo < 5; do
foo := foo + 1;
od
```

4.20 Functieaanroep

Een functieaanroep naar een eerder gedefinieerde functie

4.20.1 Syntax

```
expr_funcall
: FUNCTION^ identifier LPAREN! (expression COMMA!)* RPAREN!
;
```

4.20.2 Context

- Het aantal expressies en hun type dient overeen te komen met de declaratie van de functie
- De functie retourneert het eerder gespecificeerde type. Als er geen type was gedeclareerd dan is dat dus void.

4.20.3 Semantiek

Het returnadres wordt op de stack gezet, zodat de functie weer hiernaartoe kan terugkeren. De expressies worden op de stack gezet in de gespecificeerde volgorde. De functie wordt aangeroepen. De functie returned en het result staat op de stack.

4.20.4 Voorbeeld

```
@foo();
i := @foo('b', 'a', 'r', );
```

4.21 Closed expression

Een expressie tussen haakjes is soms handig, bijvoorbeeld bij sommetjes: $(5+2)*6$;

4.21.1 Syntax

```
expr_closed
: LPAREN! expression RPAREN!
;
```

4.21.2 Context

- De geretourneerde waarde zal de waarde van de expressie zijn binnen de haakjes.
- Het retourneerde type is ook hetzelfde als die van de expressie.

4.21.3 Semantiek

De expressie binnen de haakjes zal worden uitgevoerd binnen de haakjes.

4.21.4 Voorbeeld

```
(3*(6+8))%102;
```

4.22 Closed compoundexpression

Is een compoundexpressie binnen haakjes. Verschil met de expressie tussen haakjes is dat deze ook toestaat om declaraties te gebruiken. Een compound tussen haakjes zal een eigen scope hebben.

4.22.1 Syntax

```
expr.closedcompound  
    : LCURLY^ compoundexpression RCURLY  
    ;
```

4.22.2 Context

- retourneert het waarde en de type van de laatste expressie in de compound, dit kan van het type void zijn.

4.22.3 Semantiek

De compoundexpressie zal in een eigen scope worden uitgevoerd.

4.22.4 Voorbeeld

```
{  
var foo: integer;  
foo := 40;  
foo+2;  
}
```

expression_statement

5 Vertaalregels

Deze sectie specificeert hoe selma programmas naar Jasmin worden vertaald. Jasmin is een assembler die gegeven Jasmin assembly JVM bytecode genereert in de vorm van een `.class` file.

De vertaling wordt gedaan door middel van een compiler (`g-files/SELMACompiler.g`) in ANTLR die executeert na de checker en producties uit de Jasmin string template aanroept (`SELMACodeJasmin.stg`).

We gebruiken de volgende code functies:

- `run : Program -> Instruction*`
- `execute — ExpressionStatement -> Instruction*`
- `evaluate : Expression -> Instruction*`
- `fetch : Identifier -> Instruction*`
- `assign : Identifier -> Instruction*`
- `elaborate : VarDeclaration -> Instruction*`

Phare	Code Function	Effect
Program	run P	Run P en begin en eindig met een lege stack. Doe ook de nodige declaraties om classen en methoden te genereren. Return hierna.
ExprStat	execute E	Executeer statement S. Als S een expressie is dat een waarde op de stack genereert, pop. Dit verandert de stack niet.
Expression	evaluate E	Evalueer expressie E en laat de nieuwe waarde op de stack staan. Een expressie kan 1 of 2 oude waarden van de stack halen (e.g. AND).
ID	fetch I	Laad de waarde van I en zet het op de stack.
ID	assign I	Pop de top van de stack en sla het op in I.

In onze vertaalregels zullen we variabelen omringen met `<en >`.

5.1 Run

5.1.1 Program

```
run[P, source_file, stack_limit, locals_limit, pop] =  
  .source <source_file>  
  .class public Main  
  .super java/lang/Object  
  .field public static scanner_field Ljava/Util/Scanner;  
  
  .method public static main([Ljava/lang/String;)V  
  .limit stack <stack_limit>  
  .limit locals <locals_limit>
```

```

new java/util/Scanner
dup
getstatic java/lang/System/in Ljava/io/InputStream;
invokespecial java/util/Scanner/<init>(Ljava/io/InputStream;)V
putstatic Main/scanner_field Ljava/util/Scanner;

evaluate[P]
if <pop>:
    pop

return
.end method

```

De `source.file`, `stack.limit` en `locals.limit` geven respectievelijk aan wat de originele source file was (voor runtime excepties), de grootte van de stack en het aantal locale variabelen dat het programma gebruikt. `pop` geeft aan of `P` een expressie was en nog een waarde op de stack heeft achtergelaten. Dit zou ook opgelost kunnen worden door de regel

```

program -> expression_statement
expression_statement -> expression
expression -> ... | compoundexpression
compoundexpression -> (declaration | expression_statement)*

```

in plaats van

```

program -> compoundexpression

```

Helaas resulteert dit in Left Recursion waar we geen tijd meer voor hadden dit op te lossen. Een simpele `pop = expression.type != type.VOID` lost dit echter gauw genoeg op.

Als er labels in code regels voorkomen als `L1`, `L2`, etc, zullen deze in werkelijkheid uniek genummerd zijn.

5.2 Execute

5.2.1 ExpressionStatement

```

execute[S, pop] =
    evaluate[S]
    if pop:
        pop

```

`S` is hierbij een expressie en `pop` is wederom true iff `expression.type != type.VOID`.

5.2.2 while

```
execute[while E; do S; od] =  
  L1:  
    evaluate[E]  
    ifeq L2  
    execute[S]  
    goto L1  
  L2:
```

De `ifeq` instructie kijkt of de waarde op de top van de stack gelijk is aan 0 (boolean waarden zijn integer waarden 0 of 1), en als dat het geval is jumpst de interpreter naar de label L2 (naar de eerstvolgende instructie na de while loop). Als dit niet het geval is gaat hij verder met de eerste instructie van `S` en hierna volgt een jump naar het begin om te kijken of een volgende iteratie nodig is.

5.3 Evaluate

5.3.1 Compound Expression

Omdat een compound expression ook weer een expressie is, maar bestaat uit expressie statements, moet de codegenerator de mogelijk gegenereerde pop van de laatste expressie statement verwijderen. De regel is als volgt:

```
compoundexpression = COMPOUND (declaration | expression_statement)*
```

```
evaluate[E, last_expr_is_void] =  
  evaluate[E]  
  if not <last_expr_is_void>:  
    remove_last_instruction
```

Hier geeft de variabele `last_expr_is_void` aan of de laatste expressie (als er tenminste 1 expressie is) van type VOID is. Als dit niet het geval is, is er een pop gegenereerd door `expression_statement` die verwijderd moet worden. Dit gebeurt door een dummy instructie `remove_last_instruction` te genereren die de laatste instructie verwijderd. Dit gebeurt voordat het resultaat naar een Jasmin assembly file geschreven wordt.

5.3.2 if then else expression

```
evaluate[if E1 then E2 (else E3)?] =  
  evaluate[E1]  
  ifeq L1  
  evaluate[E2]  
  goto L2  
  L1:  
    evaluate[E3]  
  L2:
```

5.3.3 Identifier

```
evaluate[ID, kind] =  
    if kind == CONST:  
        ldc getvalue(<ID>)  
    else:  
        iload address_of(<ID>)
```

Als de variabele een constante is wordt de waarde bijgehouden in de symbol table. De functie `getvalue` haalt hier de waarde van de constante op. `address_of` is hier een functie die gegeven een identifier zijn address als locale variabele ophaalt.

5.3.4 Integer Literal

```
evaluate[literal, iconst, bipush, ldc] =  
    if iconst:  
        iconst_<literal>  
    elif bipush:  
        bipush <literal>  
    else:  
        ldc <literal>
```

Als de integer literal in de juiste range van `iconst` of `bipush` zit, worden deze geprefereerd over `ldc` voor compactere bytecode.

5.3.5 Character Literal

```
evaluate[literal] =  
    bipush <literal>
```

5.3.6 Boolean Literals

```
evaluate[true] =  
    iconst_1  
  
en  
  
evaluate[false] =  
    iconst_0
```

5.3.7 Arithmetic, AND en OR

```
evaluate[E1, op, E2, instruction] =  
    evaluate[E1]  
    evaluate[E2]  
    <instruction>; E1 <op> E2
```

Hierbij is op een binaire arithmetische operator zoals +, -, etc, of AND/OR. `instruction` is de bijbehorende JVM Jasmin instructie. De operators mappen als volgt naar hun instructies:

- + : `iadd`
- - : `isub`
- * : `imul`
- / : `idiv`
- % : `irem`
- && : `iand`
- — : `ior`

5.3.8 Relational operators

```

evaluate[E1 op E2, instruction] =
    evaluate[E1]
    evaluate[E2]
    <instruction> L1    ; E1 <op> E2
    iconst_0
    goto L2
L1:
    iconst_1
L2:

```

Voor elke relationele operator zoals <=, == etc wordt deze code gegenereerd met bijbehorende instructie. De operator naar instructie mapping is als volgt:

```

< : ifcmp_lt
<= : ifcmp_le
== : ifcmp_eq
!= : ifcmp_ne
>= : ifcmp_ge
> : ifcmp_gt

```

5.3.9 Unary Plus and Minus

```

evaluate[-E] =
    evaluate[E]
    ineg

```

Voor unary + hoeft er niets te gebeuren.

5.3.10 NOT

```
evaluate[!E] =
    evaluate[E]
    ifeq L1
    iconst_0
    goto L2
L1:
    iconst_1
L2:
```

5.3.11 Assignment

```
evaluate[ID := E, address] =
    evaluate[E]
    dup
    istore <address>
```

`address` is het address van de locale variabele. De `dup` is nodig om de waarde eerst te dupliceren aangezien assignment een expressie is.

5.3.12 Print

```
evaluate[print(E+), type_denoters, bools, dup_top] =
    for expr, type_denoter, is_bool in E, type_denoters, bools:
        evaluate[expr]

        if <dup_top>:
            dup

        if <is_bool>:
            ifeq L1
            ldc "true"
            goto L2
        L1:
            ldc "false"
        L2:

        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println(<type_denoter>)V
```

Hier krijgt de vertaalregel voor elke expressie mee of het type `boolean` is (`bools`), en welke overloaded versie van `System.out.println` moet worden aangeroepen (`type_denoters`). In het geval van een boolean moet de waarde `"true"` of `"false"` worden geladen in plaats van de integer waarde. De boolean `dup_top` geeft aan of de print een statement of expressie is. Als het een expressie

is (in het geval van een enkele print), moet de waarde worden gedupliceerd op de stack voordat de println de waarde popt.

5.3.13 Read

```
evaluate[read(ID+), bools, ints, dup_top] =
  for id, is_bool, is_int in ID, bools, ints:
    getstatic Main/scanner_field Ljava/util/Scanner;

    if <is_bool>:
      invokevirtual java/util/Scanner/nextBoolean()Z
    elif <is_int>:
      invokevirtual java/util/Scanner/nextInt()I
    else:
      invokevirtual java/util/Scanner/nextByte()I

    if <dup_top>:
      dup

    istore address_of(<id>)
```

Hier krijgt de vertaalregel voor elke expressie mee of het type `boolean` of `int` is (`bools`, `textttints`) en of de `read` een expressie is (`dup_top`). `address_of` is hier een functie die gegeven een identifier zijn address als locale variabele ophaalt. Afhankelijk van het type (`boolean`, `integer` of `character`) wordt een aanroep gedaan naar de methoden `nextBoolean()`, `nextInt()` of `nextByte()` van `Scanner`.

5.4 Elaborate

De `elaborate` code functie genereert geen code, maar maakt entries aan in de compiler's symbol table.

```
elaborate[VarDeclaration type ID] =
  increase amount of local variables by one
  enter <ID> of <type> in symbol table

elaborate[ConstDeclaration type ID value] =
  increase amount of local variables by one
  enter <ID> of <type> in symbol table
  set value on symbol table entry of <ID>
```

Constanten nemen geen ruimte in op de stack of als locale variabelen maar worden in de symbol table bijgehouden. Variabelen zijn locale variabelen in de JVM. Dit betekent dat variabelen in de global scope niet beschikbaar zijn voor functies, aangezien dat andere methoden zijn en geen closures over de main functie. Hiervoor zouden het statische of non-statische fields moeten zijn. Zie ook de subsectie over Identifiers 5.3.3.

6 Beschrijving van Java programmatuur

6.1 main - SELMA

SELMA.java is het main-programma. Je kunt een aantal opties en een SELMA-sourcecodefile meegeven. Hierna zal SELMA desbetreffende file parsen en compileren. De opties die mogelijk zijn zijn:

- -ast Er zal een ast-diagram naar de stdout worden geprint van de source-code.
- -dot Er zal een dot-diagram naar de stdout worden geprint van de source-code.
- -no_checker De source-code wordt geparsed maar niet gechecked.
- -code_generator De source-code zal worden gecompileerd

De sourcecode zal de volgende stappen doorlopen:

Lexer	Parser	-no_checker	-ast	Ast-diagram
Lexer	Parser	-no_checker	-dot	Dot-diagram
Lexer	Parser	Checker	-ast	Ast-diagram
Lexer	Parser	Checker	-dot	Dot-diagram
Lexer	Parser	Checker	-code_generator	Code

Alle resultaten zullen altijd naar de stdout worden geprint.

6.2 SELMAException

Als er wat fout gaat in bijvoorbeeld de checker dan zal er een exception worden gegooit. Deze exception is een SELMAException. Aan de exception wordt de node meegegeven waar de checker op dat moment mee bezig is. En de toString()-functie van SELMAException zal dat dan ook mooi formatten in de vorm van "(regelnummer:columnnummer) ErrorMessage", toch wel fijn als je moet debuggen.

6.3 SELMATreeAdaptor

Deze TreeAdaptor heeft SELMATree als nodes, in plaats van een normale Tree.

6.4 SELMATree

SELMATree is een uitbreiding op de normale tree. En kan een aantal extra dingen bijhouden, namelijk of een expressie constant is of variabel, wat later handig is voor optimizing. En wat het type is van de expressie, dat is zeer handig voor de checker. Daarvoor heeft SELMATree een paar extra attributen, zijnde:

```

    public SR.Type SR_type = null;
    public SR.Kind SR_kind = null;
    public SR.Func SR_func = null;

    /* Given a type as AST Tree node, return the SR.Type */

```

En verder kent SELMATree nog drie functies om mooi te kunnen printen:

```

    switch (token.getType()) {
        return SR.Type.BOOL;
        SELMATree t = (SELMATree)children.get(i);
    }

```

6.5 SymbolTable

De symboltable houdt al onze variabelen en constanten bij. Ook kun je in de symboltable scopes aanmaken, om bijvoorbeeld variabelen binnen een compoundexpressie te kunnen declareren. De dataopslag van de symboltable geschiedt middels een Map waarin een string aan een stack van IDEntries wordt gekoppeld. De string verwijst naar de naam van de variabele of constante. De stack bevat meerdere declaraties van die variabele met die naam in verschillende scopes. Zodat het mogelijk is de zelfde naam tweemaal te gebruiken, mits ze in een andere scope gebruikt worden.

De symboltable kent een aantal functies, de belangrijkste zijn:

```

    public SymbolTable() {
        currentLevel = -1;
        entries = new HashMap<String, Stack<Entry>>();
    }
    /*
    public void openScope() {
        currentLevel++;
    }

    /**
     * Closes the current scope. All identifiers in
     */

    /** Returns the current scope level. */
    public int currentLevel() {
        return currentLevel;
    }

    /** Return whether the entry takes up space on the stack */
    private boolean isLocal(Entry entry) {
        if (s.isEmpty() || s.peek().level != currentLevel) {
            entry.level = currentLevel;
            s.push(entry);
            if (isLocal(entry))
                nextAddr++;
        }
    }

```

6.5.1 SymbolTableException

SymbolTableException is er om fouten in de symboltable aan te geven. Deze fouten zullen vergelijkbaar worden geformat als die van SELMAException, namelijk "(line:column) ErrorMsg."

6.6 IDEntry

De symboltable bevat voor elke variabele of constante een IDEntry. Een IDEntry bevat de scopelevel van desbetreffende declaratie. Wij gebruiken in onze code echter een tweetal klassen die ge-extend zijn op IDEntry; CheckerEntry en CompilerEntry.

6.7 CheckerEntry

De CheckerEntry wordt gebruikt in de Checker. Een checkerEntry verschilt van een IDEntry op het punt dat een checkerEntry twee extra waardes heeft om bij te houden wat het type is van de variabele of constante (int,bool of char). De tweede waarde is om bij te houden of we met een constante of een variabele te maken hebben.

```
import org.antlr.runtime.tree.Tree;
```

6.8 CompilerEntry

De compilerEntry is weer een uitbreiding op de CheckerEntry. Voor de compiler is het namelijk noodzakelijk om te weten op welk adres in de te genereren code de variabele staat. Dit wordt bijgehouden door:

```
public class CompilerEntry extends CheckerEntry {
```

7 Testplan en -resultaten

Voor het testen hebben we testprogramma's geschreven in onze taal. Ook zit er een testrunner bij die automatisch alle tests in de 'test' subdirectory vind en compileerd en optioneel executeerd. Tests kunnen van de volgende typen zijn:

- Compile - Compileer de test
- Error - Compileer en (als succesvol), executeer
- Run - Compileer en executeer

Bij deze tests kunnen in het programma tags gezet worden, namelijk `{input;text}/input` voor input voor het programma op stdin, en `{output;text}/output` voor output van het programma (of de compiler, in het geval van een compile of error test). Error tests beginnen met de prefix 'error_' in de bestandsnaam, en compile tests met 'compile_'. Zo kan getest worden voor juiste syntax en semantiek, juiste error reporting bij onjuiste syntax en semantiek, en correctie vertaalregels door middel van correcte executie, en runtime error checking voor juiste programmas met runtime fouten. Om de tests te runnen is Python 2.5+ of 3.0 nodig. De tests kunnen als volgt worden geexecuteerd:

```
\$ python test.py
```

of

```
\$ make tests
```

Als een run test geen output heeft gespecificeerd is de exit status van het programma bepalend of de test faalt of niet. Bij een error test geldt het tegenovergestelde: zonder gespecificeerde output moet de exit status nonzero zijn.

De tests in de test directory testen alle constructen uit de taal, zoals arithmetiek, alle operators, typen, constanten, scope rules, etcetera. Hieronder is output gegeven van de test runner. Als een test faalt zal de output worden weergegeven:

```
[0] [11:11] ~/selma git(master!) python test.py
Run      test/correct.selma
... OK
Error    test/error_context.selma
... OK
Error    test/error_if.selma
... OK
Error    test/error_runtime_uninitialized.selma
... OK
Error    test/error_runtime_zerodivision.selma
... OK
Error    test/error_syntax.selma
... OK
Error    test/error_while.selma
... OK
```

<pre> Error test/error_while_void.selma ... OK Run test/sample.selma ... FAIL (exit status 0) Got: h >>> a l l o Expected: h >>> e l l o </pre>	
<pre> Run test/test_if.selma ... OK Run test/test_operators.selma ... OK Run test/test_while.selma ... OK Run test/test_functions/correct_functions.SELMA ... FAIL (exit status 1) ERROR: recognition exception thrown by compiler: null org.antlr.runtime.EarlyExitException at SELMA.SELMACompiler.compoundexpression(SELMACompiler.java:271) at SELMA.SELMACompiler.program(SELMACompiler.java:170) at SELMA.SELMA.main(SELMA.java:99) </pre>	
<pre> Error test/test_functions/error_doublefunction.selma ... OK Error test/test_functions/error_wrongparamcount.selma ... OK Error test/test_functions/error_wrongparamtype.selma ... OK Error test/test_functions/error_wrongreturntype.selma ... OK Ran 17 test(s), SUCCESS=15, FAILURE=2 </pre>	

Hier zien we dat `test/sample.selma` niet de correcte output heeft, maar wel executeerde zonder fouten (exit status 0), terwijl `test/test_functions/correct_functions.SELMA` een compilatie fout had met een exit status 1. De eerste kolom geeft het type test aan, in dit geval 'Error' of 'Run'. Ter demonstratie is `test/sample.selma` bijgevoegd:

```
<output>
  h
  e
  l
  l
  o
</output>
print('h', 'a', 'l', 'l', 'o');
```


8 Conclusies

We hebben een taal gedefinieerd, uitgeschreven, regels aan toegevoegd en dit omgezet naar een stuk Jasmin. Zonder ons in al te veel bochten te moeten wringen om de taal werkend te krijgen of binnen de LL(1) restrictie zien te krijgen.

Tot zover lijkt ons dat het over het algemeen wel goed gegaan is - begrijp ons niet verkeerd, we hadden graag nog arrays geïmplementeerd - maar we hebben een werkende taal, welke niet aan elkaar geplakt zit van de lelijke oplossingen.

We hebben zeker 40 pagina's verslaglegging (met appendices 100+). Waarin we hebben geprobeerd alles uit te leggen, hier en daar op een wat informele toon, maar we weten wie het leest. En om de lezer continu met 'u' aan te spreken - en geen kleine grapjes te kunnen maken - is het wel een erg droog verslag om te lezen.

Het had ons persoonlijk erg leuk geleken om een optimizer te schrijven, dit is echter nogal wat werk en we zijn daar niet aan toegekomen. Bovendien dienen we aan de boekhouding te denken, in het verslag stond niet duidelijk of dat ons pluspunten zou opleveren. De uitdaging aan de optimizer was denk ik het puzzelen geweest, om net overal de code net iets vlotter te maken. Zo is er bijvoorbeeld al rekening gehouden door van elke expressie bij te houden of deze variabele onderdelenbevat, als een expressie compleet constant zou zijn dan zou je immers het net zo goed een maal kunnen uitrekenen en als dergelijk in de code te zetten.

Mark, mening?

9 Appendix

9.1 ANTLR Lexer & Parser specificatie

```

5 grammar SELMA;

options {
    k=1; // LL(1) - do not use LL(*)
    language=Java; // target language is Java (= default)
    output=AST; // build an AST
}

10 tokens {
    COLON = ':';
    SEMICOLON = ';';
    LPAREN = '(';
    RPAREN = ')';
    LCURLY = '{';
    RCURLY = '}';
    COMMA = ',';
    EQ = '=';
    APOSTROPHE = '\'';
    20 UNDERSCORE = '_';
    //arethemithic
    NOT = '!';

    MULT = '*';
    DIV = '/';
    25 MOD = '%';

    PLUS = '+';
    MINUS = '-';

    30 RELS = '<';
    RELSE = '<=';
    RELGE = '>=';
    RELG = '>';
    35 RELE = '==';
    RELNE = '<>';

    AND = '&&';

    40 OR = '||';

    //expressions
    BECOMES = ':=';
    PRINT = 'print';
    45 READ = 'read';

    //declaration
    VAR = 'var';
    CONST = 'const';

    50 //types
    INT = 'integer';
    BOOL = 'boolean';
    CHAR = 'character';

    55 //keywords
    IF = 'if';
    THEN = 'then';
    ELSE = 'else';
    FI = 'fi';

    60 WHILE = 'while';
    DO = 'do';
    OD = 'od';

```

```

65     FUNCDEF = 'function ';
        FUNCRETURN = 'return ';
        FUNCTION = '@';

70     UMIN;
        UPLUS;

        BEGIN;
        END;
        COMPOUND;
75     EXPRESSION_STATEMENT;
    }

    @header {
        package SELMA;
80    }

    @lexer::header {
        package SELMA;
85    }

90

95

// Parser rules – program at line 100 due to the report

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

105 compoundexpression
    : cmp -> ^(COMPOUND cmp)
    ;

110 cmp
    : ((declaration SEMICOLON!)* expression_statement? SEMICOLON! )+
    ;

    //declaration

115 declaration
    // : VAR^ identifier (COMMA! identifier)* COLON! type
    // | CONST^ identifier (COMMA! identifier)* COLON! type EQ!
    unsignedConstant
    : VAR identifier (COMMA identifier)* COLON type
      -> ^(VAR type identifier)+
120 | CONST identifier (COMMA identifier)* COLON type EQ
    unsignedConstant
      -> ^(CONST type unsignedConstant identifier)+
    | FUNCDEF^ identifier LPAREN! (funcpars SEMICOLON!)* RPAREN!
    funcbody
    ;
    funcpars : identifier (COMMA identifier)* COLON type -> (identifier type
125 type )+;
    : INT
    | BOOL

```

```

130         | CHAR
        ;

funcbody
: COLON type LCURLY compoundexpression FUNCRETURN expression
  SEMICOLON RCURLY -> ^(FUNCRETURN type compoundexpression
    expression)
  | LCURLY! compoundexpression RCURLY!
  ;

135

140

//expression statement at line 146
145 expression_statement
    : expression -> ^(EXPRESSION_STATEMENT expression)
    ;
    // note: - arithmetic can be "invisible" due to all the *-s that's why
    // it is nested
150 // - assignment can be "invisible" due to the ? that's why it can also
    // be only a identifier
    expression
    : expr_assignment
    ;

155 expr_assignment
    : expr_arithmetic (BECOMES^ expression)?
    ;

expr_arithmetic
160 : expr_al1
    ;

    expr_al1                                     //expression
    arithmetic level 1
    : expr_al2 (OR^ expr_al2)*
165 ;

    expr_al2
    : expr_al3 (AND^ expr_al3)*
    ;

170 expr_al3
    : expr_al4 ((RELS|RELSE|RELG|RELGE|RELE|RELNE)^ expr_al4
    )*
    ;

175 expr_al4
    : expr_al5 ((PLUS|MINUS)^ expr_al5)*
    ;

    expr_al5
180 : expr_al6 ((MULT|DIV|MOD)^ expr_al6)*
    ;

    expr_al6
    : PLUS expr_al7
    -> ^(UPLUS expr_al7)
185 | MINUS expr_al7
    -> ^(UMIN expr_al7)
    | NOT expr_al7
    -> ^(NOT expr_al7)

```

```

190         | expr_al7
        ;

        expr_al7
195         : unsignedConstant
        | identifier
        //      | expr_assignment //can be identifier
        | expr_read
        | expr_print
        | expr_if
200         | expr_while
        | expr_closedcompound
        | expr_closed
        | expr_funccall
        ;

205 expr_read
      : READ^ LPAREN! identifier (COMMA! identifier)* RPAREN!
      ;

expr_print
210      : PRINT LPAREN expression (COMMA expression)* RPAREN
      -> ^(PRINT expression+)
      ;

expr_if
      : IF^ compoundexpression THEN compoundexpression (ELSE
215         compoundexpression)? FI!
      ;

expr_while
      : WHILE^ compoundexpression DO compoundexpression OD
      ;

220 expr_funccall
      : FUNCTION^ identifier LPAREN! (expression COMMA!)* RPAREN!
      ;

225 expr_closedcompound
      : LCURLY^ compoundexpression RCURLY
      ;

expr_closed
230      : LPAREN! expression RPAREN!
      ;

235

240

//unsigned at line 244

unsignedConstant
245      : boolval
      | charval
      | intval
      ;

250 intval
      : NUMBER
      ;

boolval
255      : BOOLEAN
      ;

```

```

charval
    : CHARV
260     ;

identifier
    : ID
    ;

265 CHARV
    : APOSTROPHE (LETTER|UNDERSCORE) APOSTROPHE
    ;

270 BOOLEAN
    : TRUE
    | FALSE
    ;

275 ID
    : LETTER (LETTER | DIGIT)*
    ;

NUMBER
280     : DIGIT+
    ;

COMMENT
285     : '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
    | '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
    ;

WS
    : (
        '\t'
290     | '\r'
        | '\n'
    ) {$channel=HIDDEN;}
    ;

295 fragment DIGIT
    : ( '0'..'9' )
    ;

300 fragment LOWER
    : ( 'a'..'z' )
    ;

305 fragment UPPER
    : ( 'A'..'Z' )
    ;

310 fragment LETTER
    : LOWER
    | UPPER
    ;

315 fragment TRUE
    : 'true'
    ;

320 fragment FALSE
    : 'false'
    ;

//EOF

```

9.2 ANTLR Checker specificatie

```

tree grammar SELMAChecker;

options {
    tokenVocab=SELMA;
    ASTLabelType=SELMATree;
    output=AST;
}

@header {
    package SELMA;
    import SELMA.SELMATree.SR_Type;
    import SELMA.SELMATree.SR_Kind;
    import SELMA.SELMATree.SR_Func;
}

// Alter code generation so catch-clauses get replaced with this action.
@rulecatch {
    catch (RecognitionException re) {
        /*
        if (node != null)
            System.err.println(
                String.format("Error on line %d:%d: %s", node
                    .getLine(),
                                node
                                    .getCharPositionInLine
                                        (),
                                            ,
                                                re.
                                                    getMessage
                                                        ()
                                                            )
                                                                );
                */
        throw re;
    }
}

@members {
    public SymbolTable<CheckerEntry> st = new SymbolTable<
        CheckerEntry>();

    public void matchType(Tree expectedType, SR_Type exprType) {
        matchType(((SELMATree) expectedType).getSelmaType(),
            exprType);
    }

    public void matchType(Tree expectedType, Tree exprType) {
        matchType(((SELMATree) expectedType).getSelmaType(),
            ((SELMATree) exprType).getSelmaType());
    }

    public void matchType(SR_Type expectedType, SR_Type exprType) {
        if (expectedType != exprType)
            throw new SELMAException(String.format(
                "Expected type %s, got type %s",
                expectedType,
                exprType));
    }
}

program
    : ^(node=BEGIN

```

```

55     {st.openScope();}
        compoundexpression
        {st.closeScope();}
        END)
        ;

compoundexpression //do not open and close scope here (IF/WHILE)
60     : ^(node=COMPOUND (declaration|expression_statement)+)
        {
            SELMATree e1 = (SELMATree)node.getChild(node.getChildCount()
            -1);
            if (e1.SR_type==SR_Type.VOID) {
                node.SR_type=SR_Type.VOID;
                node.SR_kind=null;
65            } else {
                node.SR_type=e1.SR_type;
                node.SR_kind=e1.SR_kind;
            }
70        }
        ;

expression_statement
75     : ^(node=EXPRESSION_STATEMENT expression)
        {
            SELMATree e1 = (SELMATree)node.getChild(node.getChildCount()
            -1);
            // System.err.println("..." + e1 + " " + e1.getLine());
            $node.SR_type = e1.SR_type;
            $node.SR_kind = e1.SR_kind;
80        }
        ;

declaration
85     : ^(node=VAR type id=ID)
        {
            st.enter($id, new CheckerEntry(((SELMATree) node.getChild(0)).
            getSelmaType(),
                                SR_Kind.VAR));
        }
        | ^(node=CONST type val id=ID)
90     {
            int type = node.getChild(0).getType();
            int val = node.getChild(1).getType();

            switch (type) {
95             case INT:
                if (val!=NUMBER) throw new SELMAException(id," Expecting int
                -value");
                st.enter($id, new CheckerEntry(SR_Type.INT, SR_Kind.CONST));
                break;
            case BOOL:
100             if (val!=BOOLEAN) throw new SELMAException(id," Expecting
                bool-value");
                st.enter($id, new CheckerEntry(SR_Type.BOOL, SR_Kind.CONST));
                break;
            case CHAR:
                if (val!=CHARV) throw new SELMAException(id," Expecting char
                -value");
105             st.enter($id, new CheckerEntry(SR_Type.CHAR, SR_Kind.CONST));
                break;
            }
        }
        | ^(FUNCDEF funcname=ID
110     {
            //enter as void
            st.enter($funcname, new CheckerEntry(SR_Type.VOID, SR_Kind.VAR,
            SR_Func.YES));
            st.openScope();

```



```

115     }
        (param=ID typ1=(INT|BOOL|CHAR)
    {
        st.addParamToFunc($funcname, param, $typ1);
    }
        )*
120     (
        ^ (node=FUNCRETURN type compoundexpression
            expression
        {
            SELMATree type = (SELMATree) node.getChild(0);
            SELMATree expr = (SELMATree) node.getChild(2);
125         st.retrieve($funcname).type = expr.SR_type;

            matchType(type, expr.SR_type);
        })
        | (compoundexpression))
130     {
        //scope of function
        st.closeScope();
    });

135 type
    : node=INT
    | node=BOOL
    | node=CHAR
    ;

140 val
    : node=NUMBER
    | node=CHARV
    | node=BOOLEAN
145     ;

expression
    : ^ (node=(MULT|DIV|MOD|PLUS|MINUS) expression expression)
    {
150     SELMATree e1 = (SELMATree) node.getChild(0);
        SELMATree e2 = (SELMATree) node.getChild(1);

        if (e1.SR_type != SR.Type.INT || e2.SR_type != SR.Type.INT) {
            throw new SELMAException(
155                 $node,
                String.format("Wrong types must be int (found %s and %s)", e1.
                    SR_type, e2.SR_type));
        }

        $node.SR_type = SR.Type.INT;

160     if (e1.SR_kind == SR.Kind.CONST && e2.SR_kind == SR.Kind.CONST)
        $node.SR_kind = SR.Kind.CONST;
        else
        $node.SR_kind = SR.Kind.VAR;
165     }

    | ^ (node=(RELS|RELSE|RELG|RELGE) expression expression)
    {
        SELMATree e1 = (SELMATree) node.getChild(0);
170     SELMATree e2 = (SELMATree) node.getChild(1);

        if (e1.SR_type!=SR.Type.INT || e2.SR_type!=SR.Type.INT)
            throw new SELMAException($node,"Wrong type must be int");
        $node.SR_type=SR.Type.BOOL;

175     if (e1.SR_kind==SR.Kind.CONST && e2.SR_kind==SR.Kind.CONST)
        $node.SR_kind=SR.Kind.CONST;
        else
        $node.SR_kind=SR.Kind.VAR;
    }

```

```

180     }

        | ^(node=(OR|AND) expression expression)
    {
185     SELMATree e1 = (SELMATree)node.getChild(0);
    SELMATree e2 = (SELMATree)node.getChild(1);

    if (e1.SR_type!=SR_Type.BOOL || e2.SR_type!=SR_Type.BOOL)
        throw new SELMAException($node,"Wrong type must be bool");
    $node.SR_type=SR_Type.BOOL;

190    if (e1.SR_kind==SR_Kind.CONST && e2.SR_kind==SR_Kind.CONST)
        $node.SR_kind=SR_Kind.CONST;
    else
        $node.SR_kind=SR_Kind.VAR;
195    }

        | ^(node=(RELE|RELNE) expression expression)
    {
200    SELMATree e1 = (SELMATree)node.getChild(0);
    SELMATree e2 = (SELMATree)node.getChild(1);

    if (e1.SR_type!=e2.SR_type || e1.SR_type==SR_Type.VOID)
        throw new SELMAException($node,"Types must match and can't be void")
    ;
    $node.SR_type=SR_Type.BOOL;

205    if (e1.SR_kind==SR_Kind.CONST && e2.SR_kind==SR_Kind.CONST)
        $node.SR_kind=SR_Kind.CONST;
    else
        $node.SR_kind=SR_Kind.VAR;
210    }

        | ^(node=(UPLUS|UMIN) expression)
    {
215    SELMATree e1 = (SELMATree)node.getChild(0);

    if (e1.SR_type!=SR_Type.INT)
        throw new SELMAException($node,"Wrong type must be int");
    $node.SR_type=SR_Type.INT;

220    $node.SR_kind=e1.SR_kind;
    }

        | ^(node=NOT expression)
    {
225    SELMATree e1 = (SELMATree)node.getChild(0);

    if (e1.SR_type != SR_Type.BOOL)
        throw new SELMAException(node, "Wrong type must be bool");

230    node.SR_type = SR_Type.BOOL;
    node.SR_kind = e1.SR_kind;
    }

        | ^(node=IF {st.openScope();} compoundexpression
235    THEN {st.openScope();} compoundexpression {st.closeScope()
        ;}
        (ELSE {st.openScope();} compoundexpression {st.closeScope()
        ;})?
        {st.closeScope();})
    {
240    SELMATree e1 = (SELMATree)node.getChild(0);
    SELMATree e2 = (SELMATree)node.getChild(2);
    SELMATree e3 = (SELMATree)node.getChild(4);

    if (e1.SR_type!=SR_Type.BOOL)
        throw new SELMAException(e1,"Expression must be boolean");

```

```

245     if (e3==null) { //no else
        $node.SR_type=SR.Type.VOID;
        $node.SR_kind=null;
    } else { // there is a else
250     if (e2.SR_type==e3.SR_type) {
        $node.SR_type=e3.SR_type;
        if (e2.SR_kind==SR.Kind.CONST && e3.SR_kind==SR.Kind.CONST)
            $node.SR_kind=SR.Kind.CONST;
        else
255         $node.SR_kind=SR.Kind.VAR;
    } else {
        $node.SR_type=SR.Type.VOID;
        $node.SR_kind=null;
    }
260 }
}

    | ^(node=WHILE {st.openScope();} compoundexpression { st.
        closeScope(); }
        DO {st.openScope();} compoundexpression {st.closeScope();}
265 OD) /*{st.closeScope();} */
{
    SELMATree e1 = (SELMATree)node.getChild(0);
    SELMATree e2 = (SELMATree)node.getChild(2);

270     if (e1.SR_type!=SR.Type.BOOL)
        throw new SELMAException(e1,"Expression must be boolean");

    $node.SR_type=SR.Type.VOID;
    $node.SR_kind=null;
275 }

    | ^(node=READ (id=ID
    {
280         if (st.retrieve($id).kind!=SR.Kind.VAR)
            throw new SELMAException($id,"Must be a variable");
    })+
    {
        if ($node.getChildCount() == 1) {
285             $node.SR_type = st.retrieve(node.getChild(0)).type;
            $node.SR_kind = SR.Kind.VAR;
        } else {
            $node.SR_type = SR.Type.VOID;
            $node.SR_kind = null;
290        }
    }
    )

    | ^(node=PRINT expression+)
295 {
    for (int i=0; i<((SELMATree)node).getChildCount(); i++){
        if (((SELMATree)node).getChild(i)).SR_type == SR.Type.VOID)
            throw new SELMAException($node, "Can not be of type void");
    }
300     if ($node.getChildCount() == 1){
        $node.SR_type = ((SELMATree) node).getChild(0).SR_type;
        $node.SR_kind = SR.Kind.VAR;
    } else {
        $node.SR_type = SR.Type.VOID;
305        $node.SR_kind = null;
    }
}
-> ^(PRINT expression)+

310 | ^(node=FUNCTION ID expression*)
{

```

```

//retrieve function (if existent)
SELMATree func = (SELMATree)$node;
CheckerEntry entry = st.retrieve($ID);
315 $node.SR_type=entry.type;
    $node.SR_kind=entry.kind;

//matchparamlists
//same length?
320 int argc = func.getChildCount()-1;
    if (entry.params.size() != argc)
        throw new SELMAException(node, String.format(
            "\%s takes \%d arguments (\%d given)", $ID.text, entry.
                params.size(), argc));

//every entry matches?
325 for (int i=1; i<func.getChildCount(); i++){
    SELMATree expr = (SELMATree)func.getChild(i);
    if (expr.SR_type != entry.params.get(i-1).type)
        throw new SELMAException(expr,"Param is not of the right type");
}
330 }

| ^(node=BECOMES expression expression)
{
335 SELMATree e1 = (SELMATree)node.getChild(0);
    SELMATree e2 = (SELMATree)node.getChild(1);
    if (e1.getType()!=ID)
        throw new SELMAException(e1,"Must be a identifier");

340 CheckerEntry ident = st.retrieve(e1);

    if (ident.kind!=SR_Kind.VAR)
        throw new SELMAException(e1,"Must be a variable");
    if (ident.type!=e2.SR_type)
345 throw new SELMAException(e1,"Right side must be the same type "+
        ident.type+"/"+e2.SR_type);

    $node.SR_type=ident.type;
    $node.SR_kind=SR_Kind.VAR;
}

350 | ^(node=LCURLY {st.openScope();} compoundexpression {st.
    closeScope();} RCURLY)
{
    SELMATree e1 = (SELMATree) node.getChild(0);
    $node.SR_type = e1.SR_type;
355 $node.SR_kind = e1.SR_kind;
}

| node=NUMBER
{
360 $node.SR_type=SR_Type.INT;
    $node.SR_kind=SR_Kind.CONST;
}

| node=BOOLEAN
365 {
    $node.SR_type=SR_Type.BOOL;
    $node.SR_kind=SR_Kind.CONST;
}

370 | node=CHARV
{
    $node.SR_type=SR_Type.CHAR;
    $node.SR_kind=SR_Kind.CONST;
}
375 | node=ID

```

```
380 {
    CheckerEntry entry = st.retrieve($node);
    $node.SR_type=entry.type;
    $node.SR_kind=entry.kind;
  }
;
```

9.3 ANTLR Codegenerator specificatie

```
tree grammar SELMACompiler;

options {
    language = Java;
5   output = template;
    tokenVocab = SELMA;
    ASTLabelType = SELMATree;
}

10 @header {
    package SELMA;
    import SELMA.SELMA;
    import SELMA.SELMATree.SR_Type;
    import SELMA.SELMATree.SR_Kind;
15   import SELMA.SELMATree.SR_Func;

    import java.lang.StringBuilder;
}

20 @rulecatch {
    catch (RecognitionException re) {
        throw re;
    }
}

25 @members {
    public SymbolTable<CompilerEntry> st = new SymbolTable<CompilerEntry>
        >();

    int curStackDepth;
    int maxStackDepth;
30   int labelNum = 0;

    class StackDepthLabelCounter {
        public int curStackDepth;
        public int maxStackDepth;
35         public int labelNum;
        public int nextAddr;
        public int localCount;
    }

40   Stack<StackDepthLabelCounter> stack = new Stack<
        StackDepthLabelCounter>();

    private void incrStackDepth() {
        if (curStackDepth > maxStackDepth)
45         maxStackDepth = curStackDepth;
    }

    private void enterFuncScope() {
        StackDepthLabelCounter o = new StackDepthLabelCounter();
50         o.curStackDepth = curStackDepth;
        o.maxStackDepth = maxStackDepth;
        o.labelNum = labelNum;
        o.nextAddr = st.nextAddr;
        o.localCount = st.localCount;

55         stack.push(o);

        st.openScope();
        curStackDepth = maxStackDepth = labelNum = st.localCount = 0;
60         st.nextAddr = 0;
    }

    private void leaveFuncScope() {
        StackDepthLabelCounter o = stack.pop();
}
```

```

65         st.closeScope();
           curStackDepth = o.curStackDepth;
           maxStackDepth = o.maxStackDepth;
           labelNum = o.labelNum;
           st.nextAddr = o.nextAddr;
70         st.localCount = o.localCount;
       }

       private String getTypeDenoter(SR.Type type) {
           return st.getTypeDenoter(type, false);
75       }

       private String getTypeDenoter(SR.Type type, boolean printing) {
           return st.getTypeDenoter(type, printing);
       }
80   }

   program
   : ^(node=BEGIN {st.openScope();} compoundexpression END)
   { SELMATree expr = (SELMATree) $node.getChild(0);
85   int localsCount = st.getLocalsCount();
     List<String> globalVariableFields = st.getAllLocalVariablesWithTypes
       ();
     st.closeScope();
   }
   -> program(instructions={$compoundexpression.st},
90         source_file={SELMA.inputFilename},
           stack_limit={maxStackDepth + 3}, // +3 for print
           locals_limit={localsCount + 1}, // +1 for the String[] argv
           parameter
           fields={globalVariableFields},
           pop={expr.SR_type != SR.Type.VOID})
95   ;

   compoundexpression
   : ^(node=COMPOUND (s+=declaration | s+=expression_statement)+)
   -> compound(instructions={$s}, line={node.getLine()}, pop={$node.
100   SR_type != SR.Type.VOID})
   ;

   declaration
   : ^(node=VAR INT id=ID)
     {st.enter($id, new CompilerEntry(SR.Type.INT, SR.Kind.VAR, st.nextAddr
105   ()); }
     //-> declareVar(id={$id.text}, type="INT", addr={st.nextAddr()-1})

     | ^(node=VAR BOOL id=ID)
     {st.enter($id, new CompilerEntry(SR.Type.BOOL, SR.Kind.VAR, st.
110   nextAddr())); }
     //-> declareVar(id={$id.text}, type="BOOL", addr={st.nextAddr()-1})

     | ^(node=VAR CHAR id=ID)
     {st.enter($id, new CompilerEntry(SR.Type.CHAR, SR.Kind.VAR, st.
115   nextAddr())); }
     //-> declareVar(id={$id.text}, type="CHAR", addr={st.nextAddr()-1})

     // store the const at a address? LOAD Or just copy LOADL?
     | ^(node=CONST INT val=NUMBER (id=ID)+)
     {st.enter($id, new CompilerEntry(SR.Type.INT, SR.Kind.CONST, 0).setVal(
120   $val.text)); }
     //-> declareConst(id={$id.text}, val={$val.text}, type="integer",
           addr={st.nextAddr()-1})

     | ^(node=CONST type=BOOL val=BOOLEAN id=ID)
     {st.enter($id, new CompilerEntry(SR.Type.BOOL, SR.Kind.CONST, 0).
125   setBool($val.text)); }
     //-> declareConst(id={$id.text}, val=({$val.text.equals("true")
           ? "1" : "0"}, type="boolean", addr={st.nextAddr()}))

```

```

125 | ^ (node=CONST CHAR val=CHARV (id=ID)+)
    { char c = $val.text.charAt(1);
      st.enter($id, new CompilerEntry(SR_Type.CHAR, SR_Kind.CONST, 0).
        setChar(c));
    }
    //-> declareConst(id={$id.text}, val={{(int) c}, type={"character"}},
      addr={st.nextAddr()-1})

130 | ^ (node=FUNCDEF funcname=ID
    {
      CompilerEntry funcentry = new CompilerEntry(
        SR_Type.VOID, SR_Kind.VAR, 0, SR_Func.YES);
      st.enter($funcname, funcentry);
135 | enterFuncScope();
      int paramCount = 0;
      StringBuilder signatureBuilder = new StringBuilder("");
      //List<String> paramTypeDenoters = new ArrayList<String>();
    } (param=ID typ1=(INT|BOOL|CHAR)
140 | {
      SELMATree type1 = (SELMATree) $node.getChild(++paramCount * 2);
      signatureBuilder.append(getTypeDenoter(type1.getSelmaType()));
      //paramTypeDenoters.add(getTypeDenoter(type1.getSelmaType()));
      st.addParamToFunc($funcname, param, type1);
145 | })*
    ( ^ (return_node=FUNCRETUR (INT|BOOL|CHAR) (body+=compoundexpression)
      retexpr=expression)
    | (body+=compoundexpression)
    )
    {
150 | SELMATree funcbody;
      int stackLimit = maxStackDepth + 3;
      int localsLimit = st.getLocalsCount();

      signatureBuilder.append(" ");
155 | if ($return_node == null) {
      funcbody = (SELMATree) $node.getChild(paramCount * 2 + 1);
      signatureBuilder.append("V");
    } else {
160 | funcbody = (SELMATree) $return_node.getChild(1);
      SELMATree returnType = (SELMATree) $return_node.getChild(0);
      signatureBuilder.append(getTypeDenoter(returnType.
        getSelmaType()));
    }
    leaveFuncScope();
165 | String signature = signatureBuilder.toString();
      funcentry.signature = signature;
    })
    -> function(funcname={$funcname.text},
170 | body={$body},
      signature={signature},
      return_expression={$retexpr.st},
      is_void={signature.endsWith("V")},
      pop={funcbody.SR_type != SR_Type.VOID},
175 | stack_limit={stackLimit},
      locals_limit={localsLimit + 1},
      line={$node.getLine()})
    ;

180 | expression_statement
    : ^ (node=EXPRESSION.STATEMENT e1=expression) { curStackDepth--; }
    -> exprStat(e1={e1.st}, line={$node.getLine()}, pop={$node.SR_type !=
      SR_Type.VOID})
    ;

185 | expression

```



```

//double arg expression
: ^(node=MULT e1=expression e2=expression) { curStackDepth--; }
-> biExpr(e1={$e1.st}, e2={$e2.st}, instr={"imul"}, line={node.getLine() }, op={"*"})

190 | ^(node=DIV e1=expression e2=expression) { curStackDepth--; }
-> biExpr(e1={$e1.st}, e2={$e2.st}, instr={"idiv"}, line={node.getLine() }, op={"/"})

| ^(node=MOD e1=expression e2=expression) { curStackDepth--; }
-> biExpr(e1={$e1.st}, e2={$e2.st}, instr={"irem"}, line={node.getLine() }, op={"%"})

195 | ^(node=PLUS e1=expression e2=expression) { curStackDepth--; }
-> biExpr(e1={$e1.st}, e2={$e2.st}, instr={"iadd"}, line={node.getLine() }, op={"+"})

| ^(node=MINUS e1=expression e2=expression) { curStackDepth--; }
200 -> biExpr(e1={$e1.st}, e2={$e2.st}, instr={"isub"}, line={node.getLine() }, op={"-"})

| ^(node=OR e1=expression e2=expression) { curStackDepth--; }
-> biExpr(e1={$e1.st}, e2={$e2.st}, instr={"ior"}, line={node.getLine() }, op={"or"})

| ^(node=AND e1=expression e2=expression) { curStackDepth--; }
205 -> biExpr(e1={$e1.st}, e2={$e2.st}, instr={"iand"}, line={node.getLine() }, op={"and"})

| ^(node=RELS e1=expression e2=expression) { curStackDepth--; }
-> biExprJump(e1={$e1.st}, e2={$e2.st}, instr={"if.icmplt"}, line={node.
210 getLine() },
op={"<"}, label_num1={labelNum++}, label_num2={labelNum
++})

| ^(node=RELSE e1=expression e2=expression) { curStackDepth--; }
-> biExprJump(e1={$e1.st}, e2={$e2.st}, instr={"if.icmple"}, line={node.
getLine() },
op={"<="}, label_num1={labelNum++}, label_num2={labelNum
++})

215 | ^(node=RELG e1=expression e2=expression) { curStackDepth--; }
-> biExprJump(e1={$e1.st}, e2={$e2.st}, instr={"if.icmpgt"}, line={node.
getLine() },
op={">"}, label_num1={labelNum++}, label_num2={labelNum
++})

| ^(node=RELGE e1=expression e2=expression) { curStackDepth--; }
220 -> biExprJump(e1={$e1.st}, e2={$e2.st}, instr={"if.icmpge"}, line={node.
getLine() },
op={">="}, label_num1={labelNum++}, label_num2={labelNum
++})

| ^(node=RELE e1=expression e2=expression) { curStackDepth--; }
225 -> biExprJump(e1={$e1.st}, e2={$e2.st}, instr={"if.icmpeq"}, line={node.
getLine() },
op={"="}, label_num1={labelNum++}, label_num2={labelNum
++})

| ^(node=RELNE e1=expression e2=expression) { curStackDepth--; }
-> biExprJump(e1={$e1.st}, e2={$e2.st}, instr={"if.icmpne"}, line={node.
230 getLine() },
op={"!="}, label_num1={labelNum++}, label_num2={labelNum
++})

//single arg expression
| ^(UPLUS e1=expression)
{$st=$e1.st;}

```

```

235 | ^ (node=UMIN e1=expression)
-> uExpr(e1={$e1.st}, instr={"ineg"}, line={node.getLine()}, op={"-"})

| ^ (node=NOT e1=expression)
240 -> not(e1={$e1.st}, line={node.getLine()},
      label_num1={labelNum++}, label_num2={labelNum++})

//CONDITIONAL
| ^ (node=IF { st.openScope(); } ec1=compoundexpression { st.closeScope
245 (); } THEN
      { st.openScope(); } ec2=compoundexpression { st.closeScope
      (ELSE { st.openScope(); } ec3=compoundexpression { st.closeScope
      ()); }?)
      { boolean ec3NotEmpty = $ec3.st != null;
        SELMATree expr2 = (SELMATree) node.getChild(2);
        SELMATree expr3 = null;
250 if (ec3NotEmpty)
          expr3 = (SELMATree) node.getChild(4);
      }
-> if(ec1={$ec1.st}, ec2={$ec2.st}, ec3={$ec3.st}, label_num1={labelNum
    ++},
    label_num2={ec3NotEmpty ? labelNum++ : 0}, ec3_not_empty={
    ec3NotEmpty},
255 pop1={$node.SR_type == SR.Type.VOID && expr2.SR_type != SR.Type.
    VOID},
    pop2={ec3NotEmpty && $node.SR_type == SR.Type.VOID && expr3.
    SR_type != SR.Type.VOID})

| ^ (node=WHILE
      { st.openScope(); } ec1=compoundexpression { st.closeScope(); } DO
260 { st.openScope(); } ec2=compoundexpression { st.closeScope(); } OD
      )
      { SELMATree expr2 = (SELMATree) node.getChild(2);
        boolean pop = expr2.SR_type != SR.Type.VOID;
        if (pop)
          curStackDepth--;
265 }
-> while(ec1={$ec1.st}, ec2={$ec2.st}, pop={pop},
      label_num1={labelNum++}, label_num2={labelNum++})

//IO
270 | ^ (node=READ ID+)
      /*
      {
275 CompilerEntry entry = st.retrieve($id);
      })
-> readSingle(id={$id.text}, addr={entry.addr},
      is_bool={entry.type == SR.Type.BOOL},
      is_int={entry.type == SR.Type.INT},
      dup_top={$node.SR_type != SR.Type.VOID},
280 is_global={entry.level == 0},
      type_denoter={getTypeDenoter(entry.type)})

      */
      { boolean isExpr = $node.SR_type != SR.Type.VOID;
        List<Integer> addrs = new ArrayList<Integer>();
285 List<Boolean> isBool = new ArrayList<Boolean>();
        List<Boolean> isInt = new ArrayList<Boolean>();
        List<Boolean> globals = new ArrayList<Boolean>();
        List<String> ids = new ArrayList<String>();

        for (int i = 0; i < $node.getChildCount(); i++) {
          SELMATree child = (SELMATree) $node.getChild(i);
          CompilerEntry entry = st.retrieve(child);
          addrs.add(entry.addr);
          isBool.add(child.SR_type == SR.Type.BOOL);
290

```

```

295         isInt.add(child.SR_type == SR.Type.INT);
           globals.add(entry.level == 0);
           ids.add(child.getText());
       }
300   -> read(ids={ids}, addrs={addrs}, dup_top={isExpr},
           is_bool={isBool}, is_int={isInt},
           globals={globals}, line={node.getLine()})

| ^ (node=PRINT (exprs+=expression)+)
305 {
    boolean isExpr = $node.SR_type != SR.Type.VOID;
    int childCount = ((SELMATree) node).getChildCount();
    List<Integer> labelNums1 = new ArrayList<Integer>();
    List<Integer> labelNums2 = new ArrayList<Integer>();
310    List<String> typeDenoters = new ArrayList<String>();
    List<Boolean> exprIsBool = new ArrayList<Boolean>();

    if (!isExpr)
        curStackDepth -= childCount;
315    for (int i = 0; i < childCount; i++) {
        SELMATree child = (SELMATree) $node.getChild(i);
        boolean isBool = child.SR_type == SR.Type.BOOL;
        if (isBool) {
320            labelNums1.add(labelNum++);
            labelNums2.add(labelNum++);
        } else {
            labelNums1.add(0);
            labelNums2.add(0);
325        }
        typeDenoters.add(getTypeDenoter(child.SR_type, true));
        exprIsBool.add(isBool);
    }
330   -> print(exprs={$exprs}, type_denoters={typeDenoters}, dup_top={
        isExpr},
        expr_is_bool={exprIsBool},
        label_nums1={labelNums1}, label_nums2={labelNums2}, line={
            $node.getLine()})

| ^ (node=FUNCTION id=ID (exprs+=expression)*)
335   -> funccall(id={$id.text}, signature={st.retrieve($id).signature},
        exprs={$exprs})
//ASSIGN
| ^ (BECOMES node=ID e1=expression)
{
    CompilerEntry entry = st.retrieve(node);
340    boolean isConst = node.SR_kind == SR.Kind.CONST;
    boolean isGlobal = false;
    String typeDenoter = getTypeDenoter(entry.type);

    if (entry.level == 0) {
345        isGlobal = true;
    }
}
    -> assign(id={$node.text},
              type={$node.type},
350              addr={st.retrieve($node).addr},
              e1={$e1.st},
              is_global={isGlobal},
              type_denoter={typeDenoter})

355 //closedcompound
| ^ (node=LCURLY {st.openScope();} cmp=compoundexpression {st.
    closeScope();} RCURLY)
    -> compound(instructions={$cmp.st}, line={$node.getLine()}, pop
        ={false})

```

```

360 //VALUES
    | node=NUMBER { incrStackDepth();
                    int num = Integer.parseInt($node.text); }
    -> loadNum(val={$node.text}, iconst={num >= -1 && num <= 5}, bipush
        ={num >= -128 && num <= 127})

365 | node=BOOLEAN { incrStackDepth(); }
    -> loadNum(val={($node.text.equals("true")) ? 1 : 0}, iconst={true})

    | node=CHARV { incrStackDepth();
                    char c = $node.text.charAt(1); }
    //-> loadNum(val={(int) c}, iconst={false}, bipush={true})
    -> loadChar(val={(int) c}, char={$node.text}, line={$node.getLine()
        })

370 | node=ID
    {
        incrStackDepth();
        CompilerEntry entry = st.retrieve(node);
375         boolean isConst = node.SR_kind == SR.Kind.CONST;
        boolean isGlobal = false;
        String typeDenoter = getTypeDenoter(entry.type);

        if (entry.level == 0) {
380             isGlobal = true;
        }
    }
    -> loadVal(id={$node.text}, addr={entry.addr}, val={entry.val},
        is_const={isConst},
        is_global={isGlobal}, type_denoter={typeDenoter})
385 ;

```

9.4 ANTLR Codegenerator Stringtemplate specificatie

```
//SELMA string template
group SELMA;

5  program(instructions , fields , source_file , stack_limit , locals_limit , pop) ::= <<
    .source <source_file>
    .class public Main
    .super java/lang/Object
    .field public static scanner_field Ljava/util/Scanner;

10  <fields : { f | .field public static <f> }; separator="\n">

    .method public \<init\>()V
        aload_0
        invokespecial java/lang/Object/\<init\>()V
        return
    .end method

15  .method public static main([Ljava/lang/String;)V
    .limit stack <stack_limit>
    .limit locals <locals_limit>
    new java/util/Scanner
    dup
    getstatic java/lang/System/in Ljava/io/InputStream;
    invokespecial java/util/Scanner/\<init\>(Ljava/io/InputStream;)V
    putstatic Main/scanner_field Ljava/util/Scanner;

25  <instructions>

    <if (pop)>
        pop
    <endif>

    return
    .end method
    >>

    compound(instructions , line , pop) ::= <<
40  .line <line>
```

```

45 <instructions; separator="\n">
    <if (pop)>
        removeLastInstruction ; line <line>
    <endif>
>>

    expr (expr) ::= <<
        <expr>
    >>

50    exprStat (el, pop, line) ::= <<
        .line <line>
        <el>
    <if (pop)>
        pop
    <endif>
>>

55    //Calculations
    uExpr (el, instr, line, op) ::= <<
        .line <line>
        <el>
        <instr>
    >>

60    not (el, label_num1, label_num2, line) ::= <<
        .line <line>
        <el>
        <instr>
    >>

65    ifeq L<label_num1>
        iconst_0
        goto L<label_num2>
    L<label_num1>;
    L<label_num2>;
    >>

70    biExpr (el, e2, instr, line, op) ::= <<
        .line <line>
        <el>
        <e2>
        <instr>
    >>

80
>>

```

```

85 biExprJump(e1, e2, instr, label_num1, label_num2, line, op) ::= <<
    .line <line>
    <e1>
    <e2> <instr> L<label_num1>          ; e1 <op> e2
        iconst_0
        goto L<label_num2>
    L<label_num1>:
        iconst_1
    L<label_num2>:
        >>
95 //Declare
    declareConst(id, val, type, addr) ::= <<
        ldc <val>
        istore <addr>
100 >>

    declareVar(id, type, addr) ::= <<
105 >>

    //Load
    loadNum(val, iconst, bipush) ::= <<
110 <if (iconst)>
        iconst_<val>
    <elseif (bipush)>
        bipush <val>
115 <else>
        ldc <val>
    <endif>
    >>

120 loadVal(id, addr, val, is_const, is_global, type-denoter) ::= <<
    <if (is_const)>
        ldc <val>
        ; load constant <id>
    <elseif (is_global)>
        getstatic Main/<id> <type-denoter> ; load global <id>

```

```

125 <else>
      iload <addr>                ; load <id> from <addr>
    <endif>
  >>
130
  loadChar(val, char, line) ::= <<
    .line <line>
    bipush <val>                ; ldc <char>
  >>
135
  //Assign
  assign(id, type, addr, el, is_global, type_denoter) ::= <<
    <el>
    dup
    <if (is_global)>
      putstatic Main/<id> <type_denoter>
    <else>
      istore <addr>              ; store el in <id>
    <endif>
  >>
145
  >>
150
  read(ids, addrs, dup_top, is_bool, is_int, globals, line) ::= <<
    .line <line>
    <ids, addrs, is_bool, is_int, globals :
      { id, a, b, i, g | <readSingle(id=id, addr=a,
155         is_bool=b, is_int=i,
         dup_top=dup_top, is_global=g
         ) > }; separator="\n">
  >>
160
  readSingle(id, addr, is_bool, is_int, dup_top, is_global) ::= <<
    getstatic Main/scanner_field Ljava/util/Scanner;
    <if (is_bool)>
      invokevirtual java/util/Scanner/nextBoolean()Z
165
    <elseif (is_int)>

```



```

170         invokevirtual java/util/Scanner/nextInt() I
        <else>
        invokevirtual java/util/Scanner/nextByte() I
        <endif>
        <if (dup_top)>
        dup
175        <endif>
        <if (is_global)>
        <if (is_bool)>
        putstatic Main/<id> I
        <elseif (is_int)>
        putstatic Main/<id> I
        <else>
        putstatic Main/<id> C
180        <endif>
        <else>
        istore <addr>
185        <endif>
        <endif>
        >>
190
        print(exprs, type.denoters, dup_top, expr.is_bool, label_nums1, label_nums2, line) ::= <<
        .line <line>
        <exprs, type.denoters, expr.is_bool,
195        label_nums1, label_nums2 : { e, t, b, L1, L2 | < printSingle(expr=e,
        type.denoter=t,
        dup_top=dup_top,
        is_bool=b,
        label_num1=L1,
        label_num2=L2) > }>
200        >>

205        printSingle(expr, type.denoter, dup_top, is_bool, label_num1, label_num2) ::= <<
        <expr>
        <if (dup_top)>
        dup

```

```

210 <endif>
    <if (is_bool)>
        ifeq L<label_num1>
            ldc "true"
            goto L<label_num2>
        L<label_num1>:
            ldc "false"
        L<label_num2>:
    <endif>

215     getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println(<type_denoter>)V
    >>
220 >>
225 //conditionals
    if (ec1, ec2, ec3, label_num1, label_num2, ec3_not_empty, pop1, pop2) := <<
    <ec1> ifeq L<label_num1> ; e1 is false
    <ec2> ; e2 if true expression
    <if (pop1)>
        pop
    <endif>
    <if (ec3_not_empty)>
        goto L<label_num2>
    <endif>
    L<label_num1>:
    <if (ec3_not_empty)>
    <ec3> ; e3 if false expression
    <if (pop2)>
        pop ; pop
    <endif>
    L<label_num2>:
    <endif>
    >>
230 >>
235 while(ec1, ec2, label_num1, label_num2):=<<
240
245
250

```

```

255 L<label_num1>:
    <ec1>
    ifeq L<label_num2>
    <ec2>
    <if (pop)>
        pop
    <endif>
    goto L<label_num1>
    L<label_num2>;
>>
    function (function, signature, body, return-expression,
        pop, locals_limit, stack_limit, line) ::= <<
    \<method\>
    .method public static <function><signature>
    .limit stack <stack_limit>
    .limit locals <locals_limit>
    .line <line>
    <body; separator="\n\n">
    <if (pop)>
        pop
    <endif>
    <return-expression>
    <if (is-void)>
        return
    <else>
        ireturn
    <endif>
    .end method
    \</method\>
    >>
    funccall (id, signature, exprs) ::= <<
    <exprs; separator="\n">
    invokestatic Main/<id><signature>

```



9.5 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.

9.5.1 SELMA-code van pasen

```

/*
Methode van Gau

De Duitse geleerde Carl Friedrich Gau publiceerde in 1800 een
wiskundig algoritme waarmee de paasdatum voor een willekeurig jaar
berekend kan worden. Gau maakte toch een fout: hij hield niet
goed rekening met de maancorrectie, zodat bijvoorbeeld zijn
paasdatum voor 4200 uitkomt op 13 april in plaats van 20 april. De
methode van Gau loopt als volgt:
5 */

const maxyear: integer = 2099;

function checkjaar(jaar: integer): boolean {
10   var ok: boolean;
   if jaar < 0; then
       print('j','e','z','u','s','-','m','o','e','t','-','z','i',
           ',','j','n','-','g','e','b','o','r','e','n');
   else
       if jaar > maxyear; then
15           print('h','e','b','t','-','g','e','d','u','l','d',
               ');
       fi;
   fi;
   ok := (jaar>=0)&& jaar<=maxyear;
   return ok;
20 };

print('H','a','l','l','o','-','v','a','n','-','w','e','l','k','-','j','a',
    ',','a','r','-','w','i','l','-','j','e','-','d','e','-','p','a','a',
    's','d','a','t','u','m','-','w','e','t','e','n');

var jaar: integer;
25 read(jaar);

const negentien: integer = 19;

if @checkjaar(jaar.); then
30   // Bepaal het gulden getal:
   // Deel het jaartal door 19, neem de rest en tel er 1 bij op (zoals
   Dionysius). Noem dit getal G. Voor het jaar 1991 geldt G = 16.
   var G: integer;
   G := jaar/negentien;
   G := G+1;
35   // Bepaal het eeuwtaal:
   // Geheeldeel het jaartal door 100 en tel daar 1 bij op. Noem dit
   getal C. Voor het jaar 1991 geldt C = 20.
   var C: integer;
   C := jaar;
   C := C/100 + 1;
40   // Corrigeer vervolgens voor jaren die geen schrikkeljaar zijn:
   // Vermenigvuldig C met 3, geheeldeel het resultaat door 4 en trek er
   12 van af. Noem dit getal X. Voor de twintigste en eenentwintigste
   eeuw geldt X = 3.
   var X: integer;
45   const stap: integer = 1;
   const twaalf: integer = 12;
   function nest(stap,w: integer): integer {
       var returnvalue: integer;
       if stap==1; then
50           returnvalue := 3*w;
       else
           if stap == 2; then
               returnvalue := w/4;

```

```

55         var loop: integer;
           loop := 1;
           while loop <= twaalf; do
               returnvalue := returnvalue - 1;
               loop := loop + 1;
           od;
60         fi;
           fi;
           return returnvalue;
       };
       X := @nest(stap+1,@nest(stap,C,)) ;
65 // Maancorrectie:
//   Neem 8 maal C, tel er 5 bij op, deel het geheel door 25 en trek er
   5 vanaf. Noem dit getal Y. Voor de twintigste en eenentwintigste
   eeuw geldt: Y = 1.
       var Y: integer;
       if jaar >= 1900; then
70           Y := 1;
       else
           Y := (8*C)/25 - 5;
       fi;

75 // Zoek de zondag:
//   Vermenigvuldig het jaartal met 5, geheeldeel de uitkomst door 4,
   trek er X en 10 vanaf, en noem dit getal Z. Voor 1991 geldt: Z =
   2475.
       function nested(): integer {
           var Z: integer;
           Z := (((jaar*5)/4)-X)+-10;
80           function copy(jaar: integer): integer {
               42;
               return jaar;
           };
           Z := @copy(Z,);
85           return Z;
       };
       var Z: integer;
       Z := @nested();

90 // Bepaal de epacta:
//   11 maal G + 20 + Y. Trek daarvan X af, geheeldeel het resultaat
   door 30 en noem de rest E. Als E gelijk is aan 24, of als E gelijk
   is aan 25 en het gulden getal is groter dan 11, tel dan 1 bij E op.
   De Epacta voor 1991 is 14.
       var E,EE: integer;
       E := EE := ({const elf: integer = 11; elf*(G+20+Y);} - X) / 30;
       const mnoue: boolean = false;
95       if EE==24 || (E==25&&G>11) || mnoue; then
           E := 1+E;
       fi;

// Bepaal de volle maan:
100 //   Trek E af van 44. Noem dit getal N. Als N kleiner is dan 21, tel
   er dan 30 bij op. Voor 1991 geldt: N = 30
       var N: integer;
       const minus2: integer = 44;
       N := minus2-E;
       var tosmall: boolean;
105       tosmall := N<21;
       N := N+
           if tosmall; then
               30;
           else
110               0;
           fi;

//   Nu door naar zondag:

```

```

115 //      Tel Z en N op. Geheeldeel het resultaat door 7 en trek de rest af
      van N+7. Noem dit getal P. Voor 1991 geldt: P = 31.
      var P: integer;
      P := (N+7)-(Z+N)%7;

      //      Paasdatum: Als P groter is dan 31, trek er dan 31 vanaf, en de
      paasdatum valt in April. Anders is de paasdag P in Maart. Zo wordt
      voor 1991 gevonden 31 maart.
      var month: integer;
      var day: integer;
120      if P>31; then
          month := 4;
          day := P%31;
      else
125          month := 3;
          day := P;
      fi;

      function printdatum() {
130          print(day);
          var under: character;
          under := print('_');
          if month==3; then
              print('M','a','a','r','t');
135          else
              print('A','p','r','i','l');
          fi;
          print(under,jaar);
      };
140      @printdatum();
fi;

```


9.5.2 Jasmin-code van pasen

```

5  .source pasen/pasen.SELMA
   .class public Main
   .super java/lang/Object
   .field public static scanner_field Ljava/util/Scanner;
   .field public static jaar I
   .method public <init>()V
       aload_0
       invokespecial java/lang/Object/<init>()V
       return
   .end method
   .method public static main([Ljava/lang/String;)V
       .limit stack 3
       .limit locals 19
       new java/util/Scanner
       dup
       getstatic java/lang/System/in Ljava/io/InputStream;
       invokespecial java/util/Scanner/<init>(Ljava/io/InputStream;)V
       putstatic Main/scanner_field Ljava/util/Scanner;
       .line 7
       .line 22
       bipush 72
       getstatic java/lang/System/out Ljava/io/PrintStream;
       swap
       invokevirtual java/io/PrintStream/println(C)V
       .line 22
       bipush 97
       getstatic java/lang/System/out Ljava/io/PrintStream;
       swap

```

30	invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 108 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 108 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 111 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 95 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 118 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 97
35	
40	
45	
50	
55	

```

60      getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 110
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 95
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 119
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 101
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 108
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V

```

85	.line 22 bipush 107 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22
90	bipush 95 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22
95	bipush 106 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22
100	bipush 97 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22
105	bipush 97 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22
110	bipush 114 getstatic java/lang/System/out Ljava/io/PrintStream;

115	<pre> swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 95 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 119 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 105 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 108 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 95 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 </pre>
120	
125	
130	
135	
140	

```

bipush 106                ; ldc 'j'
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println (C)V
.line 22
bipush 101                ; ldc 'e'
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println (C)V
.line 22
bipush 95                 ; ldc '-'
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println (C)V
.line 22
bipush 100                ; ldc 'd'
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println (C)V
.line 22
bipush 101                ; ldc 'e'
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println (C)V
.line 22
bipush 95                 ; ldc '-'
getstatic java/lang/System/out Ljava/io/PrintStream;
swap

```

170	invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 112 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 97 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 97 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 115 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 100 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 22 bipush 97
175	; ldc 'p' ; ldc 'a' ; ldc 'a' ; ldc 's' ; ldc 'd' ; ldc 'a'
180	
185	
190	
195	

```

200      getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 116
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 117
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 109
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 95
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        .line 22
        bipush 119
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V

```

225	<pre> .line 22 bipush 101 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V </pre>
230	<pre> .line 22 bipush 116 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V </pre>
235	<pre> .line 22 bipush 101 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V </pre>
240	<pre> .line 22 bipush 110 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V </pre>
245	<pre> .line 25 getstatic Main/scanner_field Ljava/util/Scanner; invokevirtual java/util/Scanner/nextInt () I dup putstatic Main/jaar I pop </pre>
250	<pre> .line 29 getstatic Main/jaar I ; load global jaar </pre>

255	invokestatic Main/checkjaar (I)I
260	ifeq L19 ; e1 is false
	.line 32
	.line 33
	getstatic Main/jaar I ; load global jaar
	ldc 19 ; load constant negentien
	idiv ; e1 right hand for assignment
	dup
	istore 3 ; store e1 in G
	pop
	.line 34
	iload 3 ; load G from 3
	iconst_1
	iadd ; e1 right hand for assignment
	dup
	istore 3 ; store e1 in G
	pop
	.line 39
	getstatic Main/jaar I ; load global jaar
	; e1 right hand for assignment
	dup
	istore 4 ; store e1 in C
	pop
	.line 40
	iload 4
	bipush 100
	idiv
	iconst_1
280	

	iadd		; e1 right hand for assignment
	dup		
	istore 4		; store e1 in C
	pop		
285	.line 64		
	ldc 1		; load constant stap
	iconst_1		
	iadd		
	ldc 1		; load constant stap
	iload 4		; load C from 4
290	invokestatic Main/nest(II)I		
	invokestatic Main/nest(II)I		; e1 right hand for assignment
	dup		
	istore 5		; store e1 in X
	pop		
295	.line 69		
	getstatic Main/jaar I		; load global jaar
	ldc 1900		
	if_icmpge L0		; e1 >= e2
300	iconst_0		
	goto L1		
	L0:		
	iconst_1		
	L1:		
	ifeq L2		; e1 is false
305	.line 70		
	iconst_1		; e1 right hand for assignment

310	dup		
	istore 7		; store e1 in Y
	goto L3		
	L2:		
	.line 72		
	bipush 8		
315	iload 4		; load C from 4
	imul		
	bipush 25		
	idiv		
	iconst_5		
320	isub		; e1 right hand for assignment
	dup		
	istore 7		; store e1 in Y
	L3:		
	pop		
325	.line 88		
	invokestatic Main/nested()I		; e1 right hand for assignment
	dup		
	istore 9		; store e1 in Z
	pop		
330	.line 93		
	ldc 11		; load constant elf
	.line 93		
	iload 3		; load G from 3
	bipush 20		
	iadd		
335	iload 7		; load Y from 7

340	iadd		
	imul		
	iload 5		; load X from 5
	isub		
	bipush 30		
	idiv		; e1 right hand for assignment
	dup		
	istore 11		; store e1 in EE
345			; e1 right hand for assignment
	dup		
	istore 10		; store e1 in E
	pop		
	.line 95		
350	iload 11		; load EE from 11
	bipush 24		
	if_icmpeq L4		; e1 = e2
	iconst_0		
	goto L5		
355	L4:		
	iconst_1		
	L5:		
	.line 95		
	iload 10		; load E from 10
	bipush 25		
	if_icmpeq L6		; e1 = e2
360	iconst_0		
	goto L7		
	L6:		

365	iconst_1	
	L7:	
	.line 95	; load G from 3
	iload 3	
	bipush 11	
	if_icmpgt L8	
370	iconst_0	; e1 > e2
	goto L9	
	L8:	
	iconst_1	
375	L9:	
	iand	
	ior	
	ldc 0	; load constant mnop
	ior	
	ifeq L10	; e1 is false
380	.line 96	
	iconst_1	
	iload 10	; load E from 10
	iadd	; e1 right hand for assignment
	dup	
385	istore 10	; store e1 in E
	pop	
	L10:	
	.line 103	
	ldc 44	; load constant minus2
390	iload 10	; load E from 10
	isub	; e1 right hand for assignment

395	dup		
	istore 12		; store e1 in N
	pop		
	.line 105		
	iload 12		; load N from 12
	bipush 21		
	if_icmplt L11		; e1 < e2
400	iconst_0		
	goto L12		
	L11:		
	iconst_1		
	L12:		; e1 right hand for assignment
405	dup		
	istore 13		; store e1 in tosmall
	pop		
	.line 106		
	iload 12		; load N from 12
410	.line 107		
	iload 13		; load tosmall from 13
	ifeq L13		; e1 is false
	.line 108		
	bipush 30		
415	goto L14		
	L13:		
	.line 110		
	iconst_0		
	L14:		
420	iadd		; e1 right hand for assignment

	dup		
	istore 12		; store e1 in N
	pop		
425	.line 116		
	iload 12		; load N from 12
	bipush 7		
	iadd		
	.line 116		
	iload 9		; load Z from 9
430	iload 12		; load N from 12
	iadd		
	bipush 7		
	irem		
	isub		; e1 right hand for assignment
∞ ₄₃₅	dup		
	istore 14		; store e1 in P
	pop		
	.line 121		
	iload 14		; load P from 14
440	bipush 31		
	if_icmpgt L15		; e1 > e2
	iconst_0		
	goto L16		
	L15:		
445	iconst_1		
	L16:		
	ifeq L17		; e1 is false
	.line 122		

450	iconst_4	
	dup	; e1 right hand for assignment
	istore 15	; store e1 in month
	pop	
455	.line 123	
	iload 14	; load P from 14
	bipush 31	
	irem	; e1 right hand for assignment
	dup	
	istore 16	; store e1 in day
460	goto L18	
	L17:	
	.line 125	
	iconst_3	
	dup	; e1 right hand for assignment
465	istore 15	; store e1 in month
	pop	
	.line 126	
	iload 14	; load P from 14
		; e1 right hand for assignment
470	dup	
	istore 16	; store e1 in day
	L18:	
	pop	
475	.line 140	
	invokestatic Main/printdatum()V	

```

L19:      ; e2 if true expression
      return
    .end method
    .method public static checkjaar(I)I
    .limit stack 3
    .limit locals 3
    .line 9
    .line 10
    .line 11
    iload 0      ; load jaar from 0
    iconst_0
    if_icmplt L0 ; e1 < e2
    iconst_0
    goto L1
L0:      iconst_1
L1:      ifeq L5      ; e1 is false
    .line 12
    bipush 106
    getstatic java/lang/System/outLjava/io/PrintStream;
    swap
    invokevirtual java/io/PrintStream/println(C)V
    .line 12
    bipush 101
    getstatic java/lang/System/outLjava/io/PrintStream;
    swap

```

505	invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 122 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 117 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 115 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 95 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 109 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 111
510	
515	
520	
525	
530	

535	<pre> getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 101 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 116 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 95 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 122 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 12 bipush 105 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V </pre>
540	
545	
550	
555	
560	

```

.line 12
    bipush 106
    getstatic java/lang/System/out Ljava/io/PrintStream;
    swap
    invokevirtual java/io/PrintStream/println (C)V
.line 12
    bipush 110
    getstatic java/lang/System/out Ljava/io/PrintStream;
    swap
    invokevirtual java/io/PrintStream/println (C)V
.line 12
    bipush 95
    getstatic java/lang/System/out Ljava/io/PrintStream;
    swap
    invokevirtual java/io/PrintStream/println (C)V
.line 12
    bipush 103
    getstatic java/lang/System/out Ljava/io/PrintStream;
    swap
    invokevirtual java/io/PrintStream/println (C)V
.line 12
    bipush 101
    getstatic java/lang/System/out Ljava/io/PrintStream;
    swap
    invokevirtual java/io/PrintStream/println (C)V
.line 12
    bipush 98
    getstatic java/lang/System/out Ljava/io/PrintStream;

```

```

590 swap
    invokevirtual java/io/PrintStream/printLn (C)V
    .line 12
        bipush 111
        getstatic java/lang/System/out Ljava/io/PrintStream; ; ldc 'o'
    swap
595 invokevirtual java/io/PrintStream/printLn (C)V
    .line 12
        bipush 114
        getstatic java/lang/System/out Ljava/io/PrintStream; ; ldc 'r'
    swap
600 invokevirtual java/io/PrintStream/printLn (C)V
    .line 12
        bipush 101
        getstatic java/lang/System/out Ljava/io/PrintStream; ; ldc 'e'
    swap
605 invokevirtual java/io/PrintStream/printLn (C)V
    .line 12
        bipush 110
        getstatic java/lang/System/out Ljava/io/PrintStream; ; ldc 'n'
    swap
610 invokevirtual java/io/PrintStream/printLn (C)V
    .line 12
        goto L6
L5:
    .line 14
        iload 0
        ldc 2099
        ; load jaar from 0
        ; load constant maxyear

```

```

if_icmpgt L2      ; e1 > e2
iconst_0
goto L3
L2:
iconst_1
L3:
ifeq L4          ; e1 is false
.line 15
bipush 104
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println (C)V
.line 15
bipush 101
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println (C)V
.line 15
bipush 98
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println (C)V
.line 15
bipush 116
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println (C)V
.line 15

```

620

625

630

635

640

645	<pre> bipush 95 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 15 </pre>
650	<pre> bipush 103 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 15 </pre>
655	<pre> bipush 101 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 15 </pre>
660	<pre> bipush 100 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 15 </pre>
665	<pre> bipush 117 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println (C)V .line 15 </pre>
670	<pre> bipush 108 getstatic java/lang/System/out Ljava/io/PrintStream; swap </pre>

675	invokevirtual java/io/PrintStream/println(C)V .line 15 bipush 100 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println(C)V ; e2 if true expression L4: L6: .line 18 iload 0 iconst_0 if_icmpge L7 iconst_0 goto L8 L7: iconst_1 L8: .line 18 iload 0 ldc 2099 if_icmple L9 iconst_0 goto L10 L9: iconst_1 L10: ; e3 if false expression
680	
685	
690	
695	
700	

```

iand                ; e1 right hand for assignment
dup
istore 1            ; store e1 in ok
pop
iload 1             ; load ok from 1
ireturn
.end method
.method public static nest(II)I
.limit stack 3
.limit locals 5
.line 47
.line 48
.line 49
    iload 0          ; load stap from 0
    iconst_1
    if_icmpeq L0      ; e1 = e2
    iconst_0
    goto L1
L0:
    iconst_1
L1:
    ifeq L9           ; e1 is false
    .line 50
    iconst_3
    iload 1           ; load w from 1
    imul              ; e1 right hand for assignment
    dup
    istore 2          ; store e1 in returnvalue

```

730	pop		
	goto L10		
	L9:		
	.line 52		
	iload 0		; load stap from 0
	iconst_2		
735	if_icmpeq L2		; e1 = e2
	iconst_0		
	goto L3		
	L2:		
	iconst_1		
740	L3:		; e1 is false
	ifeq L8		
	.line 53		
	iload 1		; load w from 1
	iconst_4		
	idiv		; e1 right hand for assignment
745	dup		
	istore 2		; store e1 in returnvalue
	pop		
	.line 55		
	iconst_1		
750			; e1 right hand for assignment
	dup		
	istore 3		; store e1 in loop
	pop		
755	.line 56		
	L6:		

<pre> 760 765 770 775 780 </pre>	<pre> .line 56 iload 3 ldc 12 if_icmple L4 iconst_0 goto L5 L4: iconst_1 L5: ifeq L7 .line 57 iload 2 iconst_1 isub dup istore 2 pop .line 58 iload 3 iconst_1 iadd dup istore 3 pop goto L6 L7: L8: </pre>	<pre> ; load loop from 3 ; load constant twalf ; e1 <= e2 ; load returnvalue from 2 ; e1 right hand for assignment ; store e1 in returnvalue ; load loop from 3 ; e1 right hand for assignment ; store e1 in loop ; e2 if true expression </pre>
----------------------------------	---	--

```

785 L10:      iload 2      ; load returnvalue from 2
          ireturn
          .end method
          .method public static copy(I)I
          .limit stack 3
          .limit locals 2
          .line 80
          .line 81
          bipush 42
          pop
          iload 0      ; load jaar from 0
          ireturn
          .end method
          .method public static nested()I
          .limit stack 3
          .limit locals 3
          .line 77
          .line 78
          .line 79
          getstatic Main/jaar I ; load global jaar
          iconst_5
          imul
          iconst_4
          idiv
          iload 5      ; load X from 5
          isub
          .line 80
          .line 81
          .line 82
          .line 83
          .line 84
          .line 85
          .line 86
          .line 87
          .line 88
          .line 89
          .line 90
          .line 91
          .line 92
          .line 93
          .line 94
          .line 95
          .line 96
          .line 97
          .line 98
          .line 99
          .line 100
          .line 101
          .line 102
          .line 103
          .line 104
          .line 105
          .line 106
          .line 107
          .line 108
          .line 109
          .line 110
          .line 111
          .line 112
          .line 113
          .line 114
          .line 115
          .line 116
          .line 117
          .line 118
          .line 119
          .line 120
          .line 121
          .line 122
          .line 123
          .line 124
          .line 125
          .line 126
          .line 127
          .line 128
          .line 129
          .line 130
          .line 131
          .line 132
          .line 133
          .line 134
          .line 135
          .line 136
          .line 137
          .line 138
          .line 139
          .line 140
          .line 141
          .line 142
          .line 143
          .line 144
          .line 145
          .line 146
          .line 147
          .line 148
          .line 149
          .line 150
          .line 151
          .line 152
          .line 153
          .line 154
          .line 155
          .line 156
          .line 157
          .line 158
          .line 159
          .line 160
          .line 161
          .line 162
          .line 163
          .line 164
          .line 165
          .line 166
          .line 167
          .line 168
          .line 169
          .line 170
          .line 171
          .line 172
          .line 173
          .line 174
          .line 175
          .line 176
          .line 177
          .line 178
          .line 179
          .line 180
          .line 181
          .line 182
          .line 183
          .line 184
          .line 185
          .line 186
          .line 187
          .line 188
          .line 189
          .line 190
          .line 191
          .line 192
          .line 193
          .line 194
          .line 195
          .line 196
          .line 197
          .line 198
          .line 199
          .line 200
          .line 201
          .line 202
          .line 203
          .line 204
          .line 205
          .line 206
          .line 207
          .line 208
          .line 209
          .line 210
          .line 211
          .line 212
          .line 213
          .line 214
          .line 215
          .line 216
          .line 217
          .line 218
          .line 219
          .line 220
          .line 221
          .line 222
          .line 223
          .line 224
          .line 225
          .line 226
          .line 227
          .line 228
          .line 229
          .line 230
          .line 231
          .line 232
          .line 233
          .line 234
          .line 235
          .line 236
          .line 237
          .line 238
          .line 239
          .line 240
          .line 241
          .line 242
          .line 243
          .line 244
          .line 245
          .line 246
          .line 247
          .line 248
          .line 249
          .line 250
          .line 251
          .line 252
          .line 253
          .line 254
          .line 255
          .line 256
          .line 257
          .line 258
          .line 259
          .line 260
          .line 261
          .line 262
          .line 263
          .line 264
          .line 265
          .line 266
          .line 267
          .line 268
          .line 269
          .line 270
          .line 271
          .line 272
          .line 273
          .line 274
          .line 275
          .line 276
          .line 277
          .line 278
          .line 279
          .line 280
          .line 281
          .line 282
          .line 283
          .line 284
          .line 285
          .line 286
          .line 287
          .line 288
          .line 289
          .line 290
          .line 291
          .line 292
          .line 293
          .line 294
          .line 295
          .line 296
          .line 297
          .line 298
          .line 299
          .line 300
          .line 301
          .line 302
          .line 303
          .line 304
          .line 305
          .line 306
          .line 307
          .line 308
          .line 309
          .line 310
          .line 311
          .line 312
          .line 313
          .line 314
          .line 315
          .line 316
          .line 317
          .line 318
          .line 319
          .line 320
          .line 321
          .line 322
          .line 323
          .line 324
          .line 325
          .line 326
          .line 327
          .line 328
          .line 329
          .line 330
          .line 331
          .line 332
          .line 333
          .line 334
          .line 335
          .line 336
          .line 337
          .line 338
          .line 339
          .line 340
          .line 341
          .line 342
          .line 343
          .line 344
          .line 345
          .line 346
          .line 347
          .line 348
          .line 349
          .line 350
          .line 351
          .line 352
          .line 353
          .line 354
          .line 355
          .line 356
          .line 357
          .line 358
          .line 359
          .line 360
          .line 361
          .line 362
          .line 363
          .line 364
          .line 365
          .line 366
          .line 367
          .line 368
          .line 369
          .line 370
          .line 371
          .line 372
          .line 373
          .line 374
          .line 375
          .line 376
          .line 377
          .line 378
          .line 379
          .line 380
          .line 381
          .line 382
          .line 383
          .line 384
          .line 385
          .line 386
          .line 387
          .line 388
          .line 389
          .line 390
          .line 391
          .line 392
          .line 393
          .line 394
          .line 395
          .line 396
          .line 397
          .line 398
          .line 399
          .line 400
          .line 401
          .line 402
          .line 403
          .line 404
          .line 405
          .line 406
          .line 407
          .line 408
          .line 409
          .line 410
          .line 411
          .line 412
          .line 413
          .line 414
          .line 415
          .line 416
          .line 417
          .line 418
          .line 419
          .line 420
          .line 421
          .line 422
          .line 423
          .line 424
          .line 425
          .line 426
          .line 427
          .line 428
          .line 429
          .line 430
          .line 431
          .line 432
          .line 433
          .line 434
          .line 435
          .line 436
          .line 437
          .line 438
          .line 439
          .line 440
          .line 441
          .line 442
          .line 443
          .line 444
          .line 445
          .line 446
          .line 447
          .line 448
          .line 449
          .line 450
          .line 451
          .line 452
          .line 453
          .line 454
          .line 455
          .line 456
          .line 457
          .line 458
          .line 459
          .line 460
          .line 461
          .line 462
          .line 463
          .line 464
          .line 465
          .line 466
          .line 467
          .line 468
          .line 469
          .line 470
          .line 471
          .line 472
          .line 473
          .line 474
          .line 475
          .line 476
          .line 477
          .line 478
          .line 479
          .line 480
          .line 481
          .line 482
          .line 483
          .line 484
          .line 485
          .line 486
          .line 487
          .line 488
          .line 489
          .line 490
          .line 491
          .line 492
          .line 493
          .line 494
          .line 495
          .line 496
          .line 497
          .line 498
          .line 499
          .line 500
          .line 501
          .line 502
          .line 503
          .line 504
          .line 505
          .line 506
          .line 507
          .line 508
          .line 509
          .line 510
          .line 511
          .line 512
          .line 513
          .line 514
          .line 515
          .line 516
          .line 517
          .line 518
          .line 519
          .line 520
          .line 521
          .line 522
          .line 523
          .line 524
          .line 525
          .line 526
          .line 527
          .line 528
          .line 529
          .line 530
          .line 531
          .line 532
          .line 533
          .line 534
          .line 535
          .line 536
          .line 537
          .line 538
          .line 539
          .line 540
          .line 541
          .line 542
          .line 543
          .line 544
          .line 545
          .line 546
          .line 547
          .line 548
          .line 549
          .line 550
          .line 551
          .line 552
          .line 553
          .line 554
          .line 555
          .line 556
          .line 557
          .line 558
          .line 559
          .line 560
          .line 561
          .line 562
          .line 563
          .line 564
          .line 565
          .line 566
          .line 567
          .line 568
          .line 569

```

```

.line 79      ; -- bipush 10
      bipush 10
      ineg
      iadd
      dup
      istore 0
      pop
      .line 84
      iload 0
      invokestatic Main/copy(I)I
      dup
      istore 0
      pop
      iload 0
      ireturn
      .end method
      .method public static printdatum()V
      .limit stack 3
      .limit locals 2
      .line 129
      .line 130
      iload 16
      dup
      getstatic java/lang/System/out Ljava/io/PrintStream;
      swap
      invokevirtual java/io/PrintStream/println(I)V
      pop
      .line 132
      .line 133
      .line 134
      .line 135
      .line 136
      .line 137
      .line 138
      .line 139
      .line 140
      .line 141
      .line 142
      .line 143
      .line 144
      .line 145
      .line 146
      .line 147
      .line 148
      .line 149
      .line 150
      .line 151
      .line 152
      .line 153
      .line 154
      .line 155
      .line 156
      .line 157
      .line 158
      .line 159
      .line 160
      .line 161
      .line 162
      .line 163
      .line 164
      .line 165
      .line 166
      .line 167
      .line 168
      .line 169
      .line 170
      .line 171
      .line 172
      .line 173
      .line 174
      .line 175
      .line 176
      .line 177
      .line 178
      .line 179
      .line 180
      .line 181
      .line 182
      .line 183
      .line 184
      .line 185
      .line 186
      .line 187
      .line 188
      .line 189
      .line 190
      .line 191
      .line 192
      .line 193
      .line 194
      .line 195
      .line 196
      .line 197
      .line 198
      .line 199
      .line 200
      .line 201
      .line 202
      .line 203
      .line 204
      .line 205
      .line 206
      .line 207
      .line 208
      .line 209
      .line 210
      .line 211
      .line 212
      .line 213
      .line 214
      .line 215
      .line 216
      .line 217
      .line 218
      .line 219
      .line 220
      .line 221
      .line 222
      .line 223
      .line 224
      .line 225
      .line 226
      .line 227
      .line 228
      .line 229
      .line 230
      .line 231
      .line 232
      .line 233
      .line 234
      .line 235
      .line 236
      .line 237
      .line 238
      .line 239
      .line 240
      .line 241
      .line 242
      .line 243
      .line 244
      .line 245
      .line 246
      .line 247
      .line 248
      .line 249
      .line 250
      .line 251
      .line 252
      .line 253
      .line 254
      .line 255
      .line 256
      .line 257
      .line 258
      .line 259
      .line 260
      .line 261
      .line 262
      .line 263
      .line 264
      .line 265
      .line 266
      .line 267
      .line 268
      .line 269
      .line 270
      .line 271
      .line 272
      .line 273
      .line 274
      .line 275
      .line 276
      .line 277
      .line 278
      .line 279
      .line 280
      .line 281
      .line 282
      .line 283
      .line 284
      .line 285
      .line 286
      .line 287
      .line 288
      .line 289
      .line 290
      .line 291
      .line 292
      .line 293
      .line 294
      .line 295
      .line 296
      .line 297
      .line 298
      .line 299
      .line 300
      .line 301
      .line 302
      .line 303
      .line 304
      .line 305
      .line 306
      .line 307
      .line 308
      .line 309
      .line 310
      .line 311
      .line 312
      .line 313
      .line 314
      .line 315
      .line 316
      .line 317
      .line 318
      .line 319
      .line 320
      .line 321
      .line 322
      .line 323
      .line 324
      .line 325
      .line 326
      .line 327
      .line 328
      .line 329
      .line 330
      .line 331
      .line 332
      .line 333
      .line 334
      .line 335
      .line 336
      .line 337
      .line 338
      .line 339
      .line 340
      .line 341
      .line 342
      .line 343
      .line 344
      .line 345
      .line 346
      .line 347
      .line 348
      .line 349
      .line 350
      .line 351
      .line 352
      .line 353
      .line 354
      .line 355
      .line 356
      .line 357
      .line 358
      .line 359
      .line 360
      .line 361
      .line 362
      .line 363
      .line 364
      .line 365
      .line 366
      .line 367
      .line 368
      .line 369
      .line 370
      .line 371
      .line 372
      .line 373
      .line 374
      .line 375
      .line 376
      .line 377
      .line 378
      .line 379
      .line 380
      .line 381
      .line 382
      .line 383
      .line 384
      .line 385
      .line 386
      .line 387
      .line 388
      .line 389
      .line 390
      .line 391
      .line 392
      .line 393
      .line 394
      .line 395
      .line 396
      .line 397
      .line 398
      .line 399
      .line 400
      .line 401
      .line 402
      .line 403
      .line 404
      .line 405
      .line 406
      .line 407
      .line 408
      .line 409
      .line 410
      .line 411
      .line 412
      .line 413
      .line 414
      .line 415
      .line 416
      .line 417
      .line 418
      .line 419
      .line 420
      .line 421
      .line 422
      .line 423
      .line 424
      .line 425
      .line 426
      .line 427
      .line 428
      .line 429
      .line 430
      .line 431
      .line 432
      .line 433
      .line 434
      .line 435
      .line 436
      .line 437
      .line 438
      .line 439
      .line 440
      .line 441
      .line 442
      .line 443
      .line 444
      .line 445
      .line 446
      .line 447
      .line 448
      .line 449
      .line 450
      .line 451
      .line 452
      .line 453
      .line 454
      .line 455
      .line 456
      .line 457
      .line 458
      .line 459
      .line 460
      .line 461
      .line 462
      .line 463
      .line 464
      .line 465
      .line 466
      .line 467
      .line 468
      .line 469
      .line 470
      .line 471
      .line 472
      .line 473
      .line 474
      .line 475
      .line 476
      .line 477
      .line 478
      .line 479
      .line 480
      .line 481
      .line 482
      .line 483
      .line 484
      .line 485
      .line 486
      .line 487
      .line 488
      .line 489
      .line 490
      .line 491
      .line 492
      .line 493
      .line 494
      .line 495
      .line 496
      .line 497
      .line 498
      .line 499
      .line 500
      .line 501
      .line 502
      .line 503
      .line 504
      .line 505
      .line 506
      .line 507
      .line 508
      .line 509
      .line 510
      .line 511
      .line 512
      .line 513
      .line 514
      .line 515
      .line 516
      .line 517
      .line 518
      .line 519
      .line 520
      .line 521
      .line 522
      .line 523
      .line 524
      .line 525
      .line 526
      .line 527
      .line 528
      .line 529
      .line 530
      .line 531
      .line 532
      .line 533
      .line 534
      .line 535
      .line 536
      .line 537
      .line 538
      .line 539
      .line 540
      .line 541
      .line 542
      .line 543
      .line 544
      .line 545
      .line 546
      .line 547
      .line 548
      .line 549
      .line 550
      .line 551
      .line 552
      .line 553
      .line 554
      .line 555
      .line 556
      .line 557
      .line 558
      .line 559
      .line 560
      .line 561
      .line 562
      .line 563
      .line 564
      .line 565
      .line 566
      .line 567
      .line 568
      .line 569
      .line 570
      .line 571
      .line 572
      .line 573
      .line 574
      .line 575
      .line 576
      .line 577
      .line 578
      .line 579
      .line 580
      .line 581
      .line 582
      .line 583
      .line 584
      .line 585
      .line 586
      .line 587
      .line 588
      .line 589
      .line 590
      .line 591
      .line 592
      .line 593
      .line 594
      .line 595
      .line 596
      .line 597
      .line 598
      .line 599
      .line 600
      .line 601
      .line 602
      .line 603
      .line 604
      .line 605
      .line 606
      .line 607
      .line 608
      .line 609
      .line 610
      .line 611
      .line 612
      .line 613
      .line 614
      .line 615
      .line 616
      .line 617
      .line 618
      .line 619
      .line 620
      .line 621
```

845	<pre> bipush 95 dup getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println(C)V ; e1 right hand for assignment dup istore 0 pop .line 133 iload 15 iconst_3 if_icmpeq L0 iconst_0 goto L1 L0: iconst_1 L1: ifeq L2 ; e1 is false .line 134 bipush 77 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/println(C)V .line 134 bipush 97 getstatic java/lang/System/out Ljava/io/PrintStream; swap </pre>
850	
855	
860	
865	


```

870 invokevirtual java/io/PrintStream/println (C)V
    .line 134
        bipush 97
            ; ldc 'a'
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
    .line 134
        bipush 114
            ; ldc 'r'
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
    .line 134
        bipush 116
            ; ldc 't'
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
    .line 134
        bipush 112
            ; ldc 'p'
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        goto L2
    .line 136
        bipush 65
            ; ldc 'A'
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
    .line 136
        bipush 112
            ; ldc 'p'
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap

```

```

        invokevirtual java/io/PrintStream/println (C)V
    .line 136
        bipush 114                ; ldc 'r'
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
    .line 136
        bipush 105                ; ldc 'i'
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
    .line 136
        bipush 108                ; ldc 'l'
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
                                           ; e3 if false expression
    L3:
    .line 138
        iload 0                    ; load under from 0
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (C)V
        getstatic Main/jaar I ; load global jaar
        getstatic java/lang/System/out Ljava/io/PrintStream;
        swap
        invokevirtual java/io/PrintStream/println (I)V
        return

```

900

905

910

105

915

920

925 |||.end method