VERTALERBOUW
studiejaar 2011/2012 kw4
OPGAVENSERIE 1

Voor deze opgavenserie zijn **100** punten te behalen.
De opgaven mogen in het Nederlands of in het Engels beantwoord worden.

Deze opgavenserie dient **individueel** gemaakt te worden. Voorzie het eerste blad van de antwoorden van deze serie duidelijk van uw achternaam, voorletters, studentnummer en de naam van uw studentassistent (herhalers die geen studentassistent hebben dienen "HERHALER" te vermelden). *Voor het ontbreken van één van deze gegevens worden **10** punten in mindering gebracht.*

De antwoorden van deze opgavenserie dienen uiterlijk

woensdag 23 mei 2012 om **18.00 uur**

te worden ingeleverd in in het postvakje van Arend Rensink (Zilverling 5110) óf tijdens het hoorcollege.

Bovenstaande deadline voor het inleveren van de opgavenserie is **strict**. Op het te laat inleveren staan de volgende sancties:

- **één dag** te laat: **-10** punten voor deze serie;
- **twee dagen** te laat: **-20** punten voor deze serie;
- **drie dagen** te laat: **-50** punten voor deze serie;
- **nog later**: De serie geldt als niet gedaan.

Overigens is de regel bovendien dat het doen van beide huiswerkopgaven een voorwaarde is om in aanmerking te komen voor een eindcijfer van het vak.

Opgave 1 (15 punten)

Veronderstel dat de programmeertaal Triangle wordt uitgebreid met een `for`-expressie:

for *i*:Integer **upto** *c* **yield** *E*

Hier is i een (nog niet eerder gedeclareerde) variabele, c een constante, en E een expressie die i kan bevatten. Het resultaat van de expressie is een array bestaande uit de waarden die E oplevert wanneer i van 0 tot $c-1$ loopt. Een paar voorbeelden:

- **for** `i:Integer` **upto** `5` **yield** `i+1` levert de array `[1,2,3,4,5]` op.
- **for** `x:Integer` **upto** `3` **yield** `a[2-i]` (waarbij `a` een array van type **array** `3` **of** `T` is) levert de inverse van array `a` op.

Geef een definitie van de **for**-expressie in de stijl van hoofdstuk 1 van Watt & Brown (W&B), d.w.z. met beschrijvingen van de *syntax*, *contextbeperkingen* en *semantiek*.

Opgave 2 (15 punten)

Deze opgave gaat over het gebruik van grafsteen-diagrammen (*tombstone diagrams*) als in hoofdstuk 2 van W&B.

Scala is een functionele programmeertaal die vertaald kan worden naar Java bytecode (JBC). Stel, u heeft al een Scala-compiler geschreven in Scala (de ultieme bevrediging voor elke ontwerper van een nieuwe programmeertaal) die JBC oplevert. Verder heeft u een interpreter voor Scala op een x86-architectuur, en een Java Virtuele Machine die JBC kan uitvoeren op een SPARC-processor.

(a) Laat met behulp van grafsteen-diagrammen zien hoe een programma P geschreven in Scala kan worden vertaald naar een programma dat kan worden uitgevoerd op een SPARC, als u een x86-machine tot uw beschikking hebt.

(b) We willen ook Scala-programma's naar SPARC-machinetaal kunnen vertalen, eventueel in meerdere fasen, maar dan wel zo dat deze vertaling zelf ook direct op een SPARC-machine kan worden uitgevoerd (zonder tussenkomst van een interpreter). Welke extra vertaler(s) moeten ontwikkeld worden zodat dit mogelijk is? Geef met behulp van grafsteen-diagrammen aan hoe de vertaler(s) gebouwd worden, onder inachtneming van de volgende criteria:

- De te ontwikkelen vertalers moeten een zo klein mogelijke vertaalstap maken;
- De te ontwikkelen vertalers moeten zelf geschreven zijn in een zo "hoog" mogelijke programmeertaal, bij voorkeur weer Scala.

Opgave 3 (20 punten)

De Triangle-vertaler zet onderstaand programma aan de linkerkant om in de TAM-code aan de rechterkant.

```
let var i: Integer;
    const b ~ 10
in while i < b
   do i := i+1
```

```
 0:  PUSH        1
 1:  JUMP        6[CB]
 2:  LOAD  (1)   0[SB]
 3:  LOADL       1
 4:  CALL        add
 5:  STORE (1)   0[SB]
 6:  LOAD  (1)   0[SB]
 7:  LOADL       10
 8:  CALL        lt
 9:  JUMPIF(1)   2[CB]
10:  POP   (0)   1
11:  HALT
```

Beschrijf bovenstaande vertaling op dezelfde manier als de voorbeelden 3.1, 3.2 en 3.4 van W&B. Net als bij deze voorbeelden dienen de beschrijvingen geïllustreerd te worden met (versierde) ASTs. (De codes `JUMP` en `JUMPIF` staan niet in hoofdstuk 3; zie daarvoor de appendix.)

Opgave 4 (20 punten)

Welke van de volgende grammatica's zijn LL(1)? Motiveer uw antwoorden met de methode van het hoorcollege, dat wil zeggen gebruik makend van de verzamelingen *first*, *follow* en *lookahead*.

(a) De volgende grammatica beschrijft de taal van geneste haakjes, als a voor '(' en b voor ')' staat.

$$S \to aSbS \mid \varepsilon$$

(b) De volgende grammatica is equivalent met die hierboven, d.w.z., beschrijft dezelfde taal:

$$S \to FaSb \mid \varepsilon$$
$$F \to S \mid \varepsilon$$

(c) De volgende grammatica beschrijft de taal van **`if`**–**`then`**-statements met optionele **`else`**, waarbij i voor **`if`** staat, t voor **`then`** en e voor **`else`**:

$$S \to ITE \mid \varepsilon$$
$$I \to iS$$
$$T \to tS$$
$$E \to eS \mid \varepsilon$$

Opgave 5 (15 punten)

(a) In het boek van W&B worden AST-nodes gerepresenteerd d.m.v. Java-klassen uit een klassenhiërarchie met de klasse `AST` als (abstracte) superklasse. Welke representatie is hiervoor in Antlr gekozen, i.h.b., hoe worden verschillende soorten AST-nodes onderscheiden? (Bestudeer hiervoor een door Antlr gegenereerde `XxxLexer.java`.) Welke andere representatie(s) kunt u bedenken? (Geef er minstens één.) Wat zijn de voor- en nadelen van de verschillende representaties?

(b) In week 1 van het practicum hebt u een `SymbolTable` geprogrammeerd die (als u de suggestie in de slides hebt gevolgd) gebruik maakte van velden

```
private Map<String,Stack<Attribute>> symtab;
private Stack<List<String>>          scopeStack;
```

Geef een alternatieve implementatie die in plaats daarvan gebruik maakt van (uitsluitend) een veld

```
private Stack<Map<String,Attribute>> symtab;
```

Hoe verhoudt zich de efficiëntie van deze implementatie tot die volgens de slides?

Opgave 6 (15 punten)

Beschouw de volgende Triangle *record*-typedeclaraties:

```
type T1 ~ record i: Integer, c: Char end;
type T2 ~ record b: Boolean, i: Integer end
```

Bestudeer de code in `Triangle.AbstractSyntaxTrees` om u een beeld te vormen van de structuur van de ASTs die Triangle gebruikt om een `RecordType` te representeren.

(a) Schrijf vervolgens een Antlr-grammatica die, gegeven een serie declaraties van bovenstaande vorm (dat wil zeggen, **`type`**-declaraties met een willekeurig aantal velden van een primitief type), een AST met precies dezelfde structuur oplevert.

(b) Geef (met behulp van `org.antrl.runtime.tree.DOTTreeGenerator` en GraphViz) een screenshot van de door Antlr opgeleverde AST van bovenstaande code.