Hertentamen Vertalerbouw—16 augustus 2002

De gecorrigeerde tentamens zijn af te halen op de begane grond in de tentamenkast.

Opmerkingen:

- Schrijf **netjes** en duidelijk, met zwarte of blauwe pen.
- Zet op het eerste blad alle gegevens als naam, etc., en het totaal aantal ingeleverde bladen, en nummer de ingeleverde bladen.
- Lees de opgaven eerst goed door.
- Motiveer uw antwoorden.
- De opgaven zullen gewogen meetellen in het totaalcijfer, volgens de vermelde bestedingstijd.

1. (40 minuten)

   a). Geef First en Follow voor alle nonterminals in de navolgende grammatica:
   $G = (\{b, c\}, \{A, B, C\}, P, A)$, met

$$
\begin{array}{lcll}
P = \{ & A & \rightarrow & B\ B\ C \\
, & B & \rightarrow & b\ C \\
, & B & \rightarrow & b \\
, & C & \rightarrow & \\
, & C & \rightarrow & c\ A\ B \\
\} & & &
\end{array}
$$

   b). Is de grammatica $LL(1), LR(0), SLR(1), LALR(1)$ of $LR(1)$? Geef in geval van conflicten deze duidelijk aan, en leg uit waarom dit conflicten zijn.

2. (45 minuten)

   Gegeven is een eenvoudig taaltje, dat syntactisch gespecificeerd wordt door:

$$
\begin{array}{lcll}
 & \textit{Decls} & \rightarrow & \textit{Decl Decls} \\
, & \textit{Decls} & \rightarrow & \\
, & \textit{Decl} & \rightarrow & \textit{var Field} \\
, & \textit{Field} & \rightarrow & \textit{ident} : \textit{Type} ; \\
, & \textit{Type} & \rightarrow & \textit{int} \\
, & \textit{Type} & \rightarrow & \textit{real} \\
, & \textit{Type} & \rightarrow & \textit{record Fields end} \\
, & \textit{Fields} & \rightarrow & \textit{Field Fields} \\
, & \textit{Fields} & \rightarrow &
\end{array}
$$

Gegeven is verder dat een `int` 4 bytes memory kost, en een `real` 8 bytes. Het is de bedoeling de syntaxregels te voorzien van attributen en reken-voorschriften, zodanig dat van elke variabele declaratie wordt berekend hoeveel (memory) bytes nodig zijn. Daarnaast dient de totale hoeveelheid ruimte in bytes (voor alle declaraties tesamen) afgedrukt te worden.

Je mag daarbij gebruik maken van de volgende globale declaraties:

```
sizes : array [1..maxint) of integer;
n : integer = 0; (* sizes[1..n] is ingevuld *)
```

3. (50 minuten)

In een uitbreiding van Pascal zijn naast de value en var parameters ook de result parameters toegestaan. Ter herinnering: de variabele, die op een argumentplaats als result wordt meegegeven, krijgt pas zijn waarde toegekend aan het einde van de procedure–body.

In deze uitbreiding van Pascal is het volgende 'programma' gegeven:

```
PROGRAM tentamen;

CONST upb = 10;

VAR a, b: integer;

PROCEDURE p1;
  VAR b, c: integer;
      d   : ARRAY [1..upb] OF integer;

  PROCEDURE p2 (VAR x: integer);
    VAR a, b: integer;
    BEGIN ...
          b := x + c;                    (* 1 *)
          ...
    END (* p2 *);

  PROCEDURE q2 (RESULT x: integer);
    VAR c, e: integer;
    BEGIN e := b;
          ...
          x := c + d[e];                 (* 2 *)
    END (* q2 *);

  BEGIN ...
        p2 (b);                          (* 3 *)
        ...
        q2 (d[b]);                       (* 4 *)
        ...
  END (* p1 *);
```

```
BEGIN ... p1; ...
END (* tentamen *).
```

Voor het geheugenbeheer en de adresberekeningen worden de volgende registers gebruikt:

`GP` het base address van het activation record van het hoofdprogramma,

`LNB` het base address van het huidige activation record, en

`LFA` het adres van de eerste vrije geheugenlokatie.

Voor het overdragen van de omgeving van een aan te roepen procedure kan het register `ENV` worden gebruikt.

In de machineinstructies `CALL lable` en `RETURN` van de doelmachine wordt impliciet gebruik gemaakt van een (aparte) return stack. U hoeft zich dus niet druk te maken over terugkeer-adressen!

Er zijn voldoende registers (R0, R1, R2, ...) voor het opslaan van de tussenresultaten.

a) Geef de layout van de activation records van $p1$, $p2$ en $q2$.

b) Geef de te genereren (pseudo-)instructies voor de procedure-entry en exit van $p1$ en $q2$.

c) Geef de te genereren (pseudo-)instructies voor de 4 gemarkeerde statements. Controle op index–waarden, die buiten array grenzen gaan, is niet nodig!

4. (45 minuten)

a). Controleer of de grammatica uit som 2 $LL(1)$ is, en, als dit niet zo is, herschrijf deze dan tot een $LL(1)$ grammatica.

b). Schrijf een topdown recursive descent parser, die deze $LL(1)$ beschrijving accepteert. De parser moet fouten in de invoer signaleren.

Onderstaande declaraties mogen worden gebruikt, danwel aangepast, in de implementatie van de parser. Alle overige zaken dient U volledig te declareren.

```
TYPE
  symbol     = (Decls, Decl, Type, Fields, Field,
                semicolonsym, colonsym, varsym, identsym,
                intsym, realsym, recordsym, endsym, eofs);
  tsymbol    = semicolonsym..eofs;
  tsymbolset = SET OF tsymbol;
VAR
  sym: tsymbol;

PROCEDURE initscanner;
  (* Initialisatie van de scanner *)
```

```
PROCEDURE nextsym;
  (* Levert bij aanroep de tokenwaarde op (in de variabele
     sym) van het eerstvolgende symbool in de invoer *)

FUNCTION first (s: symbol): tsymbolset;
  (* retourneert de first set van symbool s *)

PROCEDURE error (sy: tsymbol;  str: string);
  (* Genereert een foutmelding in de vorm:
     "representatie van sy"++"waarde van str" *)
```