Hertentamen Vertalerbouw—4 februari 2003

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:
   $G = (\{a, b, c, d\}, \{A, B, C\}, P, A)$, met

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

   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 en hoe ze (indien mogelijk) opgelost worden.

2. (45 minuten)

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

$$
\begin{array}{lll}
Test & \rightarrow & E\ ==\ E \\
E & \rightarrow & T\ +\ E \\
E & \rightarrow & T \\
T & \rightarrow & F\ *\ T \\
T & \rightarrow & F \\
F & \rightarrow & (\ E\ ) \\
F & \rightarrow & intdenotation \\
F & \rightarrow & realdenotation \\
F & \rightarrow & Var \\
Var & \rightarrow & identifier \\
Var & \rightarrow & identifier\ [\ E\ ]
\end{array}
$$

Het is de bedoeling de syntaxregels te voorzien van attributen en rekenvoorschriften, zodanig dat van elke equality test de typecorrectheid gechecked wordt. Hierbij moet typegelijkheid gelden. Bij een array dient de indexexpressie altijd het type integer te hebben.

Je mag hierbij gebruik maken van de elders gedefinieerde en geïmplementeerde begrippen

```
type typeTp = (intTp, realTp);
var Sym: tsymbol;   { current scanned symbol }
    SymId: integer; { index of current symbol in symboltable }
function getType (id: SymId): typeTp;
{ gegeven de index in de symboltable retourneer het
  bijbehorend type. Je mag aannemen dat alle identifiers
  een gedefinieerd type hebben.
}
```

3. (50 minuten)

Gegeven is het volgende (Pascal-) programma:

```
PROGRAM tentamen;

VAR a, b: integer;

PROCEDURE p1 (y: integer; procedure fp (VAR i: integer));
  VAR a : integer;

  PROCEDURE p2 (VAR x: integer);
    VAR b: integer;
    BEGIN ...
          fp(b);                              (* 1 *)
          ...
    END (* p2 *);

  PROCEDURE q2 (VAR x: integer; y: integer);
    VAR c: integer;
    BEGIN ...
          x := c*y + a*x;                     (* 2 *)
          ...
    END (* q2 *);
```

```
  BEGIN ...
        p2 (a);                                  (* 3 *)
        ...
        q2 (b,a);                                (* 4 *)
        ...
  END (* p1 *);

BEGIN ...  p1(a,p2);  ...                        (* 5 *)
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) Leg duidelijk uit hoe je een formele procedure (parameter) in de stack representeert.

b) Geef de layout van de activation records van $p1$ en $q2$.

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

d) Geef de te genereren (pseudo-)instructies voor de 5 gemarkeerde statements.

4. (45 minuten)

Bekijk opnieuw de grammatica uit opgave 2:

a). Verander deze grammatica zodanig dat deze voldoet aan de $LL(1)$ eis (laat wel de beschreven taal gelijk!).
b). Schrijf een topdown recursive descent parser, die aan error recovery doet. [U kunt een bonuspunt verdienen wanneer U de semantische akties uit opgave 2 ook meeneemt!]

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

```
TYPE
  symbol     = (Test, E, T, F, Var, identifier, ... endoffile);
  tsymbol    = identifier .. endoffile;
  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 *)

FUNCTION follow (s: symbol): tsymbolset;
  (* retourneert de follow set van een nonterminal s *)
```