Hertentamen Vertalerbouw—23 augustus 2007

- 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.

1. a) Geef voor alle nonterminals uit onderstaande produkties de sets *first* en *follow*.

   b) Is de grammatica (gegeven door de volgende produkties) $LL(1)$, $LR(0)$, $SLR(1)$ en/of $LR(1)$? Geef in geval van conflicten deze duidelijk aan.

   $$\begin{array}{rcl} S & \rightarrow & AbSB \mid AB \\ , \; A & \rightarrow & aA \mid b \\ , \; B & \rightarrow & bB \mid \end{array}$$

2. Gegeven is het volgende (Pascal-achtige) programma:

```
PROGRAM tentamen;

TYPE rij = ARRAY [5..8] of integer;

VAR i,j: integer;
    a: rij;

FUNCTION f (j: integer): integer;
BEGIN ...
      f := a[j]+a[i]                          (* 1 *)
END;

PROCEDURE p (i: integer; VAR r: integer);
  PROCEDURE q (VAR s: integer);
  BEGIN r := s*f(i)                           (* 2 *)
  END;
BEGIN r := a[j]+i;                            (* 3 *)
      ...
      q(r)                                    (* 4 *)
END;

BEGIN ...
      p(7,j);                                 (* 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.

Een **functie** implementeren we precies zo als een procedure, behalve dat een functie ook nog een resultaatwaarde oplevert (door een assignment aan de functie-identifier). De handigste plaats in de stack voor dit functieresultaat is nog vóór de parameters (dus ook met een negatief displacement tov. de `LNB`).

a) Geef de layout van de activation records van f en q.

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

c) Geef de te genereren (pseudo-)instructies voor de 5 gemarkeerde statements. Controle op index–waarden, die buiten array grenzen gaan, is niet nodig!

3. (40 minuten)

Zij $G = (T,\ N,\ Z,\ P)$ een uitgebreide contextvrije grammatica met

$$T = \{assign, beg, end, fi, ident, if, lpar, number, op, rpar, sc, then, eof\}$$

$$N = \{S, Si, Sl, St, E, T, X\}$$

$$\begin{array}{lrcl} P = \{ & S & \rightarrow & Si \mid if\ E\ then\ Si\ fi \quad \} \\ , & E & \rightarrow & T\ X \\ , & T & \rightarrow & number \mid ident \mid lpar\ E\ rpar \\ , & X & \rightarrow & op\ T\ X \mid \\ , & Si & \rightarrow & assign \mid beg\ Sl\ end \\ , & Sl & \rightarrow & S\ St \\ , & St & \rightarrow & sc\ S\ St \mid \end{array}$$

(De terminal `sc` staat voor de semicolon, de terminal `op` kan bijvoorbeeld een gelijkheidstest zijn.)
De grammatica G is $LL(1)$.

a) Geef de default productieregels aan.
b) Gevraagd is een recursive descent parser zonder error-recovery voor deze grammatica.

U mag gebruik maken van de volgende declaraties:

```
TYPE
  tsymbol    = (assign, beg, end, fi, ident, if,
                lpar, number, op, rpar, sc, then, eof);
  tsymbolset = SET OF tsymbol;

VAR
  sym: tsymbol;

PROCEDURE initscanner;
  (* Initialisatie van de scanner *)
  BEGIN  ...  END (* initscanner *);

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

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