Hertentamen Vertalerbouw—11 maart 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. (45 minuten)

   a). Geef First en Follow voor alle nonterminals in de navolgende grammatica:
   $G = (\{w, z, j\}, \{A, N, M\}, P, A)$, met

$$
\begin{array}{lcll}
P = & \{ & A & \rightarrow & w\ N\ A \\
 & , & A & \rightarrow & w\ N \\
 & , & N & \rightarrow & z\ M\ N \\
 & , & N & \rightarrow & z\ M \\
 & , & M & \rightarrow & j\ A\ N\ M \\
 & , & M & \rightarrow & \\
 & \} & & &
\end{array}
$$

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

2. (40 minuten)

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

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

   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 rekenvoorschriften, zodanig dat van elke 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)

Gegeven is het volgende Pascal 'programma':

```
PROGRAM tentamen;

TYPE arr = ARRAY [1..10] of integer;

VAR i,j: integer;
    a: arr;

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

PROCEDURE p (j: integer; r: integer);
  VAR k: integer;

  PROCEDURE q (VAR n: integer);
  BEGIN n := r * f(j)                                  (* 2 *)
  END;

BEGIN r := a[i] * j;                                   (* 3 *)
      ...
      q(k)                                             (* 4 *)
      ...
END;

BEGIN ...
      p(i,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`). Het is daardoor de verantwoordelijkheid van de caller om deze ruimte te creëren.

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!

4. (45 minuten)

Schrijf een topdown parser met expliciete stapel, die de taal accepteert als beschreven door de grammaticaregels van opgave 2. Deze parser moet in staat zijn om bij fouten in de invoer error recovery uit te voeren.

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, 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" *)
```

Verder is gegeven een abstract data type (ADT) stack:

```
PROCEDURE initstack;
  (* Creeert een lege stack *)

FUNCTION isempty: boolean;
  (* checks if stack is empty *)

FUNCTION top: symbol;
  (* returns the symbol on top of the stack *)

PROCEDURE push (s: symbol);
  (* pushes s onto the stack *)

PROCEDURE pop;
  (* pops one symbol from the stack *)

PROCEDURE copystack;
  (* maakt een kopie van de inhoud van de stack, *)
  (* dit maakt de weg vrij om van alles met de stack te doen *)

PROCEDURE recoverstack;
  (* herstelt de stack mbv. de kopie; *)
  (* na afloop van deze operatie is de stack exact gelijk aan *)
  (* het moment van de aanroep van de laatste copystack *)
```