Tentamen Vertalerbouw—5 november 2003

De gecorrigeerde tentamens zijn af te halen op het Onderwijsbureau, 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 de nonterminals in de navolgende grammatica:
   $G = (\{a, b, c\}, \{S, A, B, C\}, P, S)$, met

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

   b). Ga na of G $LL(1), LR(0), SLR(1), LALR(1)$ of $LR(1)$ is. 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:

$Z \rightarrow E\ eofsym$
$E \rightarrow H\ E\ H\ E$
$E \rightarrow$
$H \rightarrow lbrack$
$H \rightarrow rbrack$
$H \rightarrow lpar$
$H \rightarrow rpar$

Deze grammatica beschrijft strings met 'haakjes'. Echter, dit vinden we iets te ruimhartig, we willen alleen 'nette' strings toestaan. De volgende 3 strings zijn 'netjes':
`([()]()())  () (())[]`
In de volgende 3 strings 'matchen' de haakjes op een 'onnette' manier:
`([(]))  )(  (]()`

Voorzie de syntaxregels van attributen en rekenvoorschriften (dwz. programmafragmenten), zdd. alleen een 'nette' string geaccepteerd wordt. Andere invoer moet een errormessage opleveren.

Je mag hierbij, indien nodig, gebruik maken van de volgende begrippen:

```
type symbol  = (lbrack, lpar, rbrack, rpar, eofsym, Z, E, H);
type tsymbol = lbrack .. eofsym; (* subrange van symbol *)
type HtypeTp = (open, close);    (* nog een enumeratie type *)
type H_Tp    = lbrack .. rpar;

var Sym: tsymbol;   { current scanned symbol }

procedure error(s: string); { print s en stopt de parsing }
```

Alle overige benodigde datastructuren en code dient volledig beschreven te worden.

3. (45 minuten)

Gegeven is het volgende (Pascal-achtige) programma:

```
PROGRAM tentamen;

TYPE rij = ARRAY [0..6] of integer;

VAR a, b: integer;
    c: rij;

    PROCEDURE p1 (a: integer; VAR d: integer);
    BEGIN ...
          d := a + b;                          (* 1 *)
          ...
          c[d] := d * 2;                       (* 2 *)
          ...
    END (* p1 *);

    PROCEDURE p2 (VAR a: integer);
      PROCEDURE p3 (VAR b: integer);
        VAR a: integer;
      BEGIN ...
            p1(c[a],b);                        (* 3 *)
            ...
      END (* p3 *);
    BEGIN ...
          p3(a);                               (* 4 *)
          ...
    END (* p2 *);

BEGIN ...  p2(a);  ...                         (* 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 label` 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. Verder mag u aannemen dat integer en real waarden beide 1 geheugenplaats in beslag nemen.

a) Geef de layout van alle activation records.

b) Geef de te genereren (pseudo-)instructies voor de procedure-entry en exit van $p3$.

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

4. (50 minuten)

Gegeven is de grammatica G' hieronder.

$$G' = (\{l, r\}, \{E, O, C\}, P, E)$$

$$\begin{array}{lllll} P = & \{ & E & \rightarrow & O\ E\ C\ E \\ & , & E & \rightarrow & \\ & , & O & \rightarrow & l \\ & , & C & \rightarrow & r \\ & \} & & & \end{array}$$

a) Laat zien dat deze grammatica $LL(1)$ is.
b) Schrijf een recursive descent parser, die aan error recovery doet.

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

```
TYPE
  symbol     = (E, O, C, l, r, eofsym);
  tsymbol    = l .. eofsym;
  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 *)

PROCEDURE deletion;
  (* levert foutmelding: 'Sym' deleted *)

PROCEDURE insertion (fs: tsymbol);
  (* levert foutmelding: 'fs' inserted *)
```